# Broadcast Secrecy via Key-Chain Based Encryption in Single-Hop Wireless Sensor Networks

Vijay Sivaraman [a,b,*], Diethelm Ostry [b], Jaleel Shaheen [a,b],
Antoni Junior Hianto [a], Sanjay Jha [a,b]

[a] *University of New South Wales, Sydney, Australia.*

[b] *ICT Centre, CSIRO, Australia.*

## Abstract

Broadcast is used in wireless sensor networks for operations such as software updates, network queries, and command dissemination. Though authentication of broadcast messages is probably the most critical primitive, certain applications such as battlefield control and natural resource management also require the broadcast data to be kept secret from eavesdroppers. In this paper we design, implement, and evaluate a novel scheme that meets the requirements of secrecy, authenticity, integrity, and freshness of broadcast messages in the context of a single-hop wireless sensor network. Our contributions are three-fold: first, we propose the use of time-varying keys (based on a key-chain) for broadcast encryption, emphasising advantages such as non-forgeability, protection against old-key compromise, and allowance for dynamic data. Second, we extend the basic key-chain mechanism to incorporate limited protection against key loss, allowing legitimate receivers to recover even if they have lost a small number of keys. Third, we prototype our scheme by incorporating it into Deluge, the network programming protocol distributed with TinyOS, and quantify its cost in terms of time, space, and power consumption on a TelosB mote platform. Our scheme represents a practical, efficient and scalable means of delivering broadcast data secretly to a large number of low-power sensor nodes.

*Key words:* broadcast, secrecy, key-chain, encryption.

[*] Corresponding author: Vijay Sivaraman, School of Electrical Enginering and Telecommunications, UNSW, Sydney, NSW 2052, Australia. Email: Vijay.Sivaraman@ieee.org, Tel: +61 2 9385 6577, Fax: +61 2 9385 5993.

*Email addresses:* `Vijay.Sivaraman@ieee.org` (Vijay Sivaraman), `Diet.Ostry@csiro.au` (Diethelm Ostry), `z3161407@student.unsw.edu.au` (Jaleel Shaheen), `z3134754@student.unsw.edu.au` (Antoni Junior Hianto), `sanjay@cse.unsw.edu.au` (Sanjay Jha).

## 1 Introduction

Broadcast is an essential feature in any sensor network for critical operations such as network query, software updates, time synchronisation, and network management. Given its importance, there is growing interest in addressing broadcast security [1]. Much of the research literature has focused on *authenticity* of the broadcast source and data; we refer the reader to a recent article [2] that summarises the challenges of broadcast authentication in resource constrained wireless sensor networks. In this paper, we consider *secrecy* (also refered to as *confidentiality* or *privacy*) of the broadcast data. Several critical applications warrant secrecy, such as command and control signaling in the battlefield. The application that motivates this paper is a project undertaken by our organisation, the Commonwealth Scientific and Industrial Research Organisation (CSIRO), to build a Water Resources Observation Network (WRON) [3] to assist in managing and controlling the national water resources of Australia. CSIRO has developed sensor nodes called Flecks [4] which are candidates for deployment at sites such as farms, rivers, lakes, dams, and catchment areas. Secrecy of various broadcast data and control messages is important in such a scenario: for example, the sensory parameters (such as sampling periods and thresholds) that would from time-to-time be updated using broadcast mechanisms need to be kept secret, and software upgrades need to be kept confidential to prevent exploitation of code weaknesses. This paper develops mechanisms that operate within the resource constraints of sensor nodes to ensure secrecy of such broadcast data, while also guaranteeing authenticity, integrity, and freshness of the broadcast messages.

Several schemes, for example [5–9], have been proposed in the literature for broadcast *authentication*, but to the best of our knowledge there exists only one other proposal [10] that can provide *secrecy* of broadcast data in wireless sensor networks. We emphasise that our work was undertaken independently and concurrently to the work in MiniSec [10], and our approaches have fundamental differences . While MiniSec uses a fixed key (known to all parties) with a time-varying initialisation vector (IV), our approach uses a time-varying key (derived from a key-chain). As explained later, our method, though restricted in this work to single-hop networks (we have subsequently extended our scheme to multi-hop networks), provides authentication which is robust to key compromise unlike MiniSec. Lastly, we note that though several of the existing authentication schemes can be leveraged to incorporate secrecy, they either entail high storage requirements (e.g. [5]) or are cost-effective only for bulk data transfers (e.g. [6–9]) but not for sporadic transmission of broadcast data.

In this work we propose, design, prototype, and evaluate a practical method for incorporating secrecy, authenticity, integrity, and replay protection (aka freshness) of broadcast data in a wireless sensor network. Our work in this paper is restricted to **single-hop** networks. There are two reasons for this:

- First, broadcast secrecy is a very challenging problem. Symmetric encryption based on a static shared key requires all parties to know the key, which is problematic since receivers should only be able to verify but not originate valid broadcasts. Asymmetric encryption (which does not require all parties to share a key) is impractical on a per-packet basis due to high computation and communication overheads. New solution techniques are required, and to contain the complexity this paper considers the relatively simpler scenario of a single-hop network (as we will soon see solutions for even this restricted scenario are non-trivial). Our subsequent work in [17,18] has extended our solution technique to multi-hop networks, with corresponding increase in solution complexity (discussion of which is beyond the scope of the current paper).
- Second, single-hop transmission suffices in many application scenarios, particularly in hierarchically organised 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 a set of instructions to all sensor devices in a region in a single-hop fashion. In such networks multi-hopping may not even be desirable (for possible reliability and energy reasons). So there is indeed value in developing security solutions that apply to such single-hop networks.

Our novel approach to broadcast secrecy in this paper uses symmetric encryption but changes the encryption key on a per-packet basis using the known concept of a "key-chain", namely a set of successive keys derived from repeated one-way hashing of an initial key. For our **first** contribution we show how a key chain can be used for encrypting broadcast messages to ensure secrecy, authenticity, replay protection (freshness), and high message entropy (i.e. cipher-messages do not repeat even if the plain-text messages do). We also highlight several natural advantages of our approach, such as the ability to accommodate dynamic data, as well as protection against compromised keys (we note that the latter is not available in MiniSec [10]). For our **second** contribution we enhance the key-chain based scheme to incorporate limited resilience to key losses. With our method a node that has lost some keys gets a probabilistic opportunity (that diminishes with the number of lost keys) to recover the missing keys from the key chain, and the rate at which this opportunity diminishes can be adjusted system-wide to balance a node's recovery ability against an intruder's window of opportunity to compromise the key chain. As our **third** contribution we prototype our mechanism for secret broadcasts in the context of the network reprogramming protocol *Deluge* included in TinyOS, and present experimental results which quantify the associated time, space, and power overheads in a TelosB mote based single-hop network.

The rest of this paper is organised as follows: Section 2 defines the problem setting, solution requirements, and prior approaches from the literature. In Section 3 we describe our solution and discuss its properties, while Section 4 extends it to allow recovery from key losses. Section 5 describes our prototype implementation, with experimental results presented in Section 6, and the paper concludes in Section 7 with pointers to future work.

## 2 Problem Overview and Prior Solutions

This section defines the operating environment and threat model, outlines the solution requirements, and discusses relevant prior work in wireless sensor network broadcast security.

### *2.1 Operating Environment and Threat Model*

We assume a single-hop wireless sensor network in which a single source of broadcast data, called the base-station, can directly communicate with all sensor nodes. Single-hop topologies arise in applications ranging from battlefield command and control operations between a command centre (e.g. satellite or unmanned aerial vehicle) and deployed soldiers, to emerging body area networks for continuous health monitoring [11]. We assume that the base-station has abundant computation and energy resources, and cannot be compromised.

