Tradeoffs in Configuring Secure Data Dissemination in Sensor Networks: An Empirical Outlook

Patrick E. Lanigan, Priya Narasimhan, Rajeev Gandhi

May 25, 2007
CMU-CyLab-07-006

CyLab
Carnegie Mellon University
Pittsburgh, PA 15213

Tradeoffs in Configuring Secure Data Dissemination in Sensor Networks: An Empirical Outlook

Patrick E. Lanigan, Priya Narasimhan, Rajeev Gandhi

May 25, 2007

Abstract

Network-reprogramming is a valuable service for maintaining sensor networks. Network-reprogramming services need to be not only efficient and reliable, but secure as well. A number of strategies for providing authentication and integrity have been proposed and evaluated, but the tradeoff space has yet to be explored to any significant depth. The recently proposed strategies are mainly distinguished through their structure, granularity and strength of hashing. We propose a configurable, secure data dissemination scheme that uses a different hash-structure to allow for a high degree of flexibility in choosing from a full spectrum of hash granularities and strengths. Using this structure, we are able to experimentally explore the tradeoffs inherent in choosing a hash-granularity under different attack scenarios and densities.

1 Introduction

Network reprogramming has emerged as a necessary service for wireless sensor networks. These services disseminate large data objects, such as code images, efficiently and reliably through large, multi-hop networks of wireless sensor nodes. This allows nodes to be reprogrammed remotely, obviating the need to collect and flash nodes by hand.

Initial network-reprogramming protocols focused on efficiency and reliability, but provided no inherent security mechanisms [4, 8, 18]. The lack of authentication was particularly worrisome, as it allowed any party with network access the ability to disseminate and install arbitrary code images. Due to the epidemic nature of the protocols in question, an attacker could gain complete control of an entire deployed network by compromising just a single node and leveraging it to inject malicious code.

A number of researchers have recently proposed authentication mechanisms that aim to prevent such attacks [1, 2, 10]. These mechanisms typically use a combination of hash functions and digital signatures to provide authenticity and integrity for the disseminated data. They differ mainly in their precise structure (e.g., hash-trees vs. hash-chains), granularity (e.g., individual packets vs. groups of packets called pages), and strength of hashing (e.g., full vs. truncated). These differences imply a number of tradeoffs. For example, a hash-tree [1] allows data packets to be verified in arbitrary order, but incurs more overhead than a simple hash-chain. Likewise, hashing at the granularity of a page incurs less overhead than hashing individual packets, but forces nodes to re-request a larger amount of data when a hash verification fails. Using a truncated hash reduces the amount of cryptographic data that needs to be transmitted, but increases the hash's vulnerability to compromise.

Existing authentication mechanisms are fairly rigid with regard to these tradeoffs. For example, schemes that apply packet-level hashing are usually forced to truncate the hash in order to offer acceptable overhead. Arguments in favor of both packet-level and page-level hashing have been made, but no comparative data has been presented to support either strategy over the other [2, 10]. A sensor-network developer faced with many choices of secure update strategies needs more information to make an informed choice for his/her system.

The concrete contributions of this paper are:

- **A configurable approach.** We propose a *configurable, secure data dissemination scheme that explores security-performance tradeoffs*, by using a hash-list as an alternative to previous hash structures. Our application of the hash-list combines the strengths of hash-chains and hash-trees, and allows a high degree of flexibility in choosing appropriate parameters such as hash size and hashing granularity.

- **A prototype implementation.** We describe a prototype of the configurable approach, and discuss various issues that arose in bringing it from theory to implementation.

- **An experimental evaluation.** We evaluate the effect of the hash granularity under two likely attack scenarios, and explore the resulting tradeoffs. We find that pure page-level and pure-packet level strategies are less efficient than strategies that strike a balance between the two.

The remainder of this paper is organized as follows. We describe existing approaches and motivate our configurable approach in Section 2. We present

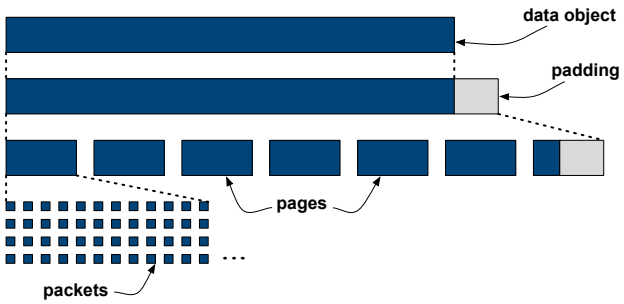

Figure 1: Deluge splits an update into pages, and then packets.

our new approach in Section 3, and a brief qualitative discussion in Section 4. Our implementation is detailed in Section 5. The results of our experimental evaluation are in Section 6. Section 7 has a brief discussion of additional related work, and Section 8 discusses future work. Finally, we summarize our conclusions in Section 9.

2 Motivation

A number of protocols have been proposed that facilitate efficient and reliable network-reprogramming services for sensor networks. Most of the protocols employ an *epidemic* dissemination strategy, whereby nodes propagate received data to their in-range neighbors, which then propagate them further downstream, and so on. The most prominent network-reprogramming protocol, Deluge, is distributed with TinyOS and is enabled by default in many sample TinyOS applications [4, 12].

Prior to dissemination, Deluge segments an data object into fixed-size fragments called *pages*. Each page is further broken up into fixed-size *packets* (see Figure 1). Deluge leverages the page construct to enable *pipelining*, whereby nodes can forward completed pages before they have received all of the pages that form a complete data object. Pipelining takes place under the condition that pages must be transmitted *in-order* (i.e., a node must have received page n before page $n + 1$ can be forwarded).

Existing network-reprogramming protocols generally assume correct operation among all of the nodes in the network. The lack of authentication implies that any party with access to the wireless communication channel can advertise, and initiate the propagation of, arbitrary data objects. While symmetric-key based mechanisms exist for controlling access to the wireless medium [5, 14, 16], physical node compromise remains a serious concern. If

an attacker is able to compromise a single node, that node could be used to disseminate code updates to its neighbors, and so on. The epidemic nature of data-dissemination protocols thus allows local vulnerabilities to have network-wide consequences. To protect against node compromise, the data object itself needs to be verified as arriving unmodified from an authorized source.

Recent research has sought to provide more secure network reprogramming services by augmenting existing data dissemination protocols with authentication and integrity mechanisms. These schemes typically leverage digital signatures and cryptographic hashes, but apply those primitives in different ways. The recently proposed schemes are mainly distinguished through their *structure*, *granularity* and *strength* of hashing.

Hash-structure The hash-structure describes the way multiple hashes are combined in order to ensure the integrity of a single, large data object. Common hash structures are hash-trees [15], one-way hash-chains [9], and hash-lists.

Hash-granularity The hash-granularity describes what amount of object data is covered by a single hash value and what level of data verification is possible. If a hash verification fails, the integrity of the data covered by that hash is uncertain and cannot be trusted. Current schemes adopt either a page-level or a packet-level approach. Page-level hashing provides coarse granularity by employing a single hash for each page. Packet-level hashing provides fine granularity by hashing each data packet individually.

Hash-strength The hash-strength describes the size of the hash value used for verification, relative to the size of the value produced by the hash-function[1]. Some schemes use full-strength hashes, where the output of the hash-function is used as is. Other schemes truncate the hash value, using just the first few bytes of output. For a given hash-function, its truncated output will generally provide weaker security guarantees than a full-strength hash.

Sections 2.1 and 2.2 describe a few different schemes with regard to these characteristics. Table 1 provides a summary of those differences.

[1]Note that our definition is not meant to imply anything regarding the cryptographic security of the hash-function itself. For example, both a 128-bit MD5 digest and a 160-bit SHA-1 digest would be categorized as "full-strength" hashes, regardless of their respective cryptographic weaknesses. The criteria for choosing a "good" hash-function are beyond the scope of this paper.

	Structure	*Granularity*	*Strength*
Sluice	chain	page	full
SecureDeluge	chain	packet	truncated
Deng-tree	tree	packet	truncated
Deng-hybrid	tree+chain	packet	truncated

Table 1: Summary of three different secure dissemination schemes.

2.1 A Page-Level Approach

Lanigan et. al. proposed Sluice, one of the first authentication mechanisms designed specifically for network reprogramming protocols like Deluge [10]. Since compatibility was one of the desired goals, Sluice leverages existing characteristics of Deluge (e.g., pages and pipelining) in its application of cryptographic primitives.

The basic idea of Sluice is to compute a one-way hash-chain over the pages of an update, by hashing each page and then including that hash in the previous page. The first page of the update is digitally signed, which creates a commitment to the entire chain. Figure 2 shows the sequence of n pages that results with Sluice, where each page's hash is computed and appended to the previous page's payload, e.g., the hash h_{n-1} of page p_{n-1} is computed and concatenated with the payload of the previous page, thereby forming page p_{n-2}.

The head of the hash-chain, h_1, is included in the first page, p_0, which has been digitally signed. The digital signature serves to authenticate the source of the data, and to verify the integrity of the payload and the hash contained in that page. Once the head of the hash-chain has been verified, the remaining pages can be verified through the one-way hash-chain properties, i.e., hash h_{i+1} contained in page p_i serves to verify both the payload and the hash h_{i+2} contained in page p_{i+1}.

One potential drawback of a chain-based scheme is that each hash in the chain must be verified in-order. Deluge already enforces the ordered transmission of pages, so the hash-chain allows each page of the data object to be verified as soon as it is received. Embedding hashes in pages instead of in packets allows Sluice to avoid truncating hashes in order to accommodate relatively small packet sizes. Using only a single hash per page also minimizes the number of hashes needed for each data object, reducing the amount of data transmitted and the computations required at the sensor nodes to verify the hashes.

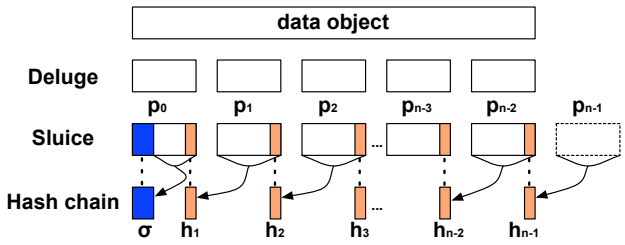

Figure 2: Sluice's hash-chain construction.

The main drawback of Sluice's page-level approach is that nodes are forced to re-request a large amount of data when a hash verification fails. Since a node has no way of knowing which packet(s) corrupted the page, it would be forced to re-request the entire page. This could potentially give an attacker an asymmetric advantage when launching a DoS-attack, wherein the attacker would only need to send a few malicious packets in order to induce a large number of retransmissions.

2.2 Packet-Level Approaches

The primary advantage of a packet-level approach over a page-level approach is that a single bad packet will not trigger the retransmission of an entire page. Packet-level approaches have been proposed by both Dutta et. al. [2] and Deng et. al. [1].

In SecureDeluge [2], Dutta et. al. propose using a chain structure similar to the one used by Sluice, except that the chaining is done over packets instead of pages. In order to leave sufficient room for data in each page, the underlying packet size is increased and the hashes are truncated. Unlike the arrival of pages, there are no guarantees on the order of packet arrival. SecureDeluge mitigates this problem by optimistically buffering packets that arrive out-of-order. Moreover, the increased packet size implies higher propagation delays and more packet losses.

Deng et. al. take a different approach to dealing with out-of-order packets [1]. Their tree-based approach uses a multi-level tree made up of index packets, which contain hashes, and data packets, which do not. The data object is split up into data packets that make up the leaves of the tree. Each data packet is hashed, and the hash value is placed in an index packet at the next higher level in the tree. The index packets themselves are hashed, and these values are placed in higher-level index packets. The process continues, until there is a single index packet at the root. The root

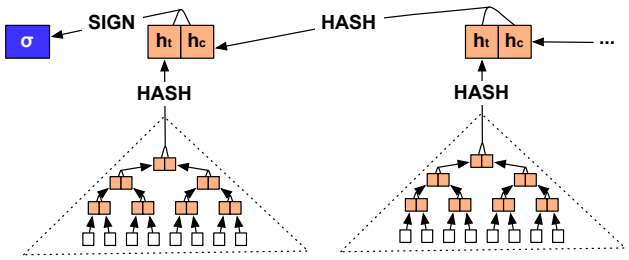

Figure 3: Deng et al., hybrid construction.

packet is then digitally signed, providing a commitment to the entire tree.

Like the chain-based approach, this scheme allows for packet-level verification. Moreover, it allows a higher number of packets to be verified if they are received out-of-order. Data packets, in particular, may be received in any order once all of the index packets have been received and verified. The main drawback to this approach is that it does not consider the fact that packets are typically grouped into pages. Constructing a single hash-tree for large data objects would require a significant amount of memory to store the index packets while the dissemination is in progress. Multiple hash-trees could be constructed for each individual page, but that would require expensive per-page digital signatures.

Deng et. al. also propose a hybrid approach that combines hash-trees and hash-chains (see Figure 3). This approach considers that a large update is broken up into pages, as is the case with Deluge. First, a hash-tree is created for each page, over all of the packets in that page. Then, a hash-chain is constructed over the root packets of each page. A single digital signature provides a commitment for the hash-chain, and the chain provides commitments for the root of each tree. Like the tree-based approach, the hybrid approach allows for out-of-order, packet-level verification. It improves on the tree-based approach by requiring only a single digital signature for each code update.

2.3 Toward a Configurable Approach

There are a number of security-performance tradeoffs to consider when choosing an appropriate hash-granularity. For example, packet-level hashing imposes high overhead in the steady (i.e., non-attack) state, but can be more efficient under attack. On the other hand, page-level hashing imposes lower overhead in the steady state, but can be less efficient under attack. High steady-state overhead is incurred by all nodes in the network, regardless

of whether the nodes are actively receiving corrupt data. However, attack overhead (e.g., re-requests for corrupted data) is limited to nodes within the broadcast range of the attacker, because secure dissemination schemes inherently limit the propagation of corrupt data.

Other important considerations arise when choosing an appropriate hash-strength. For many deployments (e.g., short-term environmental monitoring), a truncated hash could provide adequate security. More sensitive applications (e.g., critical infrastructure monitoring) might require larger hashes. The hash-strength should ideally be chosen to reflect the sensitivity of the application. However, packet-based approaches are limited in this sense. For example, the standard TinyOS packet size allows for 23 bytes of payload, while a full SHA-1 hash is 20 bytes. In this case, embedding a full SHA-1 hash in a standard TinyOS packet would only leave 3 bytes for object data.

When we first set out to explore these tradeoffs, we realized a few things. First, current schemes are fairly rigid with regard to the hash-granularity and hash-strength. Page-level solutions like Sluice allow the hash-strength to be changed fairly easily, because hash values of any size can be embedded in a page, regardless of the packet size. However, Sluice is limited in that it only supports a very coarse granularity. Packet-level solutions, such as [1] and [2] could be easily extended to support a more coarse granularity by hashing an arbitrary number of packets, but finding a place to put these hashes is a bit more complicated. Moreover, if a stronger hash is needed, the sensor-network developer is seriously constrained by limited packet size.

To address these limitations, we decided to use a different hash-structure that supports a wide range of hash-granularities and hash-strengths, regardless of the underlying packet-size (within reason, of course). This configurable approach should allow developers to balance competing interests by tuning the granularity and strength for their particular deployment and application. Furthermore, the degree of out-of-order verification should be directly related to the hash-granularity.

3 Approach

For this work, we adopt a system model similar to that proposed in [10]. Specifically, we assume untrusted sensor-nodes that might behave arbitrarily or leak cryptographic material. Malicious or compromised nodes may be located anywhere in the network. The wireless medium is insecure, and can be used by an adversary to inject, capture, or modify packets. This work is not concerned with the confidentiality of data objects, or with recovering

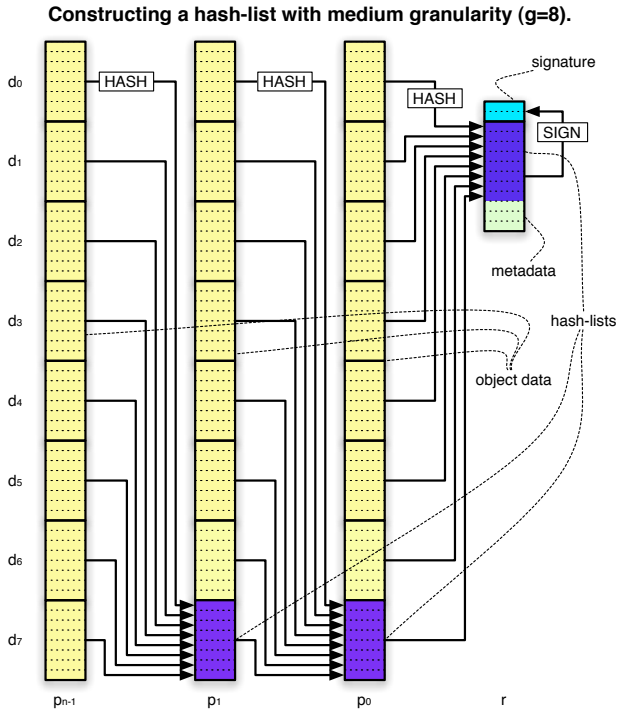

Figure 4: Utilizing hash-lists to provide authenticity and integrity.

nodes that have already been compromised.

Our approach takes cues from the methods discussed in Section 2, but uses a hash-list as the underlying structure. We describe the signing and verification procedures of our approach below, and provide pseudocode in Figures 6 and 7.

3.1 Data Signing

An appropriate hash-granularity, g, corresponding to the number of packets covered by each hash, must be chosen before constructing the hash-list. Then, a hash-size s_h must be chosen according to the security requirements of the particular application. Like Sluice, our solution embeds hashes at the page level, so $s_l = g * s_h$ bytes must be reserved in each page to store the hash-list. Unlike Sluice, there can be more than one hash in each page. The two parameters, g and s_h, directly determine the amount of cryptographic data that must be transmitted with each page. A coarse granularity will have fewer hashes per page, and more packets will be covered by each hash.

Conversely, as the granularity becomes finer, the number of hashes embedded in each page will increase, while fewer packets will be covered by each hash (see Figure 5). Note that the page size itself does not increase, but the number of pages likely will, because there is less room for data in each page (note that this increase is not represented in the figures).

Hashing then begins at the last page of the data object, and proceeds forward (see Figure 4.) Every g packets in page p_i make up a segment, d_j, that is hashed. The hashes are concatenated together to form a hash-list l_i. The hash-list l_i is placed in the reserved space in the previous page, p_{i-1}. The hash-list l_0 produced from the first page is concatenated with optional meta-data m, and digitally signed to produce a signature σ. These values together form the root block, $r = (\sigma|h_0|m)$.

3.2 Data Verification

The root block contains all of the information needed to begin verifying the data object. Therefore, it must be disseminated prior to the data pages. This can be done in a number of ways. The simplest way to integrate this with an existing protocol like Deluge would be to create a special "signature" page, p_s, containing only the root block, and prepend this page to the data object. This solution is slightly wasteful, as the signature page would contain a large amount of padding (recall that pages are all a fixed size). Once the root block has been sent, the Deluge dissemination could commence.

When a node receives the root block, it verifies the authenticity and integrity of the meta-data and the initial hash-list using the digital signature. The meta-data can be used to provide information about the data object, such as the number of pages it contains or a hash of the entire object[2]. Each individual hash in hash-list l_i can then be used to verify the corresponding packet(s) in the next page, including the following hash-list l_{i+1}.

4 Discussion

Embedding the hash-list within pages allows network architects a great deal of flexibility in choosing an appropriate strength and granularity. Using a smaller hash-strength can allow the system designer to offset the increase

[2]This prevents a potential attack where an adversary advertises an already signed data object with a new version number, but a reduced number of pages. The signature would verify, but the nodes would think that the data object is complete before it really is. This could potentially cause the nodes to write an incomplete code image into program memory, and begin executing the corrupted code, causing them to crash.

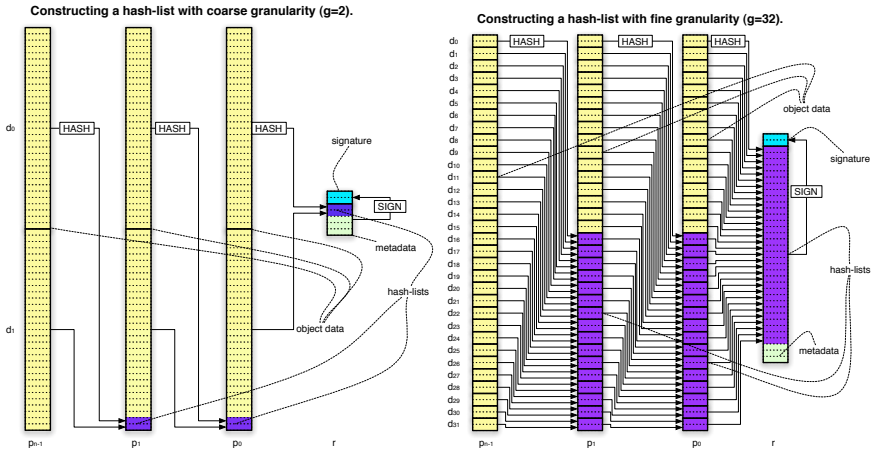

Figure 5: Utilizing hash-lists to provide authenticity and integrity with different hash-granularities.

in overhead caused by a fine granularity. For example, a granularity of $g = 4$ and a strength of $s_h = 20$ will produce the same amount of per-page overhead as a granularity of $g = 8$ and $s_h = 10$. In fact, existing methods like Sluice and SecureDeluge can both be described by this model. When $g = 1$ and $s_h = 20$, our method is essentially equivalent to Sluice. With $g = 48$ and $s_h = 4$, our method looks very similar to SecureDeluge; it really only differs in its placement of the hash data.

The hash-list scheme allows out-of-order verification without the additional overhead required to send branches in a tree-based scheme like that proposed in [1]. Once a page has been completed, every segment in the following page can be verified as soon as it received because the correct hashes were already sent and verified.

5 Prototype Implementation

We implemented the hash-list approach in TinyOS as a modification to Deluge. The structure of the existing Deluge codebase, combined with a desire on our part to keep the modifications as minimally invasive as possible, forced a number of compromises that would be less than ideal in a production-quality (rather than a prototype, as is currently the case) implementation.

The first issue was determining how to distribute the root blocks. The most straightforward approach was to embed the root block in first page,

```
partition data object into n pages p_0, p_1, ..., p_{n-1}
for all (i ← n − 1 to i = 0) do
    partition p_i into g segments d_0, d_1, ..., d_{g-1}
    for all (j ← 0 to j = g − 1)) do
        h_j ← HASH(d_j)
    end for
    l_i ← [h_0, h_1, ..., h_{g-1}]
    if (i > 0) then
        p_{i-1} ← (p_{i-1}|l_i)
    end if
end for
σ ← SIGN(l_0|n)
r ← (σ|l_0|n)
```

Figure 6: Data signing procedure.

before any of the object data. Certain Deluge modules expect a 256-byte block of CRC values (a 2-byte CRC for each of a maximum of 128 pages) at the beginning of the data object; thus, we placed the root block immediately following the CRC block in the first page. We choose to decouple the root block from the rest of the object data by padding the remaining space in the first page with zero bytes, effectively creating a dedicated *root page* followed by subsequent *data pages*. This abstraction led to a simplification of data structures in our experimental prototype, and allows future flexibility for determining the most efficient way to distribute the root block (see Section 7).

This approach did lead to some unforeseen problems, mainly related to the way Deluge handles the CRCs. Recall that in our construction, the digital signature is meant to provide a commitment to the hash-list. We intentionally did not extend the signature to the padding bytes. Nodes know *a priori* how many padding bytes there are, and therefore will not use the padding for anything. Suppose though, that a malicious node sends a corrupted packet that contains a part of the padding. Nominally, it does not matter to a node if the bytes are zero or non-zero; they should simply be ignored. As long as the root hash-list is not corrupted, the digital signature will pass verification. However, some lower-level Deluge modules still check the page's corresponding CRC, and reject it. This can cause a large number of retransmissions (and hence signature re-computations) for the root page even though the signatures were perfectly valid. We fixed this problem by having each node explicitly zero-out the padding bytes after receiving the

root page, and before passing the page on to the lower-level modules. Future versions might avoid requesting packets containing padding bytes altogether.

A more complicated issue arose with the CRC block itself. This problem was similar to that experienced with the padding; the CRC block was not included in the digital signature, so if it was corrupted, the root page would be rejected by lower-level modules although the root block passed the signature verification. Our solution was to completely disable the CRC checks. It would probably be better to include the CRC block in the digital signature, but this turns out to be a "catch-22" situation [3]. The CRC value for the root page is computed over all of the data embedded in that page (including the digital signature). The digital signature would need to be computed over all of the data in the CRC block, including the first CRC value. Yet we cannot compute one before the other is known. A more elegant solution would be to place the signature in the first few bytes of the root page, and not include it in the computation of the CRC. However, this would have required extensive changes to the Deluge code-base.

6 Evaluation

In order to evaluate the performance of the different hash schemes in an attack scenario, we implemented two attacker models. These models represent an attacker that aims to waste network resources by causing nodes to re-request corrupted data. In both models, a "malicious" node randomly chooses a single target packet from each page to corrupt. The two models differ in how they behave with respect to the underlying Deluge protocol.

In the *forwarding* model (`fwd`), malicious nodes follow the correct Deluge protocol. These nodes respect all of the rules regarding back-offs and message suppression, and only respond to those requests addressed to the nodes themselves. When a target packet is requested from a malicious node, it responds with corrupted data. Otherwise, malicious nodes forward correct data.

In the *non-forwarding* model (`nofwd`), malicious nodes follow the Deluge protocol with respect to receiving data, but *not* with respect to sending data. They respond to *every* overheard request for their target packets, not just those addressed to them directly. They do not forward correct data, nor do they respect back-off or message suppression rules.

g	*# of pages*
0 (no verification)	3
1	4
4	4
8	4
16	4
24	4
48	5

Table 2: Number of pages required to disseminate `BlinkM` for various verification-granularities.

6.1 Experimental Results

Our experimental results were obtained using TOSSIM [11], a simulator for TinyOS. We tested both the `fwd` and the `nofwd` attacker models. All of our experiments were conducted using 100 simulated nodes arranged in a 10x10 node topology, attempting to disseminate the `BlinkM` sample application included with TinyOS. This application simply toggles the sensor node's built in LED. The number of pages required to disseminat `BlinkM` directly depends on the values of g and s_h (see Table 2). For each experimental run, we varied the hash-granularity g, and the probability, p_{mal}, that an individual node would be malicious. We fixed $s_h = 4$ for all experiments.

We had originally planned to run each experiment for 1800 simulated seconds. However, in many cases this was simply not enough time for the data object to be completely received by all of the nodes. We then increased the simulation time to 3000 seconds, which was enough time for all nodes to completely receive the data object in all cases. However, due to the extremely long running nature of TOSSIM, we were not able to complete the `nofwd` data set for 3000-second test cases.

Dissemination progress One important performance characteristic is the progress of the data-dissemination protocol. Figure 8 shows the percentage of nodes having completely received the data object after 1800 seconds. For the `fwd` model, nodes are, for the most part, able to make progress in the presence of malicious packets. The only exception is for the page-level ($g = 1$) granularity, where some nodes are not able to complete in time. These nodes are most likely being kept busy re-requesting large amounts of data due to the coarse verification-granularity.

Nodes in the `nofwd` model fare much worse. A few nodes in each run are unable to complete even with as little as only 10% of them being malicious. The progress drops off rapidly as the network becomes saturated with malicious nodes, suggesting that the absence of "good" data has a much greater effect on progress than the presence of "bad" data.

Data efficiency An interesting metric to quantify the efficiency of data dissemination is the amount of redundancy experienced through the course of disseminating a data object. We define redundant data as data that is dropped by node because it has received and verified a duplicate packet. For example, a node might request and receive a given packet, and then later overhear it yet again because a different node requested the same packet as well.

Figure 9 shows the amount of redundant data for the entire network, as the percentage of dropped data packets out of the total number of data packets sent. For both the `fwd` and `nofwd` models, there appears to be a large amount of redundant data. To some extent, this is due to the fact that Deluge only suppresses redundant control packets and not redundant data packets. Based on previous experimental results, we would expect around 60% redundancy [4]. Our observed increase is likely due to the compromised nodes forcing re-requests. It it interesting to note that the `nofwd` case exhibits increasing redundancy as the saturation of malicious nodes grows, while the `fwd` case remains relatively constant.

Overall efficiency Our main goal was to determine whether increasing the hash-granularity resulted in desirable tradeoffs, as compared to the total amount of data transferred. To a node trying to complete a data object, it does not matter whether extra data is transmitted over the radio because it has been re-requested or because it is cryptographic data needed to verify the data object. It all takes up energy, all the same.

In Figure 10, we see that page-level hashing ($g = 1$) results in a sharp increase in data traffic as opposed to finer granularity strategies ($g > 1$). The massive amount of re-requested data has overwhelmed any savings gained by only using a single hash per page.

However, pure packet-level hashing ($g = 48$) sends significantly more data packets than any of the medium hash-granularity strategies. The packet-level scheme adds a large amount of cryptographic data in order to guard against unnecessary retransmissions. In fact, the packet-level scheme needs an entire extra page to contain the data object (see Table 2). However,

this increase is not offset by the increased resolution with which nodes can re-request data. For all of the medium hash-granularity cases, the number of data packets transmitted is comparable.

Figure 11 shows that in addition to requiring fewer messages to complete a data object, the medium-granularity cases require less time as well.

Malicious traffic In the `fwd` model, the amount of malicious traffic in the network was actually very low with respect to the amount of good data traffic (see Figures 12 and 13). This shows that even a small amount of malicious traffic can have a relatively large effect on the operation of a the network. Also note that the ratio of malicious traffic to data traffic does not grow as quickly as the raw number of data packets. A doubling of malicious traffic results in more than a doubling of data traffic, causing a less drastic increase in the percentage of malicious versus data traffic. This shows the asymmetric effect of the attacks. In this case, a higher percentage of malicious traffic is a good sign, because it shows that fewer good packets were needed to complete the data object.

7 Related work

One commonality between the approaches described in Sections 2.1 and 2.2 is the need to distribute a secure commitment to the hash structure. While these approaches utilize a digital signature, other researchers have looked for ways to leverage less expensive cryptographic primitives [6, 7]. Both Kim et al. and Kronti et al. use page-level chaining mechanisms identical to Sluice, but verify the head of the hash-chain in different ways.

Kim et al. developed Castor, which avoids digital signatures through the use of MACs and one-way keychains with delayed key disclosure [6]. This results in end-to-end dissemination latency that is significantly better than Sluice, and only marginally worse than Deluge. However, delayed key disclosure requires the network to be loosely time synchronized. In networks where this is an acceptable requirement, Castor could be combined with our configurable approach to provide more efficient verification of the root block.

Kronti et al. leverage an r-time signature scheme to verify the head of the hash-chain. This signature scheme uses symmetric cryptographic primitives (specifically, a cryptographic hash function) to provide authentication. Their method allows tradeoffs to be made between security level, signature size, and memory overhead. Combining their signature scheme with the hash-list approach proposed here would provide configurability for both the

authentication of the root block, and the verification of the data pages.

All of the work discussed so far has focused on disseminating data objects in the presence of compromised nodes. Other researchers have proposed methods for detecting and repairing compromised nodes. SCUBA, developed by Seshadri et al., is one such method [17]. It allows a sensor-network administrator to remotely verify that a data object was indeed installed on a sensor-node, even if that node has been compromised. Remote-verification techniques such as SCUBA, and secure dissemination techniques such as ours are complimentary; together, they can provide robust, secure network reprogramming services.

8 Future work

Our approach provides flexibility and is very efficient for verifying the integrity of the data pages, but there are likely better ways to distribute the root block. It would be interesting to explore the possibility of adapting the techniques described in Section 7 to distributing the root block. Also, the dissemination problem is different for large (tens of kilobytes) vs. small (a few bytes) data objects. Another alternative might be to use a dissemination protocol that is specifically designed for smaller data objects, such as Trickle [13], to quickly spread the root block through the network. It might also be worthwhile to explore the use of different data structures in order to provide a finer granularity for the root block. However, the root block is typically only a few packets so the improvement would probably be limited.

In the future, additional work is needed to evaluate more sophisticated attacker models. Our models are naive in that they randomly pick their target packets. Smarter attacks could target specific packets (e.g., packets containing parts of the signature) to further disrupt the protocol. It would also be interesting to experiment with more varied topologies.

Ultimately, we would like to provide a theoretical model that encompasses a variety of network and threat characteristics and allows sensor-network developers to easily determine the most efficient dissemination parameters for their specific application.

9 Conclusion

In this paper, we presented a method for verifying the integrity and authenticity of data disseminated with network reprogramming protocols like Deluge. This method is more flexible than previously presented approaches be-

cause it allows the user to tune the hash-granularity and hash-strength, without altering basic network characteristics such as packet size. Our method allows sensor-network developers to configure the security-performance tradeoffs to suit their particular needs. After all, enterprise level security protocols often support a number of options for choosing cryptographic functions and key sizes. This same flexibility should be extended to sensor networks as well.

We further presented experimental data to suggest that neither pure packet-level, nor pure page-level approaches are optimal. Page-level hashing appears to only be efficient when there are very few, if any, malicious nodes. Packet-level hashing rarely appears to be the most efficient strategy. However, it is possible to reach an efficient balance between the two. This result gives sensor-network developers an incentive to look more closely at performance-security tradeoffs, and to take advantage of the configurability provided by our approach.

References

[1] J. Deng, R. Han, and S. Mishra. Secure code distribution in dynamically programmable wireless sensor networks. In *ACM/IEEE Conference on Information Processing in Sensor Networks*, pages 292–300, Nashville, TN, April 2006.

[2] P. K. Dutta, J. W. Hui, D. C. Chu, and D. E. Culler. Securing the Deluge network programming system. In *ACM/IEEE Conference on Information Processing in Sensor Networks*, Nashville, TN, April 2006.

[3] J. Heller. *Catch-22*. Simon & Schuster, New York, New York, 1961.

[4] J. W. Hui and D. E. Culler. The dynamic behavior of a data dissemination protocol for network programming at scale. In *ACM International Conference on Embedded Networked Sensor Systems*, pages 81–94, Baltimore, MD, November 2004.

[5] C. Karlof, N. Sastry, and D. Wagner. TinySec: A link layer security architecture for wireless sensor networks. In *ACM International Conference on Embedded Networked Sensor Systems*, pages 162–175, Baltimore, MD, November 2004.

[6] D. H. Kim, R. Gandhi, and P. Narasimhan. Exploring symmetric cryptography for secure network reprogramming. In *International Workshop on Wireless Ad-hoc and Sensor Networks*, New York, NY, June 2007.

[7] I. Krontiris and T. Dimitriou. Authenticated in-network programming for wireless sensor networks. In *International Conference on Ad-Hoc Networks and Wireless*, Ottawa, Canada, August 2006.

[8] S. S. Kulkarni and L. Wang. MNP: Multihop network programming for sensor networks. In *IEEE International Conference on Distributed Computing Systems*, pages 7–16, Columbus, OH, June 2005.

[9] L. Lamport. Password authentication with insecure communication. *Communications of the ACM*, 24(11):770–772, November 1981.

[10] P. E. Lanigan, R. Gandhi, and P. Narasimhan. Sluice: Secure dissemination of code updates in sensor networks. In *IEEE International Conference on Distributed Computing Systems*, Lisbon, Portugal, July 2006.

[11] P. Levis, N. Lee, M. Welsh, and D. E. Culler. TOSSIM: Accurate and scalable simulation of entire tinyos applications. In *ACM International Conference on Embedded Networked Sensor Systems*, pages 126–137, Los Angeles, CA, November 2003.

[12] P. Levis, S. Madden, D. Gay, J. Polastre, R. Szewczyk, A. Woo, E. Brewer, and D. E. Culler. The emergence of networking abstractions and techniques in TinyOS. In *USENIX Symposium on Networked Systems Design and Implementation*, pages 1–14, San Francisco, CA, March 2004.

[13] P. Levis, N. Patel, D. E. Culler, and S. Shenker. Trickle: A self-regulating algorithm for code propagation and maintenance in wireless sensor networks. In *USENIX Symposium on Networked Systems Design and Implementation*, pages 15–28, San Francisco, CA, March 2004.

[14] D. Liu, P. Ning, and R. Li. Establishing pairwise keys in distributed sensor networks. *ACM Transactions on Information and System Security*, 8(1):41–77, February 2005.

[15] R. Merkle. Protocols for public key cryptosystems. In *IEEE Symposium on Security and Privacy*, Oakland, CA, April 1980.

[16] A. Perrig, R. Szewczyk, J. D. Tygar, V. Wen, and D. E. Culler. SPINS: Security protocols for sensor networks. *Wireless Networks*, 8(5):521–534, 2002.

[17] A. Seshadri, M. Luk, A. Perrig, L. van Doorn, and P. Khosla. SCUBA: Secure code update by attestation in sensor networks. In *ACM Workshop on Wirelesss Security*, pages 85–94, Los Angeles, CA, October 2006.

[18] T. Stathopoulos, J. Heidemann, and D. Estrin. A remote code update mechanism for wireless sensor networks. Technical Report CENS-TR-30, University of California, Los Angeles, Center for Embedded Networked Computing, November 2003.

```
b ← VerifyRoot
while b ≠ true do
    b ← VerifyRoot
end while

i, j ← 0
l ← l_0
for all j ← 0 to j = g − 1 do
    need[j] ← true
end for

while (i < n) do
    while (need[0] ∨ need[1] ∨ ... ∨ need[g − 1] =
true) do
        receive d'_j
        h ← Hash(d'_j)
        if (h = l [j]) ∧ (need[j]) then
            d_j ← d'_j
            needed[j] ← false
        end if
    end while
    p_i ← (d_0|d_1| ... |d_{g−1})
    ACCEPT(p_i)
end while

---

procedure VerifyRoot
    receive r
    (σ|l_0|n) ← r
    return Verify(σ, l_0|n)
end procedure

---

procedure Accept(p)
    mark p as complete
    save p to flash
    (p|l) ← p
    i ← i + 1
    for all j ← 0 to j = g − 1 do
        need[j] ← true
    end for
end procedure
```

Figure 7: Data verification procedure.

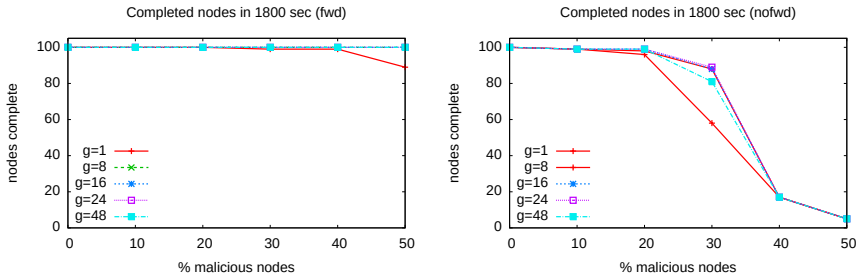

Figure 8: Showing the dissemination progress, as the number of nodes completely receiