Secure Key Loss Recovery for Network Broadcast in Single Hop Wireless Sensor Networks[☆]

Syed Taha Ali[*,a,b], Vijay Sivaraman[a], Ashay Dhamdhere[a], Diethelm Ostry[b]

[a] *University of New South Wales, Sydney, Australia*
[b] *ICT Centre, CSIRO, Australia*

Abstract

Symmetric encryption of data at the base-station using time-varying keys has been proposed as an attractive method for securing broadcasts in wireless sensor networks: symmetric decryption keeps computational costs at sensor nodes low, while time-varying group keys protect the network against key compromise at any of the receivers. However, a significant problem is that interference or disconnections may cause a receiver to miss broadcast packets and the dynamic keys contained therein, rendering it unable to participate in subsequent broadcasts. In this paper, we develop a scheme which allows receivers to recover from key loss in a secure, efficient, and scalable manner. Our scheme appends recovery information to each broadcast message to help out-of-sync receivers re-attach probabilistically using an older key. We analyze our scheme to quantify the recovery probability as a function of system parameters, and deduce fundamental asymptotic bounds on recovery. We further prototype our scheme on the MicaZ mote platform and show that it is light-weight and efficient. Our solution offers a highly configurable, efficient and scalable method for key recovery in large sensor networks that require secure broadcasts.

[☆]This submission is an extended version of a paper presented at the 19th Annual IEEE Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC'08), Cannes, Sep. 2008.

[*]Corresponding author: Syed Taha Ali, School of Electrical Engineering and Telecommunications, UNSW, Sydney, NSW 2052, Australia. Email: taha@student.unsw.edu.au, Tel: +61 2 9385 4477, Fax: +61 2 9385 5993.

Email addresses: taha@student.unsw.edu.au (Syed Taha Ali), vijay@unsw.edu.au (Vijay Sivaraman), ashay@unsw.edu.au (Ashay Dhamdhere), diet.ostry@csiro.au (Diethelm Ostry)

Key words:
broadcast, privacy, authenticity, key chain, key loss recovery

1. Introduction

Wireless sensor networks rely on network broadcast for a variety of applications that include software updates, command dissemination, time synchronization and network management. Security of broadcast messages is a fundamental concern, especially in scenarios where the data is of a critical nature, e.g. military applications and resource monitoring.

It is easy to eavesdrop on all wireless traffic and inject false data into the network. Traditional security solutions designed for point-to-point networks cannot be directly applied: sensor networks have severely constrained resources and may consist of hundreds, even thousands of nodes. Simply encrypting the data using symmetric cryptography gives rise to issues in key distribution and authentication; were a node to be compromised, an intruder could spy on all traffic as well as impersonate as any other network entity. On the other hand, asymmetric cryptography, suited for authenticated exchanges in the form of digital signatures, is computationally intensive and has high communication overhead. What is required is a lightweight and scalable scheme which intelligently integrates symmetric and asymmetric mechanisms, and minimizes the tradeoffs involved.

One approach is the use of **time-varying keys**. Successive broadcast messages are encrypted with different keys from a 'key chain' before transmission, ensuring **confidentiality**. The shared key, possessed by all nodes, can be varied on a per message basis or after a select time interval.

A one-way function is used to generate this 'key chain' - a seed value is repeatedly hashed to yield an array of 'keys' which are then used successively in reverse order. This mechanism enables **authentication**: a receiver can confirm a new key by hashing to see if it matches up with an older one in its possession - this verifies that the key has indeed originated from the same source. The one-way nature of the hash function ensures that the receiver can validate the next key it receives, but not forge it.

In a typical scenario utilizing this approach it is assumed that trusted receivers share a 'root-key' on the basis of which communication is initiated (this key could be statically input into the receiver before deployment or communicated using secure key management schemes). At the transmitter

end, data to be sent is encrypted using the root-key and then dispatched to the receiver. In the event of a key change, the transmitter sends the new key to the receiver as part of the broadcast message. The receiver decrypts the message using the root-key and verifies the new key by hashing and comparing the checksum to the current key. Successive communication proceeds on this pattern: the receiver decrypts each message using the key it currently holds; from the message, it extracts and updates the key itself. A receiver need only perform symmetric-key decryption and hashing operations which are less resource-intensive than the asymmetric alternatives.

An eavesdropper would have a very restricted time-frame in which to crack the key before the key itself got updated. Were he to physically capture a receiver and extract the key, the attack would still be limited: he would be able to decipher the communication but would not be able to reverse-engineer forthcoming keys in the chain i.e. authentication would still be guaranteed.

There is an obvious shortcoming to this method: if a legitimate receiver in the field were to miss a key update due to interference or disconnection, it would be unable to decrypt future messages and be effectively shut out of all subsequent communication. We assume a scenario where a receiver does not transmit explicit message acknowledgements to the base-station. This could be for any number of reasons: transmission might put a strain on resources, the network might be too big for the base-station to service individual key-requests, the receiver might want to keep its location secret, etc.

The contributions of this paper are as follow:

- We describe a mechanism by which a receiver is able to recover lost keys, and re-establish communication with the transmitter in a secure manner
- We analyze in detail how this scheme can be configured for optimal performance depending on application scenario
- We implement our scheme on the Crossbow MicaZ platform and compare simulated results with real experimental performance

The rest of this paper is organized as follows: Section 2 lists the assumptions we make about our network and summarizes prior work in the area of network broadcast. We detail the workings of our scheme in Section 3. We analyze its performance in Section 4 and also discuss the limitations on

recovery from key loss. In Section 5 we briefly describe our prototype implementation together with some practical results weighed against the earlier analysis. We conclude in Section 6 and highlight once again the salient features of our key loss recovery mechanism.

2. Assumptions and Broadcast Security Proposals

In this section, we delineate the scope of our work, we discuss the environment within which we operate and examine the threat model. We also provide an overview of some existing security schemes which implement network broadcast and examine their strengths and weaknesses.

2.1. Operating Environment and Assumptions

We presume to work with a single-hop wireless sensor network consisting of a base-station that is able to broadcast data directly to all sensor nodes in the field. We restrict our work to single-hop networks at the moment for two reasons:

First, broadcast security is a challenging problem and traditional solutions cannot be directly applied. To contain complexity, this paper addresses the relatively simpler scenario of a single-hop network (as we shall observe, even for this restricted scenario, solutions are non-trivial). We defer investigation for multi-hop transmission for future work.

Second, there is value in developing security solutions for single-hop networks, particularly in hierarchically organized networks. For example, in a battlefield scenario, it would not be uncommon for a satellite or unmanned aerial vehicle (UAV) to directly (i.e. single-hop) broadcast command and control messages to all soldiers in a troop unit. Likewise in a natural resource monitoring application, a mobile base-station could periodically broadcast instructions to all sensor devices in the region in single-hop fashion. In such networks, multi-hopping might not even be desirable (for possible reliability and energy reasons).

We assume that the base-station has sufficient computation and energy resources and is secure against attack. We recommend that receiver nodes be fitted with some form of tamper-resistant hardware, such as Trusted Platform Module (TPM) [20], to ensure that cryptographic keys are securely protected. In the event that physical security of the node cannot be guaranteed, data will no longer be confidential. However, our scheme would still guarantee authenticity.

We note that sensor network nodes will generally possess limited resources and will be similar in computation and communication capability to the current generation of Berkeley MICA motes [2]. However, some nodes deployed in the field may perform a more critical function in the network and, therefore, may be superior to other nodes in terms of hardware, available resources and functionality. In the event of key loss, these nodes may have a higher recovery priority than other nodes.

We assume that packet loss is unavoidable and unpredictable. Studies have shown that radio connectivity is a complex phenomenon [4] even in ideal settings and may differ considerably from node to node. Using error recovery algorithms for reliable communication can become fairly complicated and would require extensive on-site testing. And in certain wireless sensor applications (resource monitoring, espionage, etc.) sensor node placement and site conditions may not be known beforehand or may be impractical to model: the nodes may simply be scattered by hand or aerial craft and self-organize into a functioning network.

The wireless channel is open and insecure and an adversary can eavesdrop without detection, masquerade as another entity, inject false data into the network and retransmit previously captured messages at a later time. In this paper, we do not consider more advanced attacks such as denial-of-service, wormhole attacks, etc. We assume the network can resist physical jamming of signals or attacks on the MAC layer by employing frequency hopping, spread spectrum techniques [14], etc. For the case where a receiver itself has to broadcast to the network, we assume it follows the strategy suggested in [13] where the receiver unicasts the message securely to the base-station which then broadcasts it to the entire network.