If the application warrants confidentiality of the broadcast data, the sensor nodes are expected to be protected against physical compromise. The sensor nodes in the WRON (water resources observation network) initiative developed at CSIRO are expected either to be physically inaccessible to attackers (e.g. in secured areas), or hardened by incorporation of tamper-resistant hardware such as a Trusted Platform Module (TPM) [12]. TPMs provide highly secure storage of cryptographic keys, along with secure hash storage for attestation and integrity verification of platform configuration, ensuring that physically captured nodes cannot be made to reveal cryptographic keys or have their software altered without detection. If one or more nodes in the network are not compromise-resistant, confidentiality of the broadcasts is unavoidably put at risk, though authenticity of all broadcasts *can* still be ensured.

The wireless medium is by nature broadcast and hence a passive eavesdropper can listen to all transmissions. An active intruder can transmit arbitrary messages, or replay a valid captured message at a later time. We make no assumptions about the number of intruders, their locations, their radio range, or the degree of collusion amongst them. In the case where nodes are not hardened against physical compromise, no assurances on data confidentiality can be given if an intruder can extract the cryptographic keys. Nevertheless, we assume it is an operational requirement that authenticity of the broadcast source and data not be sacrificed even if one or more sensor nodes are compromised. We assume that the intruder does not have the capability to block reception of packets at an uncompromised node; such "jamming" will allow the intruder to act as an intermediary between the base-station and a receiver, in effect making the network multi-hop which is beyond the scope of this paper. Finally, we do not explicitly address denial-of-service or battery-drain attacks.

### *2.2 Solution Requirements*

We seek a security mechanism that provides the following properties for broadcast traffic in a single-hop wireless sensor network:

(1) **Confidentiality:** The broadcast data should be kept secret from eavesdroppers. As noted earlier, confidentiality cannot be guaranteed if one or more nodes in the network are physically compromised.
(2) **Authenticity:** Messages not originating from the base-station should be discarded (ensuring source authenticity), as should messages that have been tampered with (ensuring data authenticity, also known as **message integrity**). Note again that authenticity should be guaranteed even if one or more sensor nodes in the network are compromised.
(3) **Freshness:** Packets that have been captured and replayed at a later time should be discarded by the sensor nodes.
(4) **Semantic security:** Even if the broadcast messages are chosen from a small set, the encryption should produce ciphertext that does not give information to an intruder about which of these messages was sent.
(5) **Dynamic data:** The scheme should be cost-effective even when the content of the sequence of broadcast messages is not known in its entirety before-hand by the base-station. For example, the scheme should be efficient not just for broadcast file transfers (e.g. a new code image), but also for short dynamic broadcast messages (e.g. battlefield commands).
(6) **Delay Tolerance:** No time synchronisation should be required in the system.
(7) **Incremental processing:** Each received packet must be immediately verifiable without having to wait for additional data.
(8) **Resilience to loss:** A receiver that loses a small number of packets should be able to receive and read subsequent broadcast messages.

In section 3.2 we will discuss how our proposed scheme meets the above requirements.

### *2.3 Prior Proposals*

We now briefly summarise exisiting schemes for broadcast security in wireless sensor network that are relevant to the curent work.

We are aware of only one existing scheme, MiniSec [10], that provides for secrecy of broadcast (and indeed of unicast) messages in wireless sensor networks. MiniSec broadcast requires the sender and all recipients of the broadcast to hold a shared key $K$. Further, time is divided into "epochs" and all broadcast participants have clocks that are loosely synchronised to within an epoch. A broadcast message payload $M$ is appended with a nonce (which is a combination of the packet counter and the epoch number), and then encrypted using offset code-book (OCB) mode [16] of block cipher encryption. OCB encryption essentially makes the payload and nonce non-separable in cipher-text, and a receiver can thus verify authenticity of the message by checking that the nonce obtained post-decryption matches the expected counter value. The use of OCB therefore provides both secrecy and authenticity in MiniSec. The loose time-synchronisation in MiniSec poses some concerns about replay attacks within an epoch, and these are addressed via use of Bloom filters to detect and discard such replayed packets. Though MiniSec operates in general multi-hop networks, we believe its fundamental weakness (when applied to broadcast) lies in the assumption that the shared key can be kept safe at all nodes. Even if one node in the network were to be physically compromised by an intruder to obtain the shared key, they could forge messages that would pass the authenticity tests at other nodes. In other words, MiniSec does not satisfy the second desirable property listed in the previous subsection, which requires authenticity of broadcast messages to be guaranteed even if one or more nodes in the network are compromised.

We now summarise a few relevant broadcast authentication schemes (that do not provide secrecy). TinySec [13] develops mechanisms for symmetric-key encryption of data at the link-layer of the communication protocol. Though TinySec does not mandate how the encryption key is derived, the expectation is that the key would have a long lifetime and would be shared by all parties involved in the communication. As mentioned earlier, this is problematic for broadcasts, since receivers are untrusted and can potentially use the shared key to forge broadcast messages.

The $\mu$TESLA [5] protocol overcomes the above problem by using symmetric-key encryption with time-varying keys. The base-station constructs a key chain by repeatedly applying a hash function to an initial random value, and the root-key (the last hash value obtained) is distributed to each node securely based on a pre-distributed symmetric key. The chain construction allows nodes to verify that disclosed keys are authentic. Loose time-synchronisation of the network into regular time intervals is assumed, and the base-station uses a single key from the key chain for the whole duration of a time interval. The key is disclosed by the base-station at a later time, when nodes can verify that the key is a valid member of the chain, the message authentication codes (MACs) of stored broadcast packets are correct, and that the time delay is such that only the base-station could have constructed the received packets. Some of the drawbacks of this scheme are the need for network-wide (loose) time synchronisation, and the high storage requirement (of potentially malicious or vacuous packets) at each node until the authenticity of the packets can be verified (i.e. after the relevant key is disclosed).

Several schemes have been proposed recently [6–9] for authentication of broadcast messages in the context of network programming. A network programming protocol called Deluge [14], which is included by default in the TinyOS distribution, allows multi-hop broadcast dissemination of new code images on mote-based platforms. In the absence of authentication, an arbitrary node under Deluge could broadcast new versions of the software, disseminate malicious packets, program any number of nodes, and take over the operation of the entire network. In [6], the authors of Deluge have extended their scheme to incorporate authentication of the program image. Their scheme, which we term *SecDeluge*, uses a hash-chain to verify authenticity of received packets. The base-station sends the code update in a sequence of packets, each of which includes the hash of the next packet to be sent. A node receiving the broadcast packet stores this hash value, and compares it to the value obtained from hashing the next received packet, thus making an immediate decision as to whether the packet is authentic and in sequence. The initial packet is digitally signed so the initial hash value is authenticated. *Sluice* [7] is very similar to SecDeluge except that the hash in the chain is computed over "pages" rather than packets (where a page typically carries around 1KBytes of the program image). Deng *et. al.* [8] also employ a signed hash scheme, but use a tree structure that allows packet verification even when packets arrive out-of-order. A recent extension called *Seluge* in [9] further enhances security in Deluge to address various DoS attacks.

## 3 Our Scheme for Secret Broadcasts

In this section we describe and discuss our scheme for guaranteeing secrecy and authenticity of broadcast messages in single-hop wireless sensor networks.

