A Key Loss Recovery Scheme for Secure Broadcasts in Wireless Sensor Networks

Syed Taha Ali*, Vijay Sivaraman*, Ashay Dhamdhere*, Diethelm Ostry†
*School of Electrical Engineering and Telecommunications, University of New South Wales, Sydney, Australia.
Emails: taha@student.unsw.edu.au, {vijay, ashay}@unsw.edu.au
†ICT Centre, CSIRO, Sydney, Australia. *Email: Diet.Ostry@csiro.au*

***Abstract*—Authenticity and secrecy of broadcast message content is important in wireless sensor networks deployed for battlefield control, emergency response, and natural resource management. Encryption of broadcast data requires the key to vary in time, typically via a key chain, so that a key compromised at a receiver does not compromise broadcast security for the entire network. An unfortunate consequence of time-varying keys is that a receiver that misses (due to packet loss) one or more keys from the chain cannot decrypt subsequent messages, thereby getting excluded from all broadcasts. In this paper we develop a scheme that allows receivers to recover from one or a few lost keys by having the transmitter probabilistically reuse old keys from the chain. Our scheme makes the broadcast system more robust to packet loss, at the expense of increasing vulnerability to compromised old keys. Analysis of our scheme shows how the trade-off can be controlled by tuning parameters, and a prototype implementation on a MicaZ mote testbed demonstrates the feasibility of our scheme in real sensor network platforms.**

I. Introduction

Large-scale wireless sensor networks deployed for battlefield control and resource management are expected to use broadcasting capability for operations such as command dissemination and network query. Securing such broadcast messages is important, and has been gaining attention in the research literature [1]. Central to broadcast security is *authentication* [2] of the broadcast source and data. Additionally, *privacy* of the broadcast data is mandated in certain applications, such as battlefield messaging and natural resources monitoring [3]. Scalable solutions for ensuring authenticity and privacy of broadcast data pose a challenge given that wireless sensor nodes have very limited resources. Asymmetric cryptographic techniques for encryption and digital signatures are impractical due to their high computation and communication overheads. Symmetric encryption that uses a common *static key* shared by all broadcast recipients is problematic since key compromise at one node compromises the entire network, and the likelihood of key compromise increases with the number of receivers. Scalable solutions for broadcast security therefore need to use symmetric cryptography with *time-varying keys*.

The idea of time-varying keys for broadcasts was used in [4] for authentication and by us in [5] for encryption. The central idea is that the key changes in time according to a "key chain", where each key in the chain hashes to the previous key in the chain; the one-way nature of hash functions ensures that a received key can be verified but not generated from the previous key in the chain. Assuming the initial "root" key of the chain is known to all legitimate receivers (either statically programmed at deployment time or dynamically conveyed by the transmitter via unicast mechanisms), upon each key change the new key is included by the transmitter in the broadcast message. Receivers decrypt the broadcast message using the key they hold, and accept a new key only if it hashes to their existing key, thereby ensuring authenticity. Keys can be varied slowly once every pre-determined time period [4], or from one packet to the next [5]. Time-varying keys offer several important advantages in the context of broadcast: 1) Receivers need perform only hashing (e.g. MD5, SHA1) and symmetric-key decryption (e.g. RC5), which are **computationally less intensive** than public-key cryptography and/or digital signatures; 2) Key compromise at a receiver does not allow the intruder to deduce subsequent keys in the chain needed for forging broadcast messages, and **authenticity** is thus guaranteed; 3) Intruders get a reduced window of time in which to crack a key before the key changes, thereby needing increased computational/storage capability to compromise broadcast **privacy**.