In this paper, we do not address the issue of key-loss recovery for multi-hop networks and leave it for future work.

2.2. Network Broadcast Schemes

There exist schemes in the literature which broadly address different aspects of network broadcast security using various mechanisms. We classify them accordingly:

2.2.1. Shared-key Schemes for Secure Broadcast

TinySec [6] implements symmetric-key encryption of data at the link layer of the communication protocol. The encryption-key in this case is a global

key shared by all receivers and is assumed to have a long lifetime. An essentially static communication key eliminates the need for a key loss recovery mechanism but at the cost of security: an attacker could potentially crack this key over time, and an untrusted or compromised receiver in the network could easily forge messages. TinySec also has no inherent protection against replay attacks.

MiniSec [11] borrows from TinySec and Zigbee [23] and improves on both in terms of stronger security and significantly greater power savings. MiniSec has different modes for unicast and broadcast communication. The latter, MiniSec-B, uses OCB[17] encryption to provide data confidentiality and authentication. The encryption key is again shared by all receivers and does not change. Sequencing of packets is partially achieved by using a counter as the initialization vector. MiniSec demarcates network lifetime into 'epochs' A received packet is first checked to see if it originated within the threshold of the previous or current epoch. The receivers use space-efficient Bloom filters to identify replayed packets. The drawbacks of this scheme include network-wide loose-time synchronization and, as in the case of TinySec, having a global shared key. The attacker model used by the authors of MiniSec precludes scenarios where a receiver in the field might be physically compromised.

2.2.2. Broadcast Authentication using Hash Constructions

Secure Deluge [3] secures the Deluge network programming protocol by enabling receivers to verify authenticity of received packets. Deluge [5], included by default in TinyOS, enables the network operator to disseminate code images and issue software updates using wireless broadcast in multi-hop networks. Without proper authentication, an attacker can easily reprogram other nodes and hijack the entire network. Secure Deluge uses a hash chain to enable source and data authentication. The code image is segmented and each packet is sent out with a hash digest for the next packet to follow. On receiving the next packet, the node computes the hash over the data field and compares it with the hash value received earlier to verify authenticity and packet sequence. A digital signature authenticates the very first hash value. The issue of missing hash digests due to packet loss is mitigated because Deluge itself has in-built mechanisms to ensure reliable and in-sequence distribution of data. However, this scheme does not address data confidentiality.

The Sluice [8] protocol operates in the same fashion except that hash values are not computed on a per packet basis but over "pages" (one page is

typically 1 kB of program image).

Chang et.al propose an efficient broadcast authentication scheme [1] using one-time signatures based on hash trees [12]. The transmitter segments the data to be sent to construct a hash tree, the root of which is advertised across the network as the transmitter's 'public' key. Data items are periodically broadcast allowing receivers to reconstruct the tree and confirm authenticity of the data. Packet loss would halt the verification process because the node would need to receive all the data to fully reconstruct the tree. To make the process resilient to loss, the authors propose that data items be transmitted along with their 'authentication path' to allow the receiver to traverse up the tree without having all the data in possession. The authors include mechanisms to optimize this process by effectively manipulating parameters such as number of hash trees, depth of the tree, and include a suggestion for re-keying. One restriction of this scheme is that all the data to be sent must be known beforehand to construct the hash tree. This scheme, also, does not address data confidentiality.

2.2.3. Broadcast Authentication using Time-Varying Keys

The μTesla protocol [13], uses time-varying keys to guarantee data authentication. A key chain is generated using a hash function. The root-key is distributed securely to all receivers by unicast. The network uses loose time-synchronization to maintain regular time-intervals and keys are updated per interval. Data integrity is ensured using message authentication codes (MACs), generated using the secret key, appended to the data in the packets. The base-station only reveals the secret key for a time interval after the interval has lapsed. Whatever messages the node has received during this interval are buffered in memory until key disclosure. A node authenticates a newly received key by hashing to see if it yields the previous key in its possession. An attacker is unable to forge messages using revealed keys because their lifetime has already expired. This scheme is not severely impacted if a key is dropped: the receiver can use later keys to generate lost keys and verify stored data. The drawbacks of this scheme include loose network-wide time synchronization and buffer space to store received messages till they can be authenticated. An attacker could flood the network with forged packets forcing receiver buffers to overflow.

Multi-level μTesla [10] extends the capabilities of μTesla by simplifying the key-distribution phase and introducing the concept of a multi-level key chain generated using pseudo-random functions to greatly improve protocol

efficiency. Multi-level μTesla reduces the need to re-initialize the network by implementing multiple levels of key-chains, in which high-level keys are used to communicate root-keys (or *commitments*) for low-level chains which are used in turn for broadcast authentication as in standard μTesla. The chains are further linked in that each root-key is derived from the corresponding high-level chain using another pseudo-random function. Network lifetime is extended many times over. This also significantly reduces authentication operations for the receiver which would not have to cycle through a single long chain in the event of multiple packet loss but could traverse entire chains in single steps using the high-level keys. A problem would arise if a receiver were to drop a *commitment distribution message* (CDM) initializing a new low-level chain; it would be unable to verify any broadcast data received during the entire lifetime of the chain itself. The data would still be verifiable eventually as the receiver could use any later commitment distribution message to reconstruct all the lost high-level keys and the corresponding chains. This would require significant computation and storage, and the the authors propose randomly timed retransmissions of commitment distribution messages during each time interval to improve robustness to high-level key loss.

2.2.4. Secure Broadcast using Time-varying Keys

The Localized Encryption and Authentication (LEAP) protocol suite [22] aims to support in-network processing of sensor data and restricting the impact of a security breach to the immediate neighborhood of the compromised node. For this purpose, LEAP implements four different keys on each node, individual keys between sensor node and base-station, pairwise keys between sensor nodes, a cluster key shared with multiple neighboring nodes and a group key which is shared by all nodes. This group-key is used by the base-station for global broadcast purposes and ensures data confidentiality. The authors suggest frequent rekeying of the group-key and assume use of a routing protocol (similar to the TinyOS beaconing protocol [7]) to securely propagate the new key to individual nodes. To authenticate the data, LEAP assumes the presence of a broadcast authentication protocol, such as μTesla. This brings in the need for network-wide time synchronization and delayed verification of broadcast data.

Earlier, we proposed a scheme [18] for confidential and secure broadcast in single-hop wireless sensor networks that used time-varying keys. We detail the operation of this scheme as a precursor to enhancing it for key loss recovery:

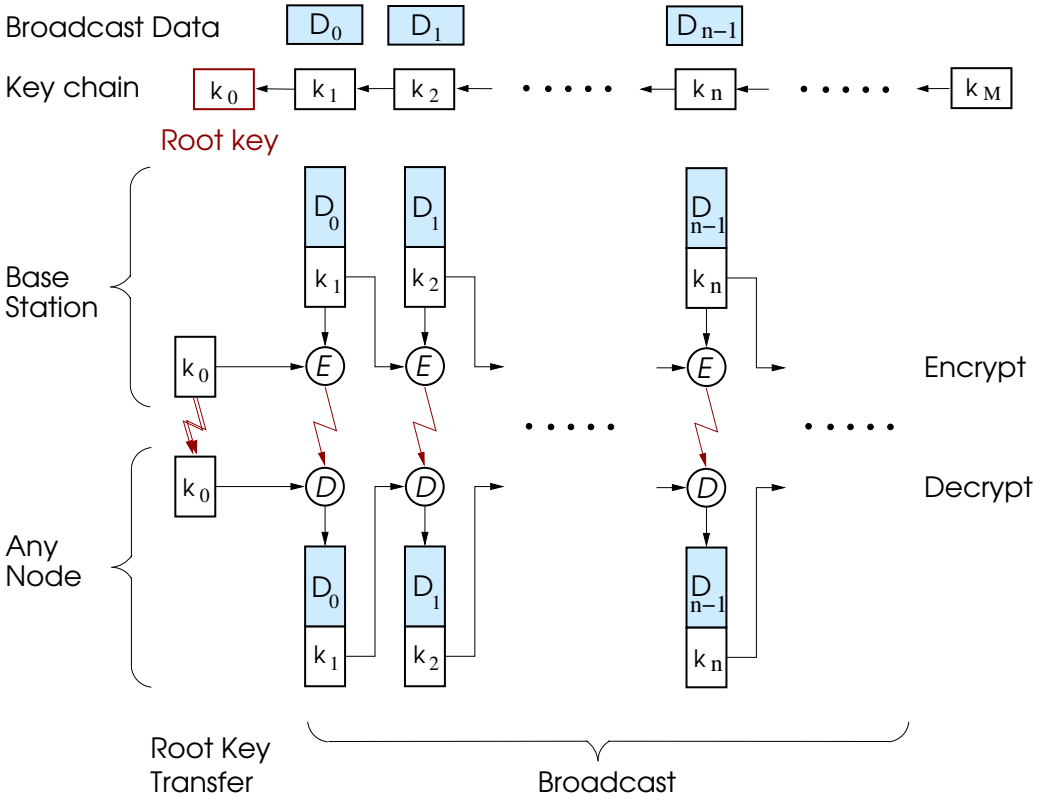


Figure 1: Key chain encryption of broadcast packets

The base-station generates a key-chain by successively hashing (SHA1 or MD5) a seed value, k_M, to yield a key chain $k_M, k_{M-1}, ...k_1, k_0$. Where the broadcast data is known beforehand, the key-chain length can be determined in advance, equivalent to the number of packets to be broadcast. In dynamic situations, the key chain can be of arbitrarily long length.

In the bootstrapping phase, we assume that, k_0, the last key in the chain is securely implanted in all receivers. This could either be programmed into the nodes before deployment, else transmitted via an authenticated Diffie-Hellman exchange. This only has to be performed once on network initialization.

Once this step is successfully accomplished, the base-station proceeds to create the broadcast packet. The structure of this packet consists of the data to be sent, together with the successor key, k_1. The packet is then encrypted using key k_0 and a suitable symmetric key cipher (RC5 [16], Skipjack [21],

etc.). The encryption scheme must ensure that the data and key are not separable in ciphertext, so that any modification of the encrypted data also destroys the key. Such message integrity can be assured by using the OCB mode [17] of block cipher encryption.

A receiver in the field decrypts this packet using k_0 already in its possession to extract the data and the next key, k_1. It verifies authenticity by checking if key k_1 hashes to yield k_0. The node can now process the data and discard key k_0 and update with k_1. Using key k_1, the node can now decipher the next message that the base-station will transmit.

All other communication follows a similar pattern to the last two steps: every message the base-station sends contains the key for the next message to follow. The node decrypts each successive message using the key it already has, authenticates the source and updates its key. This process is summarized in Fig1.

The novelty of this scheme is that each packet is encrypted with a different key and that the key itself contains the means of authentication. Nodes can individually verify each key they receive but cannot forge the next key in the chain on the basis of previously disclosed keys. An intruder has a very limited window to mount a successful attack because the broadcast key expires with every message transmitted. And even if the attacker were to obtain a valid key and decipher all communication, authenticity would still be guaranteed: the attacker would be unable to forge new keys and masquerade as the base-station. A further layer of security could be adding by employing a suitable block cipher mode (e.g. OCB [17]) to bind the key and the data in the ciphertext at the base-station.

3. Key Loss Recovery

To illustrate our mechanism for key loss recovery, we extend our own secure broadcast scheme described earlier in Section 2.2.4 and elaborated upon in [18]. The principles are basic and can be applied to different schemes with a little modification.

We extend the original scheme by appending a **Recovery Field** to the original broadcast packet structure; this field contains information that would help a receiver reattach to the network in the event of key loss. The Data Field in our original scheme consists of a data segment, D_i, and the next key in the chain, k_{i+1}; this field is encrypted using the current key in the chain, k_i. In the Recovery Field, we put the next key, k_{i+1}, an integer value, m, and

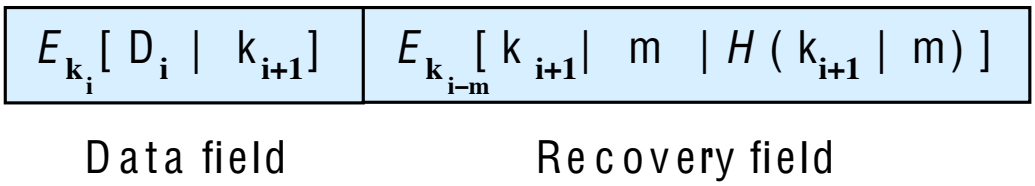


Figure 2: Packet structure enhanced to include recovery information

a hash digest of the concatenation of the two, $H(k_{i+1}|m)$, to allow receiver to verify successful decryption; we encrypt this field with an older key from the chain, k_{i-m}. Fig.2 shows the modified packet structure. A receiver that has lost m packets in succession can decrypt the Recovery Field using the last key it holds, k_{i-m}, and recover the next key, k_{i+1}, which will enable it to recover from the key loss and decipher future broadcast messages.

The basic algorithm for packet processing is spelt out in Fig.3: when a node receives a packet, in step 1, it decrypts the Data Field to obtain the data segment, D_i and the next key, k_{i+1}. In step 2, it performs a key verification check. If the test is successful, the node processes the data accordingly. If the test fails, it may possibly be due to packet loss and the node goes into **recovery mode**, steps 5 to 14.

Only in this instance is the Recovery Field examined. The node uses the last received key in its possession to decrypt this field (step 6). The decryption may or may not be successful: the node has no way of knowing how many keys it has missed nor which value of m the receiver might have chosen. To check if the data decrypted successfully, step 7 hashes the fields, $k_{i+1}|m$ and compares the result with the hash digest already contained in the Recovery Field. If the two digests do not match, it means recovery was unsuccessful and the packet is discarded (step 13). In the event the digests match, the node extracts the key, k_{i+1}, in step 8, and hashes it m+1 times to authenticate it against the current key in its possession. If successful, the new key is verified as having originated from the base-station; the node subsequently updates its key (step 11). It is now in a position to be able to decrypt future broadcast messages.

The immediate advantage of this scheme is that the receiver does not have to transmit explicit requests to reattach to the network, thereby keeping its location secret and conserving resources. Receivers that do not need recovery will ignore the Recovery Field and do not have to pay any cost

```
// 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
```

Figure 3: Operations performed by node upon arrival of broadcast packet P_i

(other than merely receiving the Recovery Field); and for nodes that have lost keys, the tradeoff is in the time the receiver will take to recover and the local computational effort involved. Here, too, in an attempt to recover, unnecessary packets are dropped after a decryption (step 6) and hash (step 7) operation. And the computation effort spent in authenticating a recovery key will, at most, be linear to the number of lost packets.

The choice of m is a fundamental concern, given that packet loss is unpredictable and that the base-station has no way of knowing how many packets each of the receivers has missed. If a small value of m is chosen, a node that has lost a potentially larger number of packets, j, such that $j > m$, will be unable to reattach to the network because the key it holds, k_{i-j} will precede k_{i-m} in the key-chain for every value of i. Likewise, if m is a large value such that $m \gg j$, the node will have to miss out on a large number of messages before the keys align and it can reattach. Not only would this impair the node's performance in a dynamic network, but the node would also waste resources in the large number of futile attempts at recovery.

In view of this fact, rather than keep m as a constant value, the base-station can choose to vary the value in randomized fashion from packet to packet. This strategy would be effective in a large dynamic network where different nodes have missed different numbers of broadcast packets. We propose the base-station select m using a geometric distribution of the form $(1-p)^{m-1}p$ for a fixed parameter $p \in (0, 1]$. We will show analytically in Section 4 that there are fundamental asymptotic bounds on receiver recovery and it is not possible to get better results with a different approach.

This scheme can be implemented easily by simulating a coin toss, the bias of which corresponds to parameter p. Parameter p can be visualized as a measure of *newness*: higher values of p would lead to more current keys being used to encrypt the Recovery Field, and lower values of p would correspond to older keys being used.

Having p as variable gives the scheme a high degree of configurability. The network operator can control this value as per application requirements or even environmental conditions: setting p at a high value will enable quick recovery for critical applications but will likely not be suitable for very lossy conditions; in this case setting p to a lower value will improve the chances of the receiver to recover in the longer run.

We propose a further addition to this scheme: as we noted earlier, the network nodes may not be identical in terms of computational resources or application function. The operator may prefer certain nodes to recover faster than others. In this event, a receiver could store up more than one previous key with which it could attempt recovery. A node that stores, say, M former keys, would have a greater chance of recovery. The tradeoff in this case would be that the receiver would have to perform at most M times as many computations in its attempts to recover as if there were only one key. Some nodes may have sufficient processor power and battery life to make this a feasible option. The receiver would maintain a buffer of M most current received keys, stored in order; if recovery fails with the most recently received key, the receiver tries decrypting the Recovery Field with the key it received before that, and so on and so forth.

We highlight certain attractive features of this key recovery mechanism: it complements the existing secure broadcast scheme and preserves its essential strength. Encrypting the Recovery Field assures **confidentiality**. Supplying the receiver with the value of m allows it to hash back that many times and verify the source of the new key, guaranteeing **authenticity**. This scheme is **scalable** to a very large number of receivers because the base-station does

not have to service individual key recovery requests. There is no bidirectional communication and all computation and verification is done at the receivers' end. **Configurability** is another desirable feature. In mission-critical scenarios, it may be imperative that a node recover from key loss as soon as possible (e.g. battlefield communication). In passive monitoring scenarios, a longer delay may be permissible (e.g. monitoring non-critical resources, preventive maintenance, etc.) Environmental conditions might also vary greatly, certain deployments may be more prone to packet loss than others. The network operator can get desired results by fine-tuning the values of operational parameters.

4. Analysis

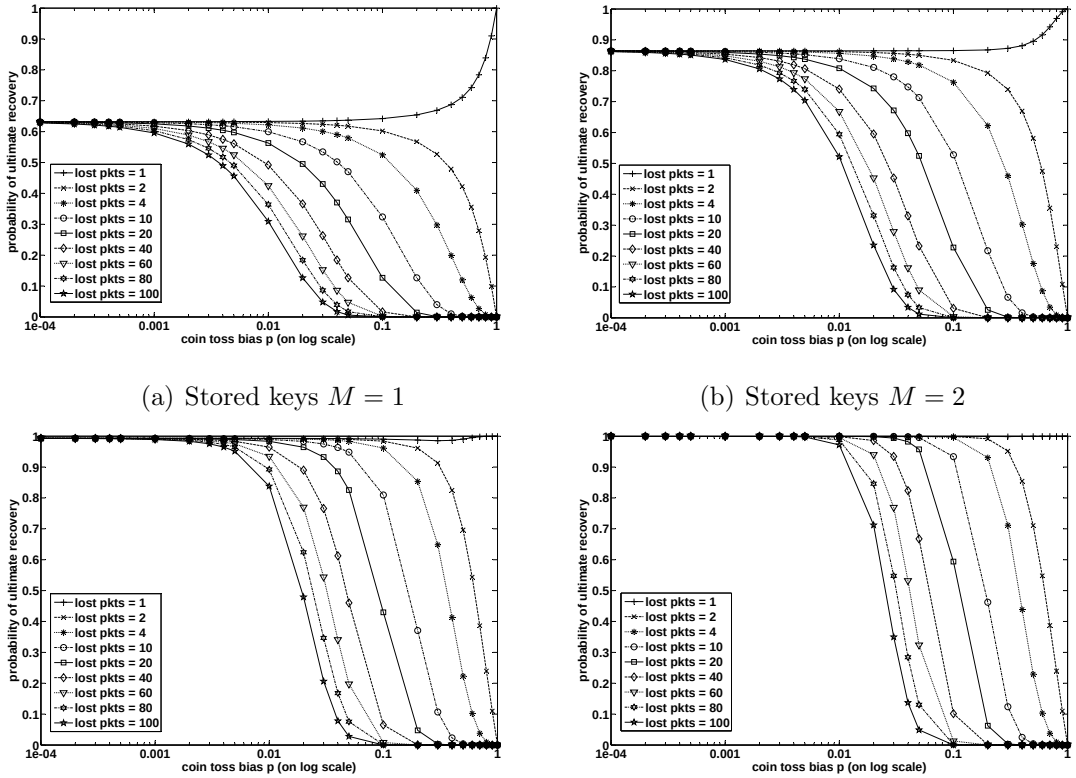
4.1. Probability of Recovery

In this section we analyze the impact coin toss bias, p, and number of older stored keys,M, have on the probability that a receiver will recover from key loss. We define l to be the number of lost keys and, k, the number of subsequent packet receipts (or recovery attempts) it takes the receiver to successfully reattach. We denote $P_{l+k}^{(M)}$ therefore as the probability that a receiver that has lost l keys recovers on the k-th attempt on the basis of M stored keys.

For the case where the receiver recovers with just one stored key (M=1), using the very first packet (k=1) it receives after l lost packets, we note that this occurs only if the Recovery Field for that particular packet was encrypted using a key l keys back in the key chain. This corresponds to the last key saved by the receiver before the packet loss. The probability that the base-station chooses this particular key out of l possible keys is $p(1-p)^{l-1}$. For the sake of simplicity, we express $(1-p)$ as q, i.e. the probability that the base-station does *not* choose the right key when encrypting the Recovery Field. Therefore, we denote $P_{l+1}^{(1)}$, the probability that the node recovers on the very first attempt,

$$P_{l+1}^{(1)} = pq^{l-1} \tag{1}$$

This can be visualized as a sequence of Bernoulli trials and $P_{l+1}^{(1)}$ represents the probability of achieving the first success after $l-1$ failures. Generalizing this for the kth case, we express the cumulative probability that the node will have recovered by the *next* step (i.e. the k+1 attempt), to be



(a) Stored keys $M = 1$

(b) Stored keys $M = 2$

(c) Stored keys $M = 5$

(d) Stored keys $M = 10$

Figure 4: Ultimate recovery probability versus coin toss bias p for various number of lost packets l and stored keys M

$$P^{(1)}_{l+k+1} = P^{(1)}_{l+k} + (1 - P^{(1)}_{l+k})pq^{l+k-1} \tag{2}$$

This indicates that a receiver recovers within $k+1$ steps if it has either recovered already within k steps, (the probability of which being $P^{(1)}_{l+k}$), or, if not, that it then recovers in the next step itself (the probability of which is pq^{l+k-1}). This is a recursive expression and the initial condition in Eq.1 allows us to compute $P^{(1)}_{l+k}$ for arbitrary k. In particular, we can deduce the probability $P^{(1)}$ that the receiver will ultimately recover which is

$$P^{(1)} = \lim_{k\to\infty} P^{(1)}_{l+k} \tag{3}$$

Fig.4(a) expresses the probability of ultimate recovery, $P^{(1)}$, plotted against

parameter p (on log scale) for various number of lost packets. We note that recovery is only guaranteed for the special case of one lost key, where coin toss bias p=1. This is because the transmitter will *always* use a key one step back in the chain to encrypt the Recovery Field and, since the receiver has lost only one packet, this particular key is the last key that the receiver has in its possession. Reducing p however, has the effect of reducing the probability of ultimate recovery, whereas for every other number of dropped packets, it actually increases. This can be explained: as the value of p is reduced, the base-station chooses older keys in the chain to encrypt the Recovery Field. It becomes easier for the receiver that has lost packets to wait for keys to eventually align and recover.

For the case where recovery is attempted on the basis of multiple stored keys ($M > 1$), a receiver will first try to recover using the more current key, the probability of which is pq^{l-1}; in case recovery does not succeed, the receiver will then use the next key, the probability of success being pq^l, the next with probability pq^{l+1} and so on. The overall probability of recovering on the very first received packet (k =1) is the sum of the individual probabilities for each key,

$$P_{l+1}^{(M)} = pq^{l-1} + pq^l + pq^{l+1} + ... + pq^{l+M-2} \tag{4}$$

The expression can be simplified,

$$P_{l+1}^{(M)} = pq^{l-1}[1 + q + q^2 + ... + q^{M-1}] \tag{5}$$

The geometric series can be summed up to yield

$$P_{l+1}^{(M)} = q^{l-1}(1 - q^M) \tag{6}$$

Generalizing this expression, as we did for Eq.(2) , we deduce the probability that the receiver will recover by the *next* attempt (i.e. k+1) to be

$$P_{l+k+1}^{(M)} = P_{l+k}^{(M)} + (1 - P_{l+k}^{(M)})q^{l+k-1}(1 - q^M) \tag{7}$$

Solving recursively with initial condition in Eq.6, we plot recovery probability for multiple stored keys (M=1, 2, 5, 10) against parameter p for various number of lost packets in Fig.4. Chances of ultimate recovery can clearly be seen to improve with multiple stored keys. In fact, we observe in Fig.4(c), for the case where a receiver with 5 stored keys, attempts recovery after one lost packet, there is a slightly perceptible *dip* in probability of ultimate recovery

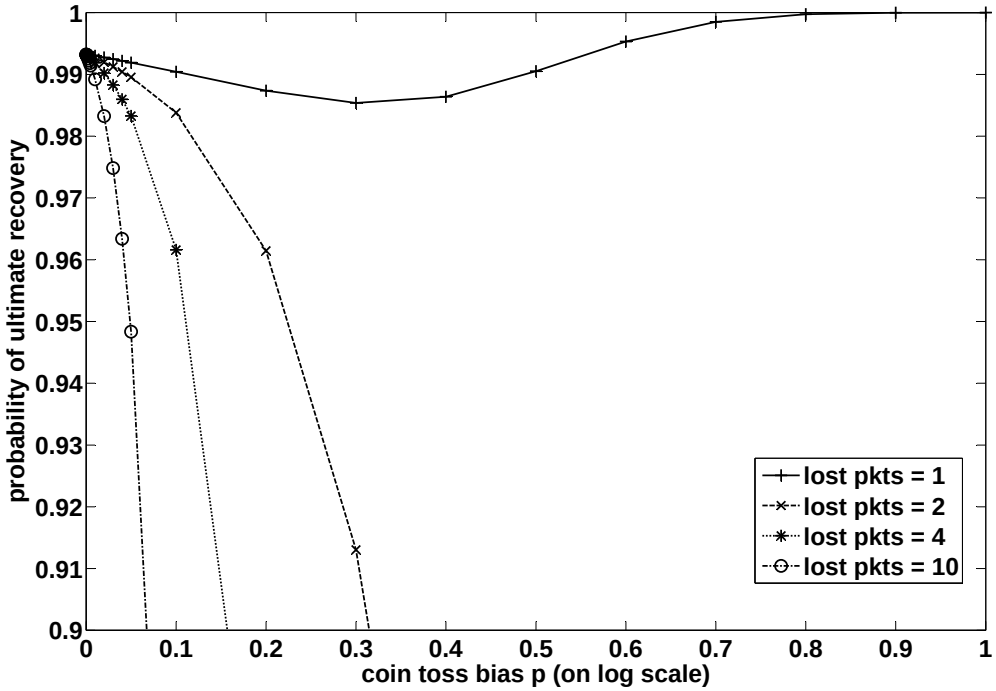


Figure 5: Highlighting trend in recovery probability versus coin toss bias p for various number of lost packets and $M = 5$ stored keys

for lower values of p. We highlight this result in Fig.5 which essentially reinterprets Fig.4(c) on linear scale and restricted axes and lesser variations of lost keys l. This trend occurs for all cases of $M > 2$ stored keys and should actually be interpreted as a *rise* in the probability of ultimate recovery as the base-station selects older keys in the chain to encrypt the Recovery Field and as the number of recovery attempts on the receiver's part tends to the infinite. This tendency is evident in the case of more than one lost key ($l > 1$) where there is a steady increase as parameter p decreases. A receiver in this scenario that has stored multiple keys has a better chance of recovery if the base-station uses older keys to encrypt the Recovery Field rather than newer ones.

4.2. Bounds On Recovery

In Fig.4, probability for ultimate recovery appears to converge for the more general case of more than one lost packet (at $P \approx 0.632$ for $M = 1$

stored key, $P \approx 0.865$ for $M = 2$ stored keys, etc.) There seems to be an upper limit on a receiver's probability for recovery. We justify this result with some general proofs from the theory of infinite products:

For a receiver that has one stored key ($M = 1$) and has lost one packet ($l = 1$), we note that the probability that it does *not* recover on the first try, denoted as $Q^{(1)}_{1+1}$, is $1 - p$ i.e. q; for non-recovery on the second attempt, $Q^{(1)}_{1+2}$, the probability is $q(1 - pq)$, for the third, $q(1 - pq)(1 - pq^2)$. We can express the probability that the node never recovers as an infinite product,

$$Q^{(1)}_1 = \prod_{k=1}^{\infty}(1 - pq^{k-1}). \tag{8}$$

Each term of this product is less than 1, so the probability that the receiver never recovers actually falls with increasing recovery attempts. We let $S^{(1)}_1$ be the sum of the corresponding infinite series

$$S^{(1)}_1 = \sum_{k=1}^{\infty} pq^{k-1} \tag{9}$$

Using observations from the theory of infinite products[15], we define parameter ρ as the largest term in the series, i.e. $\rho = max_{1 \le k \le \infty} pq^{k-1}$, and specify a, such that

$$a = (1 - \rho)^{-1/\rho} \tag{10}$$

and observe that the probability the receiver never recovers after losing one key, is tightly sandwiched between two decaying exponential functions,

$$a^{-S^{(1)}_1} \le Q^{(1)} \le e^{-S^{(1)}_1}. \tag{11}$$

Conversely the probability that the node does recover from a single key loss can be bounded:

$$1 - e^{-S^{(1)}_1} \le P^{(1)}_1 \le 1 - a^{S^{(1)}_1} \tag{12}$$

If we evaluate parameter ρ for the case of our geometric distribution, the receiver has the highest probability of recovery on the very first attempt, i.e. $\rho = p$. And $S^{(1)}_1$ represents a summation of the masses of a discrete probability density function:

$$\begin{aligned} S_1^{(1)} &= \sum_{k=1}^{\infty} pq^{k-1} \\ &= \sum_{k=1}^{\infty} p(1-p)^{k-1} \\ &= p(1/p) = 1 \end{aligned}$$

Our bounds on recovery become

$$1 - 1/e \approx 0.632 \leq P_1^{(1)} \leq 1 - (1-p)^{1/p} \tag{13}$$

In Fig.6(a), we overlay these derived limits on our existing results for probability of ultimate recovery for the particular case for a receiver that has $M = 1$ stored key and $l = 1$ lost packets (Fig.4(a)) and observe that the bounds are valid. Bounds for other cases can be similarly derived by formulating expressions for non-recovery, Q, and computing values of a, ρ and S and using the general expression

$$1 - e^{-S} \leq P \leq 1 - a^{-S} \tag{14}$$

For the case where a receiver has lost more than one key ($l > 1$), the probability of non-recovery can be expressed as an infinite product

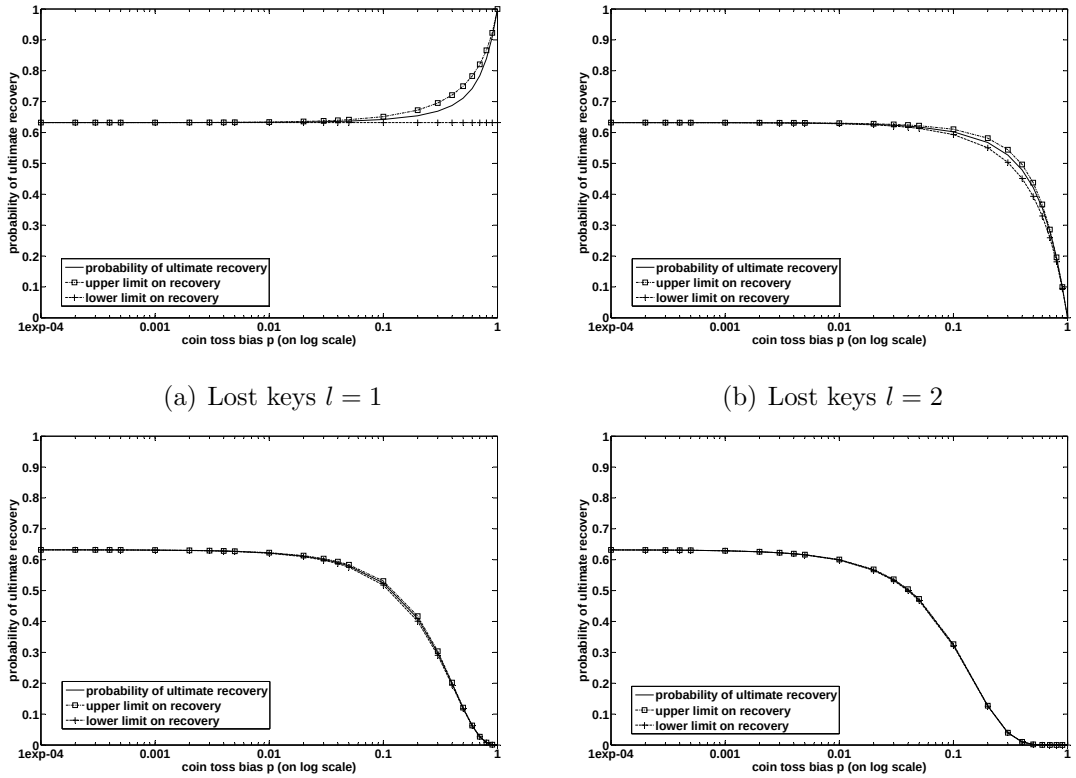
$$Q_l^{(1)} = \prod_{k=l}^{\infty} (1 - pq^{k-1}). \tag{15}$$

The corresponding infinite series is

$$S_l^{(l)} = \sum_{k=l}^{\infty} pq^{k-1} \tag{16}$$

and can be computed

$$\begin{aligned} S_l^{(l)} &= pq^{l-1} + pq^l + pq^{l+1} + ... \\ &= pq^{l-1}[1 + q + q^2 + ...] \\ &= q^{l-1} \end{aligned}$$



(a) Lost keys $l = 1$

(b) Lost keys $l = 2$

(c) Lost keys $l = 4$

(d) Lost keys $l = 10$

Figure 6: Limits on probability of ultimate recovery for various number of lost keys for the case of stored key $M = 1$

ρ can be expressed as pq^{l-1}, the maximal probability of recovery and a can be deduced as per Eq.10.

The bounds on probability of recovery for the case of l lost keys are therefore

$$1 - e^{-(q^{l-1})} \leq P_l^{(1)} \leq 1 - a^{-(q^{l-1})} \tag{17}$$

We plot these limits across the range of p in Fig.6 over the results for recovery probability presented earlier.

For the case where the receiver has $M > 1$ stored keys: in the event of losing l packets, the probability that a node with multiple keys does *not* recover on the very first packet it receives, $Q_{l+1}^{(M)}$, is $1 - P_{l+k}^{(M)}$, where $P_{l+k}^{(M)}$ is the sum of the individual probabilities of recovering with M keys in that

attempt. For non-recovery on the second packet, $Q_{l+2}^{(M)}$ is $(1 - P_{l+2}^{(M)})$. The probability that a node with multiple keys *never* recovers after losing one key is

$$Q_l^{(M)} = \prod_{k=1}^{\infty}(1 - P_{l+k}^{(M)}). \tag{18}$$

The sum of the corresponding infinite series is

$$S_l^{(M)} = \sum_{k=1}^{\infty}(P_{1+k}^{(M)}) \tag{19}$$

Each of these terms can be individually computed: ρ, the probability of recovery on the very first step was already presented in Eq.6 which allows us to compute a. $S_l^{(M)}$ can be expressed as

$$\begin{aligned} S_l^{(M)} &= \sum_{k=1}^{\infty}(P_{1+k}^{(M)}) \\ &= M\sum_{j=l}^{\infty} pq^{j-1} - \sum_{j=1}^{M-1}(M-j)pq^{l+j-2} \end{aligned} \tag{20}$$

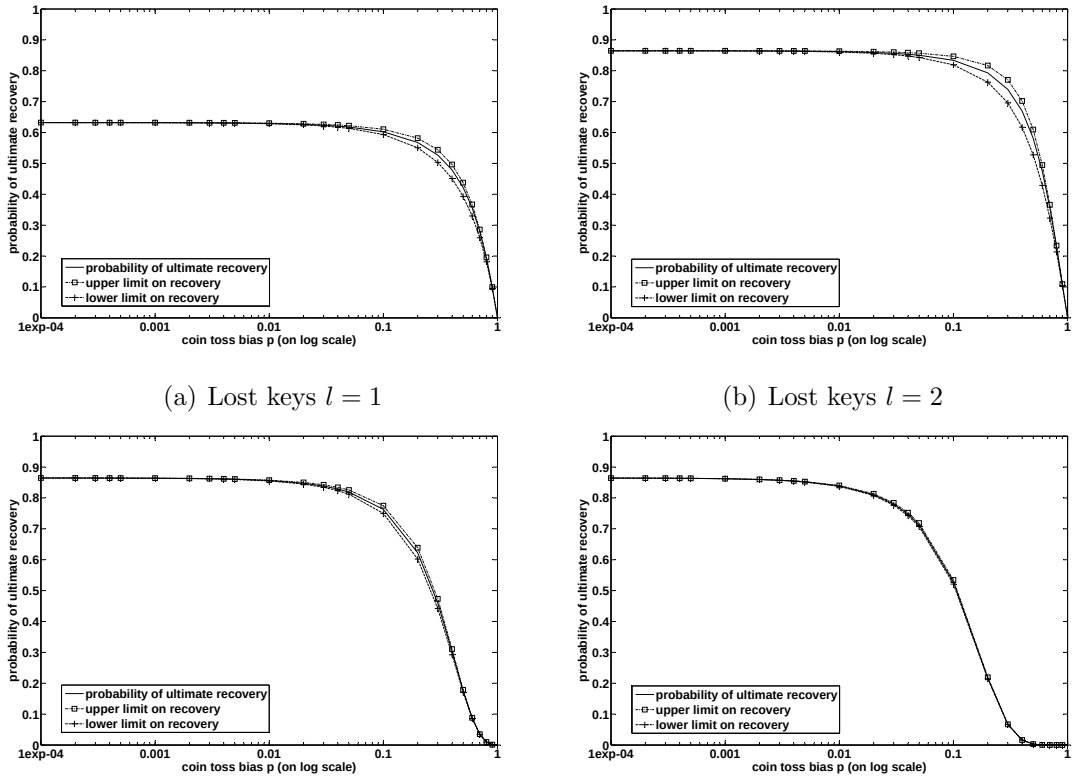
The limits on recovery are

$$1 - e^{-S_l^{(M)}} \leq P_l^{(M)} \leq 1 - a^{-S_l^{(M)}} \tag{21}$$

We plot the results for the case of $M = 2$ stored keys in Fig.7 to confirm that the probability of recovery is indeed bounded as predicted.

These results indicate that using multiple stored keys can effect a massive improvement in the receiver's ability to recover from key loss: attempting recovery with just two stored keys, $P^{(2)}$, gives the receiver an over 85% chance of ultimate recovery; for three keys, the probability is 95%; with ten stored keys, the receiver has a 99% chance of recovery. We validate these results by comparing them with experimental findings in the Prototype Implementation section.

The tradeoff is in the cost of the extra computation the receiver has to perform locally: it the receiver stores two keys, it has to perform, at most,



(a) Lost keys $l = 1$

(b) Lost keys $l = 2$

(c) Lost keys $l = 4$

(d) Lost keys $l = 10$

Figure 7: Limits on probability of ultimate recovery for various number of lost keys for the case of stored keys $M = 2$

twice as much work to recover, but recovery is likely faster and more certain. We believe that the option of using multiple keys, in conjunction with the ability to tune the value of parameter p, makes the network significantly more robust to packet loss and gives the network operator a high degree of configurability for specific applications and operating environments.

4.3. Recovery Time

A receiver's recovery time can be quantified in terms of the number of packets a receiver will need to receive before it recovers. Fig.8 gives an indication of this relationship in which coin toss bias p is plotted against the number of attempts required by the receiver for a 10% chance of recovery. We try to find the lowest value of k for which $S_k > 0.1$ as a function of p.

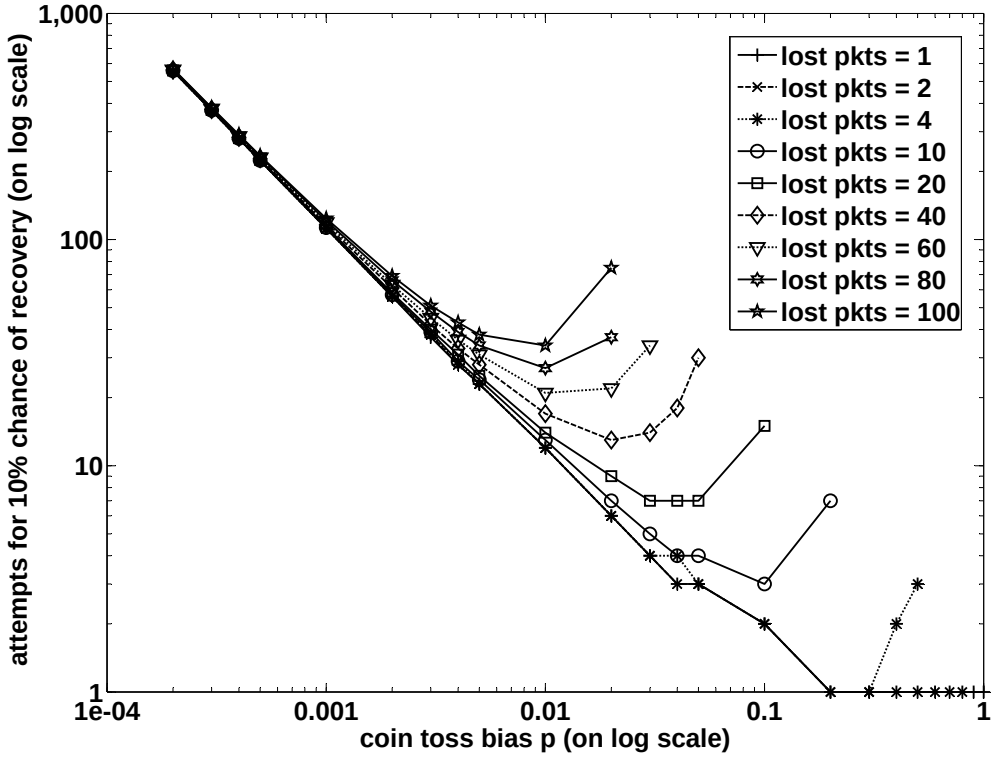


Figure 8: Attempts required for 10% chance of successful recovery versus p for various number of lost packets

From the figure it is seen that as coin toss bias p decreases, more recovery attempts (k) are required for a 10% chance of recovery.

Some curves terminate on the right, indicating that larger values of p do not yield a 10% chance of ultimate recovery (the recovery probability is actually higher as parameter p increases). It is clear, however, that increasing the probability of ultimate recovery by reducing parameter p has a tradeoff for the receiver in terms of increasing the number of attempts it takes to recover. The network operator can adjust the value of p to balance the two concerns depending on operating environment and specific application scenario.

5. Prototype Implementation

We prototyped our key loss recovery mechanism using a standard Pentium-based computer acting as the base-station and MicaZ motes [2], running TinyOS, acting as receiver nodes. We used Java for programming the base-station and freely available crypto APIs, namely the RC5 and SHA1 modules, from Legion of the Bouncy Castle [19]. On the receiver end, we implemented corresponding RC5 and SHA1 modules borrowed from the TinySec[6] and TinyECC[9] packages respectively. The program image of the receiver typically uses 17500 bytes of ROM and 1295 bytes of RAM; slight variations are due to the receivers storing different values of stored keys, M.

The implementation itself can be summarized:

1. At the base-station, an initial 8-byte seed key is continuously hashed to yield the "key chain". The SHA1 hash function originally yields a 20-byte message digest, but we truncate this to 8-bytes so that the key length matches our RC5 configuration (RC5-32/12/8); this also keeps the overall packet size from exceeding the MicaZ CC2420 radio's maximum allowable packet length. This key size can be increased to fit the security requirements of the deployment. This chain of 8-byte keys is then successively used in reverse order to encrypt broadcast messages.
2. Bootstrapping is done by securely programming the key into the nodes before deployment. If this proves problematic or inconvenient, a Diffie-Hellman key exchange can be used instead.
3. 3. The broadcast message is prepared based on the format described earlier and depicted in Fig.2. The 'data field' is 16-bytes long, the first 8-bytes consisting of 'data' as such, and the next 8-bytes consisting of the 'next' key that will be used to encrypt the following message. This whole is then encrypted using the RC5 symmetric cipher and the 'current' key in the key chain.
4. A biased coin is simulated using the Math.random() method from Java's Math class which generates pseudo-random numbers in a uniform distribution in the range of [0.0,1.0). These values are then repeatedly compared to a select biased value (p) to simulate 'heads' or 'tails'. The number of iterations it takes for a 'heads' to appear is taken as the value of the integer m.
5. The 'recovery field' is also 16-bytes long. The first 8-bytes consist of the 'next' key, the same as in the data field. The next 2-bytes contain

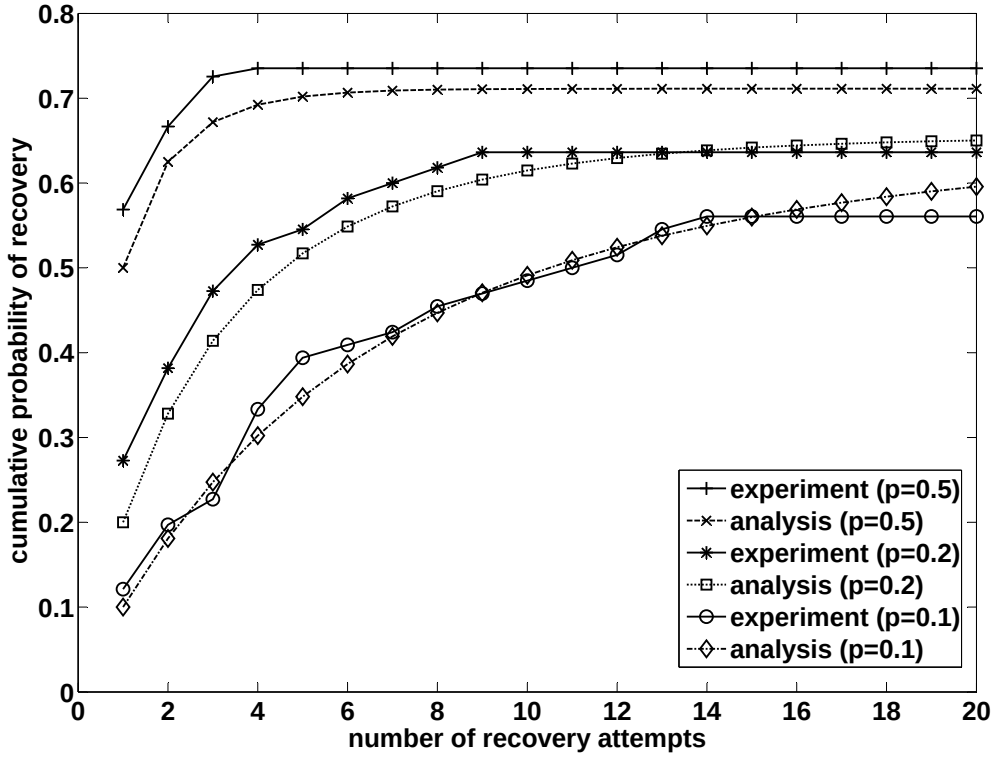


Figure 9: Comparison of experiments and analysis showing cumulative recovery probability as a function of number of attempts for one lost packet and $p = 0.5, 0.2, 0.1$

the value of m. These two fields, taken together, are then hashed and truncated to yield a 6-byte hash digest, which is added to the recovery field to enable a check on message integrity. The recovery field is then encrypted using a key m values back in the chain and the result is then appended to the data field to yield the complete message.

6. The completed message is then transmitted. For our implementation, we time broadcasts 150 ms apart.
7. To simulate a dropped packet for the motes, the transmitter intentionally suppresses packets at pre-programmed intervals.
8. The transmission concludes after 10,000 broadcast messages and the recovery results are compiled.

Receiver nodes are differentiated according to the number of 'previous' keys that they store on the basis of which they attempt to recover from packet

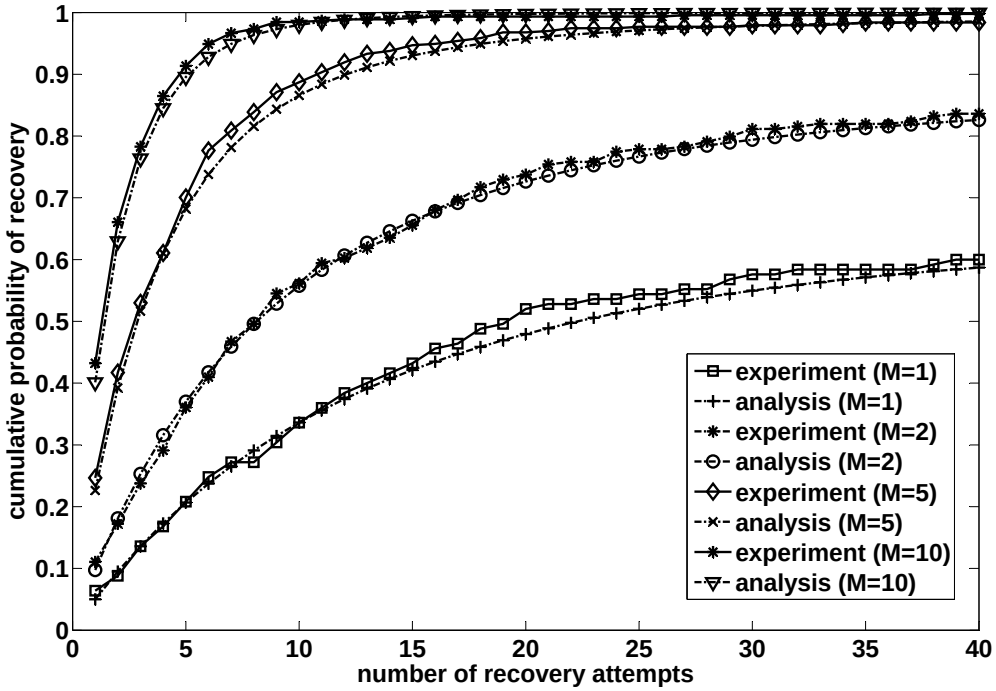


Figure 10: Comparison of experiments and analysis showing cumulative recovery probability as a function of number of attempts for one lost packet, $p = 0.05$, and varied number of stored keys $M = 1, 2, 5, 10$

loss. Previous keys are queued up in a buffer when they are received and used accordingly i.e. when recovering from packet loss, the most recently received key is tried first, then the one received before that, so on and so forth. We perform the experiment with nodes that store 1, 2, 3, 4, 5 and 10 keys respectively.

For typical broadcast messages, decryption of the data field and key update proceeds as outlined earlier. If the node drops a packet and is unable to decrypt a message, it attempts to decrypt the recovery field by successively applying previously stored keys.

For each packet loss/recovery, the node records how many attempts it took to recover from the loss and which key it used from its buffer. Non-recovery is logged as well. After the transmitter ceases broadcast, the nodes then transmit results wirelessly to a receiver to be collected and analyzed.

In Fig.9, we plot cumulative probability of recovery, P_k against number

of recovery attempts for parameter $p = 0.1$, 0.2, 0.5 to compare experimental results and simulation based on eq.3. The experimental results match fairly well with simulation and validate our analytical predictions.

In Fig.10, we plot experimental and analytical results of cumulative probability of recovery, $P^{(M)}$ for different number of receiver stored keys, $M = 1$, 2, 5 and 10 and fixed value of $p = 0.05$. Results again match with the analysis: and, as expected, using multiple keys to recover from key loss manifestly increases both a receiver's chances of recovery and the speed with which it recovers.

6. Conclusions and Future Work

Network broadcast schemes for wireless sensor networks that depend on time-varying keys for security are susceptible to key loss due to the wireless medium. In this paper we propose and analyze a key loss recovery mechanism to boost network robustness. The scheme is lightweight, secure, scales easily and is highly configurable. Receivers do not have to advertise their location by transmitting explicit key requests and the base-station does not have to individually service each receiver that has dropped keys. Confidentiality is maintained and message authenticity is guaranteed. The network operator has the option to tune the receiver's probability of recovery against the time it takes to recover and the local computational resources utilized in recovery. We analyzed our scheme and examined these tradeoffs both in simulation and on a MicaZ implementation. We believe our scheme has potential application in sensor networks where sensor broadcast is dynamic and of critical importance.

References

[1] Chang, S.-M., Shieh, S., Lin, W. W., Hsieh, C.-M., March 2006. An Efficient Broadcast Authentication Scheme in Wireless Sensor Networks. In: ACM Symposium on Information, Computer and Communication Security (ASIACCS). ACM, Taipei, Taiwan.

[2] Crossbow Technologies, online. Mica2 and MicaZ motes. URL `http://www.xbow.com`

[3] Dutta, P. K., Hui, J. W., Chu, D. C., Culler, D. E., April 2006. Securing the Deluge Network Programming System. In: International Conference

on Information Processing in Sensor Networks. ACM/IEEE, Nashville, Tennessee.

[4] Ganesan, D., Krishnamachari, B., Woo, A., Culler, D., Estrin, D., Wicker, S., Feb. 2002. Complex Behavior At Scale: An Experimental Study Of Low-power Wireless Sensor Networks. Technical Report UCLA/CSD-TR.

[5] Hui, J., Culler, D., 2004. The Dynamic Behavior of a Data Dissemination Protocol for Network Programming at Scale. In: SenSys. ACM, Baltimore, Maryland.

[6] Karlof, C., Sastry, N., Wagner, D., Nov. 2004. TinySec: A Link Layer Security Architecture for Wireless Sensor Networks. In: SenSys '04. ACM, Baltimore, Maryland.

[7] Karlof, C., Wagner, D., September 2003. Secure Routing in Wireless Sensor Networks: attacks and countermeasures. Elsevier Ad Hoc Networks 3, 293–315.

[8] Lanigan, P. E., Gandhi, R., Narasimhan, P., July 2006. Sluice: Secure Dissemination of Code Updates in Sensor Networks. In: International Conference on Distributed Computing Systems(ICDCS). IEEE, Lisboa, Portugal.

[9] Liu, A., Ning, P., 2008. TinyECC: A Configurable Library for Elliptic Curve Cryptography in Wireless Sensor Networks. Proceedings of 7th International Conference on Information Processing in Sensor Networks (ISPN 2008).

[10] Liu, D., Ning, P., November 2004. Multilevel μTesla: Broadcast Authentication for Distributed Sensor Networks. ACM Transactions on Embedded Computing Systems 3 (4), 800–836.

[11] Luk, M., Mezzour, G., Perrig, A., Gligor, V., April 2007. MiniSec: A Secure Sensor Network Communication Architecture. In: International Conference on Information Processing in Sensor Networks. ACM/IEEE, Cambridge, Massachusetts.

[12] Merkle, R., April 1980. Protocols for Public Key Cryptosystems. Proceedings of the IEEE Symposium on Research in Security and Privacy, 122–134.

[13] Perrig, A., Szewczyk, R., Wen, V., Culler, D., Tygar, J. D., 2001. SPINS: Security Protocols for Wireless Sensor Networks. In: Mobile Computing and Networking. ACM, Rome, Italy.

[14] Pickholtz, R. L., Schilling, D. L., Milstein, L. B., 1982. Theory of Spread-Spectrum Communications - A Tutorial. IEEE Transactions on Communications 30 (5), 855–884.

[15] Raimond A. Struble, P., 2005. INFINITE PRODUCTS RESCUED.

[16] Rivest, R. L., 1995. The RC5 Encryption Algorithm. Proceedings of 1st Workshop on Fast Software Encryption, 86–96.

[17] Rogaway, P., Bellare, M., Black, J., 2003. OCB: A Block-Cipher Mode of Operation for Efficient Authenticated Encryption 6 (3), 365–403.

[18] Shaheen, J., Ostry, D., Sivaraman, V., Jha, S., September 2007. Confidential and Secure Broadcast in Wireless Sensor Networks. In: IEEE International Symposium for Personal, Indoor and Mobile Radio Communications, PIMRC. Athens.

[19] The Legion of the Bouncy Castle, online. Cryptographic APIs.
URL `http://www.bouncycastle.org/`

[20] Trusted Computing Group, online. Trusted Platform Module Specifications.
URL `https://www.trustedcomputinggroup.org/specs/TPM/`

[21] US. National Security Agency, 1998. Skipjack and KEA Algorithm Specifications.

[22] Zhu, S., Setia, S., Jadojia, S., October 2003. LEAP: Efficient Security Mechanisms for Large-Scale Distributed Sensor Networks. In: Computer and Communications Security (CCS'03). ACM, Washington DC.

[23] ZigBee Alliance, June 2005. ZigBee Specification, Version 1.0. Technical Report Document.