### *3.1 The Procedure*

As stated earlier, our scheme relies in the use of a chain of keys, one key per packet, as depicted in Fig. 1, and described by the steps below:

(1) **Key-chain generation:** The base-station (BS) selects an arbitrary random key $k_M$, and from it generates a key chain $k_M$, $k_{M-1}$, $\ldots$, $k_1$, $k_0$, where $k_{i-1} = H(k_i)$ for $i = 1, \ldots, M$, where $H(.)$ denotes a hash operation (such as SHA1 or MD5). The length $M$ of the chain can in principle be arbitrarily large (allowing the chain to be used for broadcasting as many as $M$ data packets), but practical designs should bear in mind that the key-width (i.e. number of bits in the key) will limit the number of unique keys obtained by hashing

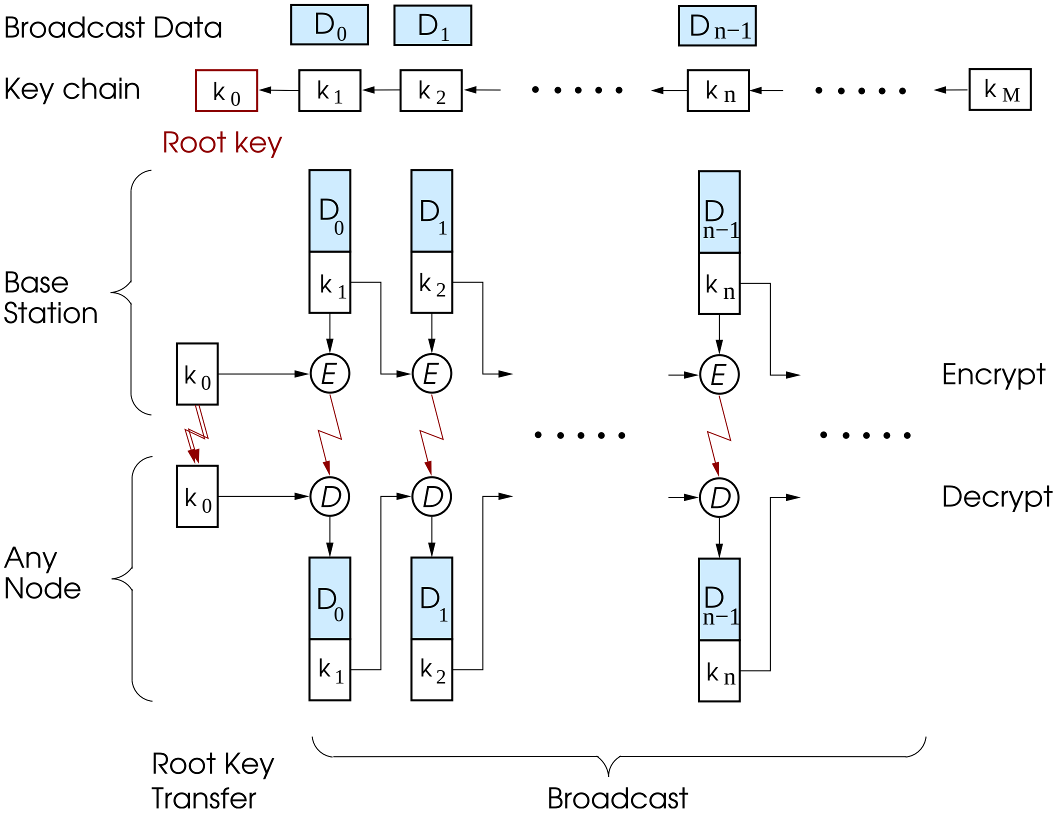

Fig. 1. Key chain encryption of broadcast packets

– successive hashing will ultimately yield repeated keys in the chain, which should be avoided to prevent key reuse. One should also bear in mind that a larger key chain length $M$ also necessitates larger processing time and storage space at the base station.

(2) **Bootstrapping:** The key commitment $k_0$, which we term the "root-key", needs to be securely conveyed to each target sensor node. The root-key could be programmed into the sensor nodes prior to deployment (if the key-chain in the previous step is long enough to be used for the expected lifetime of the node), or one of several key management schemes [15] can generate dynamic keys for secure distribution of the root key to each individual node. The mechanism for root-key distribution is very application specific, and we outline our approach in Section 5 in the specific context of a network programming application.

(3) **Data transmission:** Once all target sensor nodes have the root key, the base-station 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 using key $k_0$ (see Fig. 1). The encryption scheme must ensure that the encrypted data and encrypted key are not separable in ciphertext, so that any modification of the encrypted data also destroys the key. Such message integrity is guaranteed, for example, by the offset code-book (OCB) mode [16] of block cipher encryption. 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 (see Fig. 1). 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 $M$ broadcast packets have been sent, thereby using up all the $M$ available keys, the base station will have to return to step 1 to generate a new key chain before it can continue to send broadcast messages securely.

### *3.2 Discussion*

As described above, the key chain in our scheme serves the dual purpose of ensuring both secrecy and authenticity of the broadcast data. The non-forgeability of the successor key in a received packet derives from the authenticity of the contents of that packet - this necessitates the more sophisticated OCB block cipher encryption that prohibits any part of the broadcast message from being modified without also modifying the part that holds the successor key. In spite of its increased complexity, the advantage of this approach is that the authentication mechanism is decoupled from the actual broadcast data itself, which is particularly useful in scenarios where the broadcast data is not known before-hand. By contrast, the broadcast authentication schemes proposed in [6–9] compute a hash of the broadcast data itself, with the initial hash being digitally signed. While such an approach is acceptable for bulk data transfer applications (such as network programming), where the cost of initial secure key exchange can be amortised over the broadcast, it is not efficient for applications that require dynamic or short broadcast messages to be sent at regular or irregular intervals, as may occur in battlefield control and asset monitoring applications.

Our approach for guaranteeing secrecy (in conjunction with authenticity) is fundamentally different from that of MiniSec [10]. Though both schemes rely on the use of OCB to make the payload and nonce non-separable in cipher-text, the difference in choice of nonce leads to different properties. MiniSec uses an incrementing counter (the packet number concatenated with the epoch number) as nonce; this is simple, allows multi-hop transmission (provided there is loose time-synchronisation in the system), and is resilient to loss. However, it does not preserve message authenticity if a node is physically compromised yielding the shared key and counter. Our scheme, by contrast, uses the predecessor key of the key chain as the nonce. This makes authentication slightly more complex (since the received nonce has to be hashed and then matched against the stored key), but provides strong guarantees on message authenticity even if one or more shared keys are compromised, since a key is never reused. This additional property of our scheme comes at an expense: extension to multi-hop networks requires more complex solutions, as we outline in [17,18], and recovery from key loss also requires a more elaborate mechanism (described in section 4). Nevertheless, we believe our approach is more suited to networks where authenticity is vital even if secrecy is compromised (e.g. battlefield applications), whereas MiniSec may better suit deployments in which secure key storage is guaranteed and key compromise is therefore not a concern.

Our use of a key chain is most similar to the scheme used by $\mu$TESLA [5]. However, there are some major differences since $\mu$TESLA is designed for authentication only while our scheme provides secrecy as well. Our scheme uses the keys for encrypting data, while $\mu$TESLA uses the keys for computing message authentication codes (MACs) to validate the data. $\mu$TESLA discloses keys some time after the data has been transmitted (requiring storage of packets), whereas we send the key to decrypt a packet in the preceding packet and hence do not require storage of any (potentially malicious) packets. Lastly, $\mu$TESLA uses network-wide loose time-synchronisation, with a single key from the key chain being used for the whole duration of a time interval, while our scheme completely eliminates key reuse by changing the key from packet to packet.

All the proposed broadcast security protocols require an initial commitment step: the signed first packet or page in SecDeluge, Sluice, and Seluge commits to a data hash chain, while the root-key in $\mu$TESLA and in our scheme commits to a key chain. Confidentiality of the broadcast data requires the initial commitment to be transmitted secretly by the base-station to each target node individually. While this may be computationally expensive (unless the root-key is pre-deployed at all nodes), it is unavoidable if secrecy is required by the application. We do however note that the bootstrapping operation can be time-overlapped in nodes so that for large networks, the time needed by the initialisation step is limited by communication time requirements rather than the computational load.

Our scheme does not guarantee authenticity in a multi-hop network if one or more sensor nodes are compromised. This is because a compromised transit node in a multi-hop network can 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. The extension of our scheme to multi-hop networks is beyond the scope of this paper, and is being addressed by our current research in [17,18] by use of multiple one-way key chains.

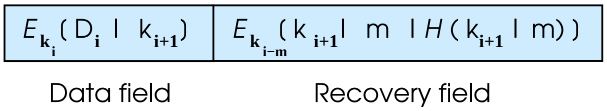
$E_{k_i}(D_i \mid k_{i+1})$
$E_{k_{i-m}}(k_{i+1} \mid m \mid H(k_{i+1} \mid m))$
Data field
Recovery field

Fig. 2. Packet structure augmented to include recovery information

## 4 Recovering from Key Losses

A drawback of using the key chain approach above is that a receiver which misses even one broadcast packet is effectively excluded from all future broadcast messages: this happens because 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 like Deluge), 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 (for example the base-station can periodically repeat the data), but the loss of the key contained in the packet is a problem (our scheme prohibits key reuse for fear of replay attacks). We believe a scheme that allows a receiver to recover from one or more lost keys should have the following important properties:

(1) The recovery scheme should balance a receiver's ability to recover against the overall vulnerability of the system. Specifically, it should assist a receiver that has lost one or a few keys to recover at sufficiently low computational cost, but it should limit the ability of an attacker, who has obtained a previous (old) key, to decrypt ongoing broadcast messages.

(2) The recovery scheme should scale well to large numbers of heterogeneous receivers. In other words receivers should be able to make independent decisions on the effort they want to invest in recovery, and should also not individually request assistance in recovery (thereby keeping their location secret).

With these requirements in mind, we propose an extension to our basic key chain scheme above that allows recovery from packet loss. 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 of course to allow a node that has missed $m$ previous broadcast packets to use its old key 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 packets missed
3.      replace current_key with extracted_key
4.      process data
5.  else // packets 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 formally 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.

The above scheme allows a receiver that has missed $m$ packets (since its last success) to reattach using the recovery information contained in the received broadcast packet, only if the base-station uses the same number $m$ in constructing the recovery information contained in the 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 $(1-p)^{m-1}p$. This scheme meets the requirements enumerated earlier in this subsection:

- A receiver's ability to reattach to the broadcast session falls exponentially with the number of broadcast packets it has missed since the last time it was attached. This allows a smooth trade-off between the network's resilience to losses and its vulnerability to attackers: a trusted receiver that has lost some packets has the opportunity to reattach, but an attacker has limited time to compromise a key in the chain and attach to the network (since old keys become exponentially less useful with time).
- The parameter $p \in (0, 1)$ that determines the range over which recovery is most effective can be adjusted system-wide to choose the desired trade-off point between network resilience and attack resistance. If $p$ is large, receivers that have lost one or a few packets can recover quickly, but the chances of recovery for a node that has missed many broadcast packets becomes vanishingly small. To take an example, consider a large $p = 0.5$ and a small $p = 0.1$. A node that has missed only $1$ packet has a chance of recovery $0.5$ and $0.1$ respectively for the large and small $p$ values above, whereas a node that has lost $10$ packets has probability $0.1\%$ and $3.9\%$ respectively for the large and small $p$ values above, thus showing that small $p$ improves loss resilience at the expense of increasing the vulnerability of the system to compromised keys. The operator of the network can choose an appropriate trade-off point depending on application requirements and expected operating conditions.
- The recovery scheme does not penalise receivers that do not need recovery (beyond the cost of receiving the recovery field), and receivers which require recovery spend a computation time linear in the number of missed packets, as seen in Fig. 3. A receiver that has the most recently used key (i.e. has not missed the previous packet) will satisfy the check in step $2$ and ignore the recovery information, hence paying no performance penalty. A receiver that has lost $m > 1$

packets, or receives a malicious packet, has to perform the normal decryption and hash check in steps 1-2, as well as the decryption and hash check in steps 6-7. Malicious packets, as well as packets that will not aid in recovery, will fail the check in step 7 and be discarded. Packets containing usable recovery information will have the key validated (step 8) in time proportional to the number of lost packets (this step protects against a sophisticated attacker who uses an old compromised key).

- Our recovery scheme requires local computation at the receivers but no radio transmissions; this makes the scheme scaleable to a large number of receivers, and is also attractive in scenarios where node location is required to remain hidden.

We believe the recovery scheme described above is amenable to implementation in applications where secrecy and authenticity of broadcast data is important but where reliable broadcast delivery is infeasible or undesirable.

## 5 Secrecy for Code Image Broadcasts: An Implementation

We undertook a first prototype implementation of our scheme in the context of network programming, namely for broadcasting new code images from a base-station to multiple target sensor nodes. Our implementation is based on the Deluge network programming protocol [14] that is distributed with TinyOS. Deluge divides a program image into pages (typically of size 1104 bytes each), and each page is transmitted in multiple packets (typically 48). A page when successfully received is stored in flash memory by each target sensor node. The lack of security is a well-known shortcoming in Deluge, and prior schemes such as SecDeluge [6], Sluice [7], and Seluge [9] mentioned earlier have extended Deluge to incorporate code image authentication. None of these proposals however ensure privacy of the code image broadcast. Our scheme, which we call *PrivCIB* (*Priv*ate *C*ode *I*mage *B*roadcast), implements privacy and authentication of Deluge packet transmissions. We emphasise to the reader that at present our scheme is limited to single-hop systems where the base-station broadcasts new code images directly to all sensor nodes; extension to the true multi-hop "epidemic" dissemination mechanism of Deluge is deferred to future work. We also note that the key loss recovery mechanism outlined earlier in section 4 above is unnecessary in this application since Deluge has in-built mechanisms for reliable packet delivery.

Our implementation platform comprised a PC (running the *cygwin* environment on Windows XP) acting as a base-station and TelosB motes [19] (commercially available from Crossbow Technology Inc.) running TinyOS as target sensor nodes. The base-station was implemented in Java using the BouncyCastle JCE provider [20]. The key size for symmetric key encryption was chosen as 8 bytes; even though real deployments would use larger keys for high security, we chose the key size in our implementation to be compatible with the RC5 encryption algorithm available in TinySec [13]. In the first step the base-station creates the key chain: it chooses an initial $8$-byte random number and hashes it using the SHA1 algorithm (with the lowest $8$ bytes of the $20$-byte result being used as the next key). The hashing was repeated to create a chain of length $4000$, which is sufficient for transfering the program images we considered. New Java classes were created for the key-chain establishment, and the Deluge Java toolchain code in the file *DelugeImageInjector.java* was modified for the data transmission operations.

We did not assume that the root-key is pre-shared between the base-station and all sensor nodes; instead a bootstrap phase was implemented to use public-key cryptography to convey the root-key securely. The base-station holds a public/private key pair, of which the public key is known to all sensor nodes. Each sensor node also holds a public/private key pair, and the base-station knows the public key of each sensor node that is to receive the broadcast data. Note that the public keys are required only during bootstrapping to establish initial trust; thereafter shared symmetric keys are used for data encryption. The bootstrapping phase is implemented using elliptic-curve public-key cryptographic operations, which have been show to be feasible for resource-constrained sensor nodes [21].

A simple way for the base-station to deliver the root-key to a particular target sensor node would be for it to use the target's public key to encrypt the root-key. However, this is susceptible to capture and replay by an adversary at a later time, potentially allowing the attacker to revert the sensor nodes to an earlier code image. To protect against this, we implement an authenticated Diffie-Hellman exchange first to generate a secure channel, and then to use that channel for the root-key transfer. The base-station initiates the Diffie-Hellman exchange by sending a digitally signed message containing its ephemeral key component, and the target node responds correspondingly with its own signed ephemeral key component. The shared ephemeral key is then generated by each side by combining the received key component with its own key component. This shared key allows secure transfer of the root-key $k_0$ via symmetric encryption. The ephemeral nature of the shared key protects the Diffie-Hellman exchange against capture-and-replay attacks, while authentication via digital signature prevents an intruder from masquerading as the base-station or as a sensor node, and protects against man-in-the-middle attacks. We implemented the ECDH (Elliptic Curve Diffie-Hellman) using primitives from the EccM package [21] from Harvard University, while the EC-DSA (Elliptic Curve Digital Signature Algorithm) was taken from the TinyECC package [22] developed at North Carolina State University. The entire procedure is repeated by the base-station for each target node of the broadcast.

Once the root-key has been sent to all nodes, the actual broadcast data transfer can begin. We used the RC5 encryption algorithm available from the TinySec [13] implementation, and incorporated it into Deluge's NesC file *DelugePageTransferM.nc*. The packet structure of Deluge was modified so that in addition to the

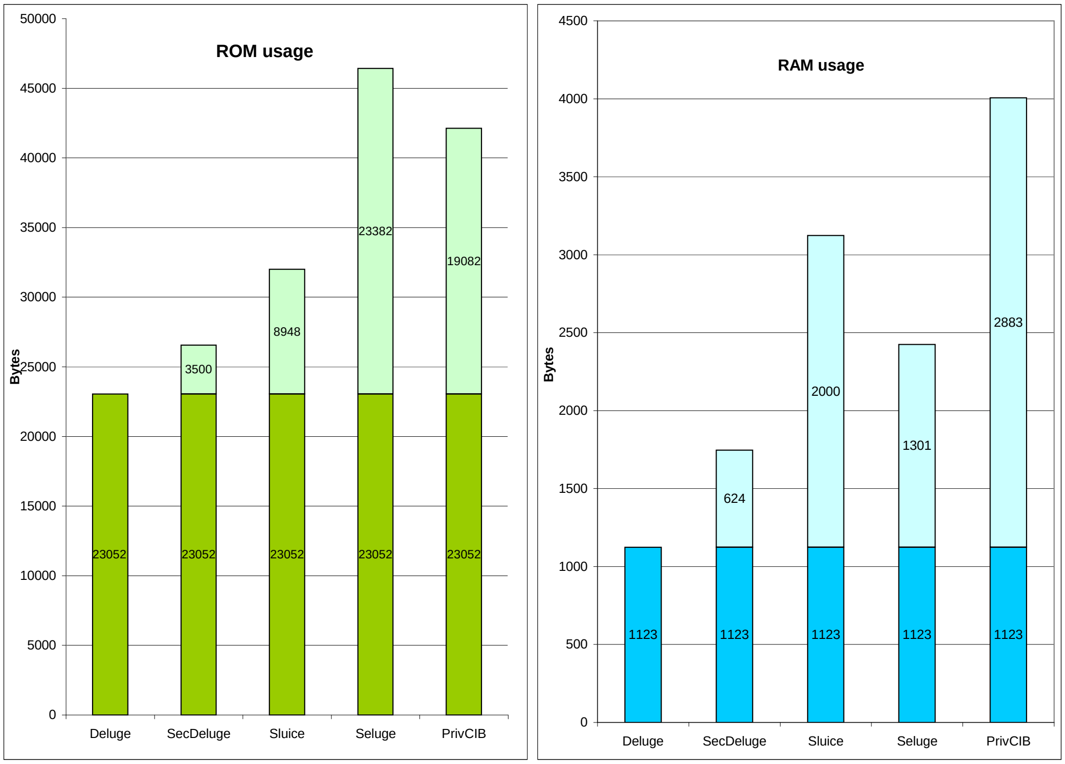

Fig. 4. ROM and RAM usage of Deluge, SecDeluge, Sluice, Seluge, and PrivCIB.

$23$ bytes of data in the packet, $8$ bytes of key was included corresponding to the successor key in the key chain. The additional $8$ bytes per packet constitutes an overhead of $384$ bytes per page (which contains $1104$ bytes of the program image).

We did not optimise our cryptographic routines for efficiency and performance. Fig. 4 compares the memory usage of our scheme PrivCIB (which performs both authentication and encryption) to prior schemes SecDeluge, Sluice, and Seluge (which perform only authentication). Our scheme requires approximately $19$KB more program memory than Deluge, and approximately $3$KB more RAM data storage space than Deluge. The ROM and RAM requirements of our scheme are only slightly higher than the other schemes, which is an acceptable price to pay for keeping the broadcast data secret. Our prototype is intended as a proof-of-concept; a production implementation would reduce both the ROM and RAM requirements by removing duplication in the ECC routines between the EccM and TinyECC packages, and will be addressed by our future work.

## 6 Experimental Results

This section profiles our PrivCIB scheme in terms of the time taken as well as the energy consumed for transfering program images of various sizes. Our first experi-

| Image | Size | Deluge | | PrivCIB | |
|---|---|---|---|---|---|
| name | (pages) | Time (sec) | Energy (mJ) | Time (sec) | Energy (mJ) |
| Blink | 3 | 10.1 | 756 | 19.7 | 2032 |
| Oscilloscope | 9 | 20.2 | 2024 | 49.1 | 4549 |
| Pong | 11 | 25.6 | 2506 | 59.2 | 5317 |
| TinyECC | 23 | 47.7 | 4787 | 116.7 | 10471 |
| PrivCIB | 39 | 79.7 | 7940 | 180.3 | 17531 |

Table 1
Program images and transfer time / energy using Deluge and PrivCIB

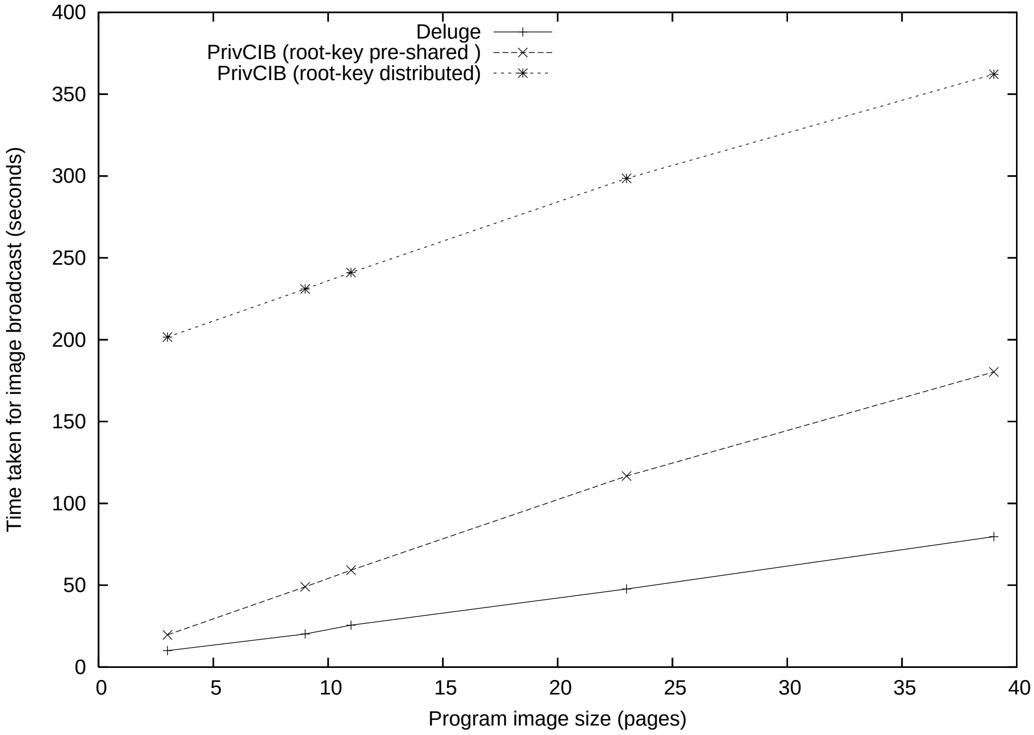

Fig. 5. Transfer time from Base-Station to one node

ment considers a single target sensor node. The time taken for the various steps was measured by incorporating program code to switch the three LEDs on the motes on or off at various stages of the algorithm, and timing such changes manually with a stopwatch. Energy consumption was obtained by integrating the product of the voltage and current used by the sensor node during the image transfer, measured using a USB connected PC oscilloscope manufactured by Cleverscope [23].

Table 1 shows measurements for the transfer of five program images (four of which are supplied as examples in the TinyOS distribution) of sizes ranging from $3$ to $39$ pages (recall that each page holds $1104$ bytes of data). The time taken for the image transfer when using Deluge and when using our PrivCIB scheme (with and without

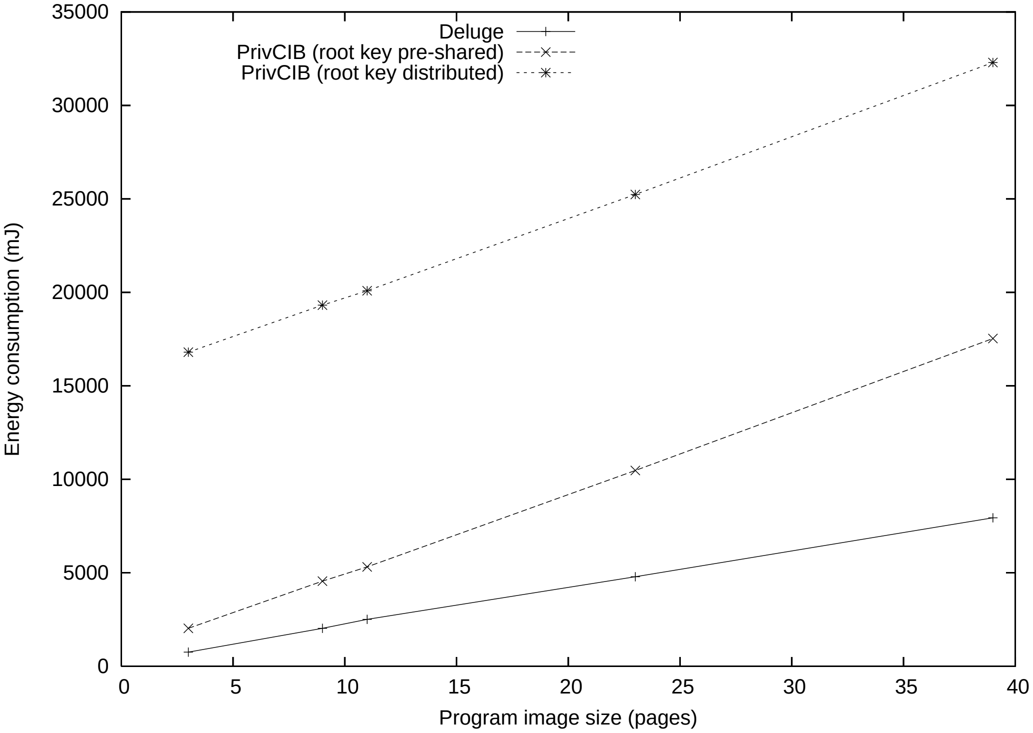

Fig. 6. Total energy consumption on target node for image transfer

the root-key distribution phase) is shown in Fig. 5, while the corresponding energy consumption is plotted in Fig. 6. The experiments were repeated several times and found to give consistent results, so error-bars are not shown.

Several observations can be made from these plots: first, the time and energy requirements under both schemes grow linearly with program image size, since the operations performed by the nodes are largely repetitive from page to page. Second, the time/energy costs of PrivCIB (excluding the root-key distribution phase) are generally a factor of $2$-$2.5$ that of Deluge; this represents the price penalty for having secrecy of the program image broadcast. Third, the dynamic distribution of the root-key (involving the authenticated Diffie-Hellman exchange) in PrivCIB requires a constant time of approximately $180$ seconds which is independent of the program image size, and this is reflected in the constant vertical distance between the top curve (that includes root-key distribution) and the middle curve (that assumes a pre-shared root-key) showing the time/energy requirements of PrivCIB in the plots.

One would expect the computational cost of PrivCIB to be substantially higher compared to Deluge (due to the encryption and hash operations), and the communication costs to be only marginally higher (approximately $35\%$, corresponding to the additional $8$ bytes per packet of $23$-byte payload). Since in general, computation is expected to use much less energy than communication [24], it is surprising that the energy requirement of PrivCIB shown in Fig. 6 grows at a similar rate to its time

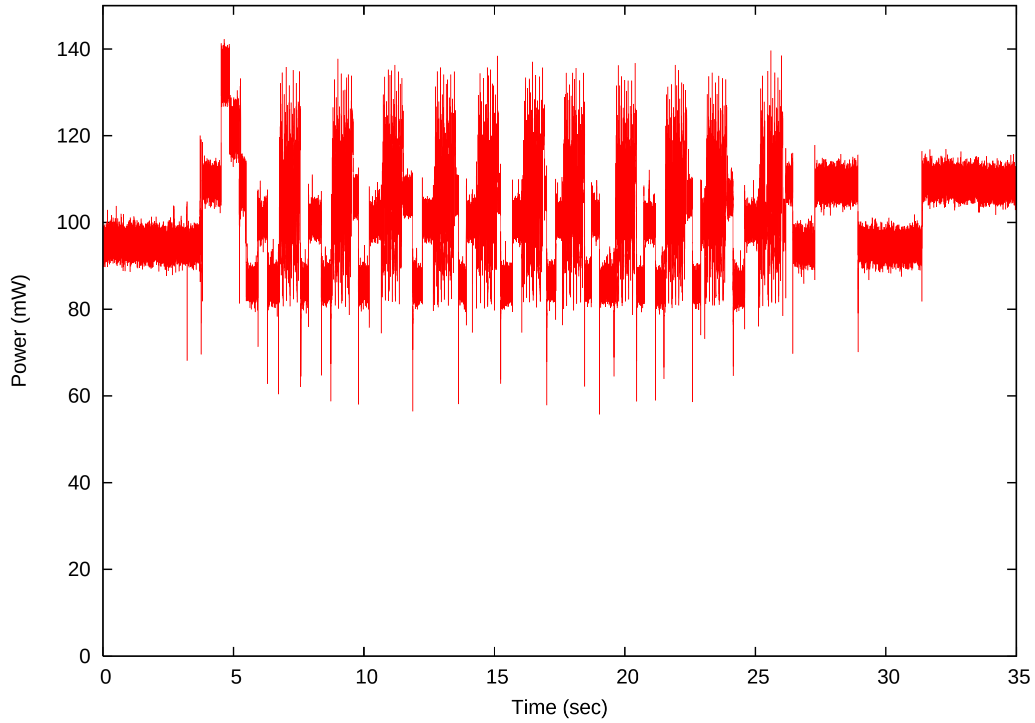

Fig. 7. Power consumption trace when transfering program “Pong” using Deluge.

requirement shown in Fig. 5. Investigation revealed that this is because the radio in the sensor node is never put into sleep mode. The high base load energy consumption rate of the radio even in idle mode masks the incremental power consumed by the processor when performing the cryptographic operations in PrivCIB. Modifying the MAC protocol to incorporate sophisticated duty-cycling techniques to put the radio in sleep mode is beyond the scope of this paper; instead, we resorted to closer analysis of the power traces obtained while the protocol was in operation in order to identify the regions where energy consumption increases. We present here traces obtained during the transfer of the “Pong” image, which is $11$ pages long, with and without our secrecy enhancement.

Fig. 7 shows the trace of the power consumed by the target sensor mote when using Deluge for the transfer, while Fig. 8 shows the power consumption when using our PrivCIB scheme. The voltage and current supplied to the sensor node were sampled at approximately $65$ KHz using the Cleverscope USB oscilloscope. To reduce plot size each data point of the plot is the average of $10$ successive samples. Both figures show an initial region of increased power consumption (in the range $4$-$5.5$ sec for Deluge in Fig. 7 and $6$-$7.5$ sec for PrivCIB in Fig. 8), where the new program image is advertised/requested as part of the dissemination protocol. Thereafter, there are exactly $11$ regions in either plot that show a marked increase in power usage: these correspond to the $11$ pages that are transferred as part of the “Pong” image. Each of these $11$ regions of activity involves successive reception of $48$ packets, followed by a write operation of the entire page to flash memory. Table

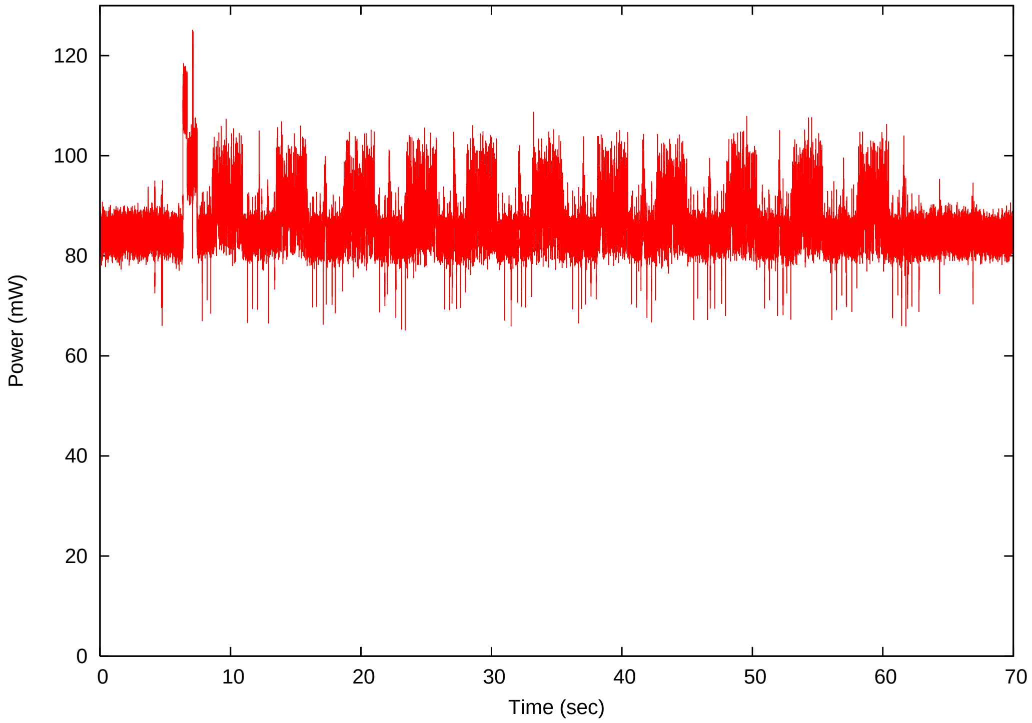

Fig. 8. Power consumption trace when transfering program “Pong” using PrivCIB.

2 shows the time and energy required for transfering each of the $11$ pages of the “Pong” image under either scheme, computed from the above traces. As expected, PrivCIB requires more time for the transfer of each page, taking $2.41$ sec on average compared to $0.88$ sec in Deluge. Correspondingly the average energy consumption per page with PrivCIB is $215.1$ mJ compared to $91.4$ mJ in Deluge. The increased time and energy requirements in PrivCIB are attributable to the need for packet decryption (RC5) and key verification (SHA1) on a per-packet basis, and also the larger packet size itself due to the need to include the successor key. Nevertheless, the time and energy cost for incorporating secrecy and authenticity via our scheme is within a factor $2.5$ of standard Deluge without any security.

We also profiled the bootstrap phase that uses the authenticated Diffie-Hellman exchange to distribute the root-key: Table 3 lists the time and energy costs of the various steps involved. As can be seen, the majority of the time and energy is spent in creation and verification of the digital signatures. While earlier proposals like SecDeluge, Sluice, and Seluge require the first packet of every image transfer to be digitally signed, our scheme can easily amortise that cost over several image transfers (or indeed any arbitrary broadcast message transmissions) by using a sufficiently long key chain.

Finally, we also tested and profiled our PrivCIB scheme for upgrading a software image on multiple nodes in a single-hop topology. Fig. 9 compares the time taken for upgrading $1$, $2$, $4$, and $10$ nodes (note that the vertical scale in this plot starts at $200$ seconds), and emphasises that upgrading each additional node incurs only a

| Program | Deluge | | PrivCIB | |
|---|---|---|---|---|
| page number | Time (msec) | Energy (mJ) | Time (msec) | Energy (mJ) |
| 1 | 847.1 | 86.2 | 2399.7 | 215.5 |
| 2 | 893.3 | 93.9 | 2433.6 | 217.4 |
| 3 | 838.1 | 88.4 | 2433.6 | 216.9 |
| 4 | 904.0 | 93.6 | 2387.7 | 213.1 |
| 5 | 914.6 | 94.2 | 2442.4 | 217.0 |
| 6 | 914.7 | 94.5 | 2431.8 | 216.6 |
| 7 | 836.3 | 85.7 | 2433.6 | 216.2 |
| 8 | 870.2 | 88.8 | 2421.1 | 215.3 |
| 9 | 1060.5 | 109.6 | 2355.6 | 209.2 |
| 10 | 838.1 | 86.5 | 2431.8 | 216.7 |
| 11 | 804.3 | 83.5 | 2376.5 | 212.1 |
| **Average** | **883.7** | **91.4** | **2413.4** | **215.1** |

Table 2
Time and Energy for each page of “Pong” under Deluge and PrivCIB

| Task | Time (sec) | Energy (J) |
|---|---|---|
| Creation of ECC keys | 53.2 | 3.25 |
| Verification of digital signature | 59.8 | 5.54 |
| Creation of digitally signed message | 35.7 | 3.58 |
| Sending message to base-station | 3.7 | 0.38 |
| Waiting period | 1.3 | 0.12 |
| Diffie-Hellman key exchange | 5.5 | 0.57 |
| Decryption of root-key | 9.7 | 0.79 |
| Sending message to base-station | 3.6 | 0.33 |
| **Total** | **172.5** | **14.56** |

Table 3
Root-key distribution steps and their time / energy usage

small additional cost, thus preserving the advantages of broadcast.

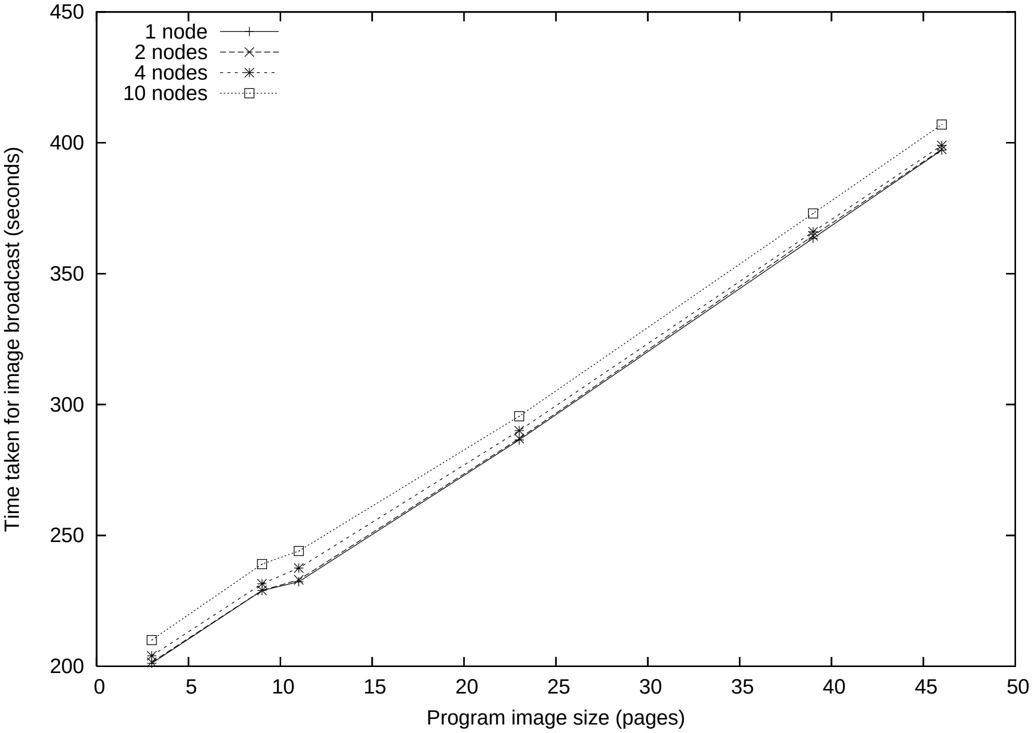

Fig. 9. Transfer time from Base-Station to multiple nodes

## 7 Conclusions and Future Work

Critical wireless sensor networks deployed in defence and resource management applications will require broadcast data to remain confidential. In this paper we developed a practical and efficient mechanism that uses low-complexity symmetric-key cryptography with a time-varying key derived from a key-chain in order to guarantee confidentiality, authenticity, freshness, and semantic security of broadcast data, while allowing the broadcast data to be dynamic and incrementally processed. We also proposed a scalable extension to the scheme that allows receivers to recover from loss of one or a few keys, with an associated penalty in system vulnerability to compromised keys as well as processing and communication costs, that can be adjusted system-wide. Finally we implemented a prototype of our mechanism as an add-on to the broadcast-based Deluge network reprogramming protocol in off-the-shelf TelosB motes running TinyOS. Our experiments show that the time and energy required to broadcast a page of a program image confidentially and securely to multiple nodes using our scheme is within a factor of three of that needed by standard Deluge, with an additional $180$ seconds for the initial bootstrap phase. This cost of the bootstrap can be amortised over a potentially large number of broadcast message transfers. We believe this is an acceptable price to pay for ensuring confidentiality and security of wireless sensor network broadcasts.

There are several directions for future work: we have undertaken further work

[17,18] to extend our scheme to multi-hop networks where transit nodes may be susceptible to physical compromise. We have also extended, prototyped and analysed our scheme [25] for recovery from lost keys. Finally, we are prototyping our scheme for broadcast applications in which the data is dynamic, unlike the network programming application considered in this paper wherein the bulk data is known before-hand.

## References

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

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

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

[4] Wireless Sensor Networks at CSIRO.
URL `http://www.sensornets.csiro.au/fleck1.htm`

[5] A. Perrig, R. Szewczyk, J. Tygar, V. Wen, D. Culler, SPINS: Security Protocols for Sensor Networks, Wireless Networks 8 (5) (2002) 521–534.

[6] P. K. Dutta, J. W. Hui, D. C. Chu, D. E. Culler, Securing the Deluge Network Programming System, in: Proc. Intl. Conf. Inf. Proc. Sensor Networks (IPSN'06), Nashville, TN, 2006.

[7] P. Lanigan, R. Gandhi, P. Narasimhan, Sluice: Secure Dissemination of Code Updates in Sensor Networks, in: Proc. Intl. Conf. Distributed Computing Systems (ICDCS), Lisboa, Portugal, 2006.

[8] J. Deng, R. Han, S. Mishra, Secure Code Distribution in Dynamically Programmable Wireless Sensor Networks, in: Proc. Intl. Conf. Inf. Proc. Sensor Networks (IPSN'06), Nashville, TN, 2006.

[9] S. Hyun, P. Ning, A. Liu, W. Du, Seluge: Secure and DoS-Resistant Code Dissemination in Wireless Sensor Networks, Tech. Rep. TR-2007-21, Dept. of Computer Science, North Carolina State University (Aug 2007).

[10] M. Luk, G. Mezzour, A. Perrig, V. Gligor, MiniSec: A Secure Sensor Network Architecture, in: Proc. Intl. Conf. Inf. Proc. Sensor Networks (IPSN'07), Cambridge, MA, Apr 2007.

[11] Toumaz-Technologies-Ltd., Digital Plaster using Sensium$^{TM}$, online.
URL `http://www.toumaz.com`

[12] Trusted Computing Group.
URL `https://www.trustedcomputinggroup.org`

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

[14] J. W. Hui, D. Culler, The Dynamic Behavior of a Data Dissemination Protocol for Network Programming at Scale, in: ACM SenSys, Baltimore, MD, 2004, pp. 81–94.

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

[16] P. Rogaway, M. Bellare, J. Black, OCB: A Block-Cipher Mode of Operation for Efficient Authenticated Encryption, ACM Trans. Information and System Security 6 (3) (2003) 365–403.

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

[18] H. Tan, D. Ostry, J. Zic, S. Jha, A Confidential and DoS-Resistant Multi-Hop Code Dissemination protocol for Wireless Sensor Networks, in: Proc. ACM Conf. Wireless Network Security (WiSec'09), Zurich, Switzerland, Mar 2009.

[19] Crossbow-Technologies, TelosB motes.
URL
`http://www.xbow.com/Products/productsdetails.aspx?sid=252`

[20] Bouncy-Castle, Cryptographic APIs, online.
URL `http://www.bouncycastle.org`

[21] D. J. Malan, M. Welsh, M. D. Smith, A Public-Key Infrastructure for Key Distribution in TinyOS Based on Elliptic Curve Cryptography, in: IEEE SECON, Santa Clara, CA, 2004, pp. 71–80.

[22] Cyber-Defense-Laboratory, TinyECC: Elliptic Curve Cryptography for Sensor Networks, online (2007).
URL `http://discovery.csc.ncsu.edu/software/TinyECC`

[23] CleverScope, CS328A Mixed-Signal PC Oscilloscope, online.
URL `http://www.cleverscope.com/products/cs328a.php`

[24] D. Culler, D. Estrin, M. Srivastava, Overview of Sensor Networks, IEEE Computers 37 (8) (2004) 41–49.

[25] S. T. Ali, V. Sivaraman, A. Dhamdhere, D. Ostry, Secure Key Loss Recovery for Network Broadcast in Single-Hop Wireless Sensor Networks, Ad Hoc Networks (Elsevier), 8 (6) (2010) 668–679.