The strong authenticity and privacy protection offered by key chain schemes have the unfortunate effect of reducing system robustness to packet loss. A receiver that has missed a key cannot decrypt broadcast messages to extract the next key (even if extracted, the received key would not validate), in effect making the receiver no better off than an intruder who has no keys. In this paper, we propose a scheme that addresses this shortcoming. The basic idea is for the transmitter to include in broadcast packets some "recovery information" that probabilistically reuses old keys of the chain to allow receivers that have lost one or a few keys to recover. The likelihood of recovery diminishes with the number of missing keys, thereby allowing system robustness (ability of a receiver to recover from a small number of missing keys) to be balanced against system vulnerability (ability of an intruder to attach using an old compromised key). We analyse our scheme to show how this balance can be controlled by tuning the probability of old key selection. We also prototype our recovery scheme on a MicaZ mote based platform to demonstrate feasibility, and show that its performance in experiments matches our analysis. We believe that our scheme represents a practical, scalable, and controllable way of improving robustness of broadcast security in sensor networks for battlefield control, resource monitoring, and other critical applications.

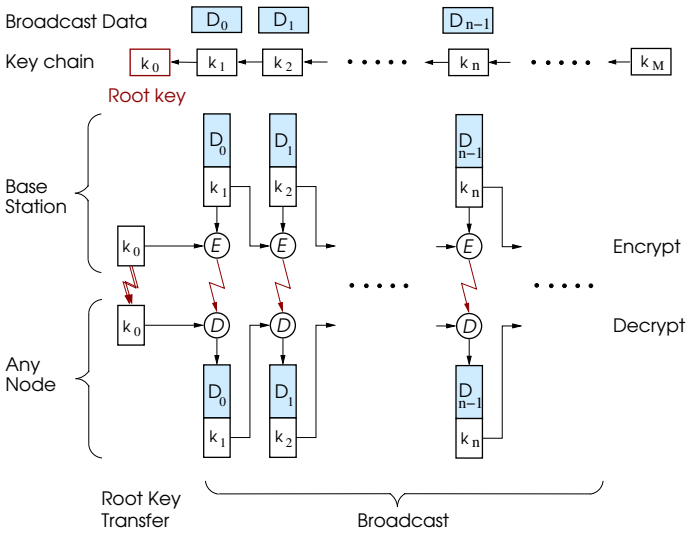

Fig. 1. Key chain encryption of broadcast packets

The rest of this paper is organised as follows: Section II describes key chain based encryption of broadcast messages. In Section III we describe our proposal for recovery from lost keys, and Section IV analyses its performance. Section V describes our prototype implementation and experimental results, and the paper concludes in Section VI with pointers to future work.

II. Background

In this section we briefly summarise key chain based encryption and authentication of broadcast data presented by us at PIMRC last year [5]. The main steps are described below and shown in Fig. 1:

1) **Key-chain generation:** The transmitter generates a key chain by first choosing an arbitrary random key k_M, and hashing successively to obtain k_{M-1}, ..., k_1, k_0, i.e. $k_{i-1} = H(k_i)$ for $i = 1, \ldots, M$, where $H(.)$ can be SHA1 or MD5. The length M of the chain can be arbitrarily large (allowing the chain to be used for broadcasting as many as M data packets), but in practice the key-width will limit the number of unique keys obtained by hashing.
2) **Bootstrapping:** The root key k_0 is securely conveyed to each receiver, either statically at deployment time or via one of several dynamic key management schemes [6].
3) **Data transmission:** Once all receivers have the root key, the transmitter creates the first broadcast packet by concatenating the broadcast data and the successor key k_1, and encrypts the entire message with a symmetric encryption technique (e.g. RC5) using key k_0. So that the encrypted data and encrypted key are not separable in ciphertext, an appropriate block cipher scheme such as offset code-book (OCB) mode [7] should be used. The encrypted packet is then broadcast to all nodes.
4) **Data reception:** A receiver sensor node can decrypt the message using key k_0 (which it already holds) to reveal the broadcast data as well as the successor key k_1. It then tests whether $H(k_1) = k_0$: if so, authenticity and integrity of the packet's source and data is assured and the packet is accepted. The key k_0 is now discarded by the node and the new key k_1 stored in its place.
5) **Iterate:** Steps 3 and 4 are repeated for successive broadcast packets, using key k_i in lieu of k_0, and k_{i+1} in lieu of k_1 for $i = 1, 2, \ldots$. Care must be taken that successive packets are transmitted at a rate which gives nodes sufficient time to extract the data payload and prepare for the next packet. Once all the M available keys have been used up by the transmitter, it has to return to step 1 to generate a new key chain before secure broadcast transmission can continue.

We note here some aspects of the above key chain based broadcast security scheme:

- **Authenticity** of broadcast source and data is guaranteed by virtue of the non-forgeability of the successor key, together with the fact that the encryption mode ensures that the key and data are inseparable in ciphertext. Authenticity is guaranteed even if one or more sensor nodes in the network are compromised.
- **Confidentiality** of broadcast data is guaranteed only if no key is compromised. An intruder trying to ascertain the key corresponding to a broadcast packet either has to crack the key before the next broadcast transmission (since the key will change), or store all packets transmitted while it is trying to crack the key (so that it can progress along the chain to "catch up" with the key in the chain currently being used by the transmitter); otherwise the cracked key will be useless for decrypting subsequent broadcast messages. Assuming that breaking the encryption to obtain a single key takes a non-trivial amount of time, only sophisticated intruders with large computation or storage capacity will be able to effectively attach to the broadcast session.
- The broadcast encryption scheme has some other attractive properties: it assures **freshness** of broadcast data since packet captured and replayed by an intruder would not contain the valid time-varying key and would be discarded by the sensor nodes. The scheme also ensures **semantic security**, namely identical data encrypted at separate time instants yields different ciphertext – this is useful in sensor networks where the broadcast messages are chosen from a small set, and the encrypted ciphertext should not give information to an intruder about which of these messages was sent. Additionally, the scheme allows **dynamic broadcast data**, namely, the broadcast data need not be known by the transmitter in its entirety before packet transmissions start; this makes the scheme efficient not just for broadcast file transfers (e.g. a new code image), but also for short dynamic broadcast messages (e.g. battlefield commands). Lastly, the scheme allows **incremental processing** whereby each received packet can be immediately verifiable without having to wait for additional data.

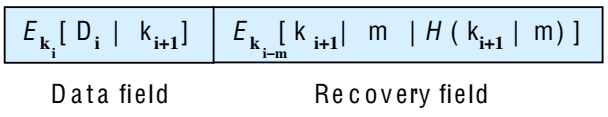

Fig. 2. Packet structure augmented to include recovery information

- The scheme currently operates in **single-hop networks**, and requires no time synchronisation in the system. In multi-hop networks checks are required to ensure that a transit node does not hold back several packets, extract the keys, and use them to generate broadcast packets containing malicious data but valid keys, which would be accepted by receivers downstream. We have extended our original scheme to multi-hop networks in [8], but restrict the current work in this paper to single-hop networks.
- The scheme has **no resilience to key loss**, which is the problem addressed by this paper, and discussed in the next section.

III. Loss Recovery Scheme

The key chain approach of [5] described above has the problem that a receiver which misses even one broadcast packet is effectively excluded from all future broadcast messages, since the key contained in the missing packet is needed to decrypt the subsequent packet, which in turn contains the key to the next packet, and so on. This is not a problem in applications that perform reliable delivery of broadcast data (e.g. network programming protocols), since lost packets will be retransmitted as part of the protocol and lost keys recovered therein. However, there are applications in which reliable delivery of broadcast data is unnecessary or prohibitive in cost. For example, consider a group of soldiers each of whom is equipped with a communication device receiving broadcast command and control data from a base-station (say a satellite or unmanned aerial vehicle). In such an application it is infeasible to make the broadcast reliable since the base-station may not know how many receivers are reachable at any time (some receivers may be inoperational or out of range), and moreover, it may be unwise to have receivers reveal their location by transmitting requests for missing data. In such unreliable broadcast scenarios, the loss of data in the packet may not be very crucial (the base-station may periodically repeat the data), but the loss of the key contained in the packet is a problem, since the encryption scheme does not reuse keys.

We propose a scheme for lost key recovery that operates as follows. Recall that the base-station in each broadcast packet P_i sends data D_i and the successor key k_{i+1}, together encrypted using the current key k_i. In addition, we include in packet P_i the following "recovery information" (see Fig. 2): the next key k_{i+1}, an integer $m \geq 1$, and the hash digest $H(k_{i+1}|m)$, the entire recovery information being encrypted with an *older* key k_{i-m} of the chain. The idea is to allow a node that has missed m previous broadcast packets to use the key it holds to jump forward in the chain and recover the next key to be used.

```
// current_key denotes the node's last correct key
1.  decrypt data_field of P_i using current_key to
        obtain data and extracted_key
2.  if extracted_key hashes to current_key // no key loss
3.      replace current_key with extracted_key
4.      process data
5.  else // keys may have been missed
6.      decrypt recovery_field of P_i using current_key
            to obtain k_{i+1}|m and recovery_hash
7.      if hash of k_{i+1}|m matches recovery_hash
8.          separate k_{i+1}|m into extracted_key and m
9.          hash extracted_key m+1 times
                and store in trial_key
10.         if trial_key matches current_key
11.             replace current_key with extracted_key
12.         else discard packet // cannot authenticate key
13.     else discard packet // decryption unsuccessful
14. end
```

Fig. 3. Operations performed by node upon arrival of broadcast packet P_i

Fig. 3 shows in pseudocode what a node does upon receipt of packet P_i. Steps 1-4 describe regular packet processing in the absence of packet loss. If the key chain validity check in step 2 fails, the node could have potentially lost previous broadcast packets, and recovery is attempted in steps 5-13. The node does not know which old key in the chain is used by the base-station for encrypting the recovery information (since it neither knows the number of packets it has missed, nor the number m chosen by the base-station); consequently the decryption in step 6 that uses the node's stored key may be unsuccessful (i.e. yield nonsense), and step 7 is needed to verify this by checking the contained hash. If correct, the successor key k_{i+1} is authenticated by hashing it $m+1$ times (step 9) to verify (in step 10) that it belongs to the key chain, and is then accepted (step 11), at which point the node has successfully reattached to the broadcast session. The packet is discarded if the key does not authenticate (step 12) or if the decryption was unsuccessful (step 13), which happens when the base-station has used a different key for encryption than the key held by the receiving node, or when the packet is malicious.

Our scheme has two attractive **properties**: 1) receivers that do not need recovery are not penalised (beyond the cost of receiving the recovery field), since they will satisfy the check in step 2 and ignore the recovery information. Further, receivers requiring recovery will spend a computation time at most linear in the number of lost pckets (step 9), and malicious packets will be dropped after a single decryption (step 6) and hash (step 7) operation; 2) our recovery scheme requires local computation at the receivers but no radio transmissions; this makes the scheme scalable to a large number of receivers, and attractive in scenarios where node location is required to remain hidden.

With the above scheme, a receiver that has missed m packets

(since its last success) can reattach using the next correctly received broadcast packet only if the base-station has picked the same number m for constructing the recovery information in that packet (otherwise the receiver cannot decrypt the recovery information). An important question therefore concerns the choice of m that the base-station should make, given absence of any knowledge of how many packets each of the (potentially large number of) receivers has missed (in fact a receiver itself may not know how many packets it has lost). If m is chosen as a small constant, a node that has lost $j > m$ packets can never reattach, since its last key k_{i-j} cannot decrypt the recovery information in packet P_i or any subsequent packet. If m is chosen to be a large constant, a node that has lost $j \ll m$ packets either has to wait for $m - j$ subsequent broadcast packets to pass before it can reattach, or spend much computational effort in trying to decrypt the recovery information in packet P_i by trying key k_{i-j} and previous keys $k_{i-j-1}, \ldots, k_{i-m}$ (that it can derive by successive hashing). No single choice of m is therefore equally effective across receivers that have missed different number of broadcast packets.

Instead of fixing m, the base-station can vary m in a randomised way from packet to packet. We propose that the base-station choose m according to a geometric distribution given by $(1-p)^{m-1}p$ for a chosen parameter $p \in (0,1)$ (discussed further below) – the base-station can implement this choice easily by simulating a (biased) coin toss. With such a choice of m by the base-station, a receiver that has missed $k > 1$ broadcast packets can successfully decrypt (step 6) the recovery information in the received packet if and only if $k = m$, which happens with probability $(1-p)^{m-1}p$. A formal analysis of our scheme to quantify the impact of the coin-toss bias probability p is presented next.

IV. Analysis

In this section we analyse our recovery scheme to study the impact of parameter p on the recovery probability and expected recovery time. Let S_k denote the probability that a chosen receiver that has lost l previous keys will recover within k subsequent packet receipts. To compute S_1, we note that the receiver recovers using the first received packet (subsequent to its l lost packets) only if the recovery information in that packet was encrypted using the last key held by the receiver – this key is l back in the chain, and the probability that the base-station chooses that key for encrypting the recovery information is $p(1-p)^{l-1}$. Therefore

$$S_1 = p(1-p)^{l-1}. \tag{1}$$

We can generalise this argument to obtain for general k:

$$S_k = S_{k-1} + (1 - S_{k-1})p(1-p)^{l+k-2}, \tag{2}$$

which states that a receiver recovers within k steps if either it has already recovered within $k-1$ steps, or if it has been unsuccessful in recovery in the first $k-1$ steps, and recovers successfully in the k-th step, the latter happening with probability $p(1-p)^{l+k-2}$. The recurrence in equation 2, together with the initial condition in equation 1, can be used to numerically compute S_k for arbitrary k. In particular, the probability S that the chosen receiver will **ultimately** recover is given by

$$S = \lim_{k \to \infty} S_k. \tag{3}$$

The ultimate recovery probability S is plotted in Fig. 4 as a function of the parameter p (on log scale) for various number of keys lost by a receiver. For the special case of exactly one lost key, $p = 1$ guarantees recovery since the transmitter will always choose one key back in the chain, which is available at the receiver, and lowering p reduces the chances of ultimate recovery. However, for the more general case where a receiver has missed more than one key, recovery probability monotonically increases with decreasing p. This can be explained by the fact that as p reduces, the transmitter is more likely to use older keys in the chain, which gives more opportunity to a receiver to wait for the right recovery

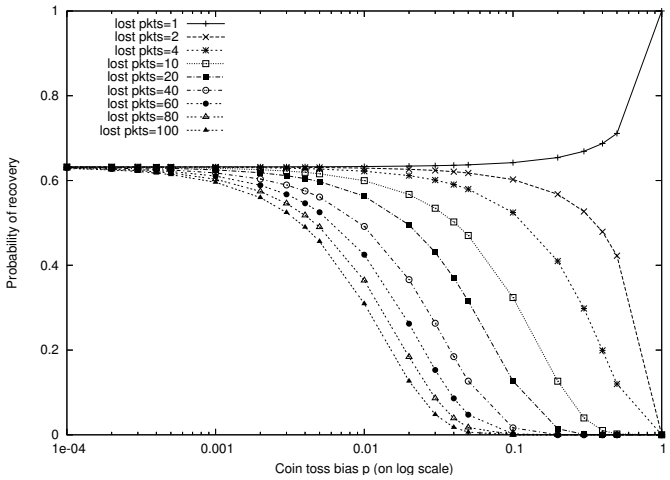

Fig. 4. Ultimate recovery probability versus p for various number of lost packets

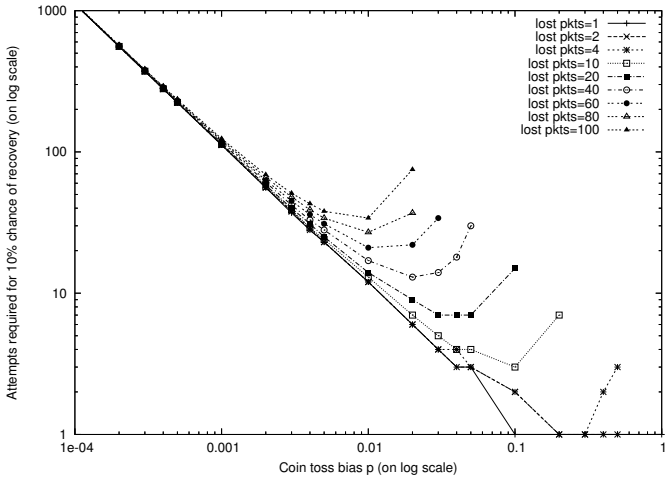

Fig. 5. Attempts required for 10% chance of successful recovery versus p for various number of lost packets

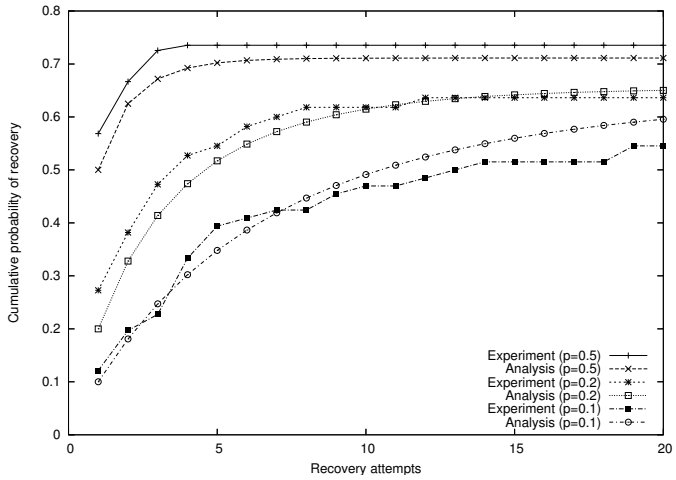

Fig. 6. Comparison of experiments and analysis showing cumulative recovery probability as a function of number of attempts for one lost packet and $p = 0.5, 0.2, 0.1$

key, irrespective of the number of keys it is missing.

Though a lower p increases chances of ultimate recovery, one would expect to pay a price in terms of the time taken for recovery (i.e. number of packets that a receiver needs before recovery succeeds). Fig. 5 shows the number of attempts (on log scale) needed to achieve a 10% chance of successful recovery (namely, the lowest k for which $S_k > 0.1$) as a function of p. Note that some curves terminate at the right since larger values of p do not yield a 10% chance of ultimate recovery. The plot clearly shows that for a given number of lost packets l, the number of attempts is minimised at approximately $p = 1/l$, and grows rapidly with p decreasing below this value. This confirms our intuition that a small p, though more likely to ultimate yield recovery, requires an increasing number of attempts to do so. An operator of this recovery scheme may therefore tune the p value in the system to balance these two aspects depending on application requirements and expected operating conditions.

V. Prototype Implementation and Results

We undertook an implementation of our scheme on a MicaZ mote [9] based platform available from Crossbow Technologies running TinyOS. We used the RC5 symmetric encryption module from TinySec [10], which uses 8-byte keys. The transmitter creates a key chain by choosing an initial 8-byte random number and hashing it using the SHA1 algorithm (with the lowest 8 bytes of the 20-byte result being used as the predecessor key in the chain). It then broadcasts successive packets with format as described earlier and shown in Fig. 2, where the data field carries just an 8-byte sequence number that increments with each packet. Each receiver implements the packet processing operations, including recovery from lost keys, described in Fig. 3. The program image on the receiver used 16652 bytes of ROM and 3762 bytes of RAM, and fits comfortably within the available memory on the MicaZ motes. We programatically forced one of the receivers to periodically drop a packet, and then timed the number of attempts required for recovery. We performed three sets of experiments, with settings of $p = 0.5$, 0.2, and 0.1 at the transmitter. For each setting the experiment was repeated nearly 100 times to measure the recovery time from one lost packet. Fig. 6 plots the observed cumulative recovery probability S_k as a function of k for each of the three values of p, and also compares them to the values obtained from eq. (2) of the analysis. The match is found to be quite good, and validates the analytical prediction of recovery performance against actual implementation.

VI. Conclusions and Future Work

In this paper we have addressed the robustness of key chain based encryption of broadcast messages in wireless sensor networks, and developed a scheme that allows recovery of keys missed by receivers due to packet loss. Our solution is easy to implement, requires small computational resources at the receiver nodes, scales well to systems with a large number of receivers, and keeps receiver location hidden. The ability to recover using an old key of the chain can be balanced against the number of packets required for recovery, allowing the operator to tune the system to balance robustness against vulnerability to a compromised key. This trade-off was quantified analytically, and verified experimentally on a MicaZ mote based platform. We believe our scheme can be useful in sensor networks used for battlefield and resource monitoring applications that require robust mechanisms for broadcast security. Our future work will undertake more experimental measurements of scenarios in which receivers miss multiple keys, and develop guidelines for tuning p under specific application settings.

References

[1] A. Perrig and J. D. Tygar, *Secure Broadcast Communication in Wired and Wireless Networks*. Kluwer Academic Publishers, 2002.

[2] M. Luk, A. Perrig, and B. Whillock, "Seven Cardinal Properties of Sensor Network Broadcast Authentication," in *ACM Workshop on Security of Ad Hoc and Sensor Networks (SASN)*, Alexandria, VA, Oct 2006, pp. 147–156.

[3] "Water Resources Observation Network," CSIRO news. [Online]. Available: http://www.csiro.au/news/newsletters/0604_water/ecos.pdf

[4] A. Perrig, R. Szewczyk, J. Tygar, V. Wen, and D. Culler, "SPINS: Security Protocols for Sensor Networks," *Wireless Networks*, vol. 8, no. 5, pp. 521–534, 2002.

[5] J. Shaheen, D. Ostry, V. Sivaraman, and S. Jha, "Confidential and Secure Broadcast in Wireless Sensor Networks," in *IEEE Intl. Symp. Personal, Indoor and Mobile Radio Comms. (PIMRC'07)*, Athens, Greece, Sep 2007.

[6] Y. Jiang, C. Lin, M. Shi, and X. Shen, *Security in Sensor Networks*. Auerbach Publications, 2007, ch. Key Management Schemes for Wireless Sensor Networks, pp. 113–143.

[7] P. Rogaway, M. Bellare, and J. Black, "OCB: A Block-Cipher Mode of Operation for Efficient Authenticated Encryption," *ACM Trans. Information and System Security*, vol. 6, no. 3, pp. 365–403, Aug 2003.

[8] H. Tan, S. Jha, D. Ostry, J. Zic, and V. Sivaraman, "Secure Multi-Hop Network Programming with One-Way Hash Chains," in *Proc. ACM Conf. Wireless Network Security (WiSec'08)*, Virginia, USA, Mar-Apr 2008.

[9] Crossbow-Technologies, "Mica2 and MicaZ motes." [Online]. Available: http://www.xbow.com

[10] C. Karlof, N. Sastry, and D. Wagner, "TinySec: A Link Layer Security Architecture for Wireless Sensor Networks," in *ACM SenSys*, Baltimore, MD, Nov 2004, pp. 162–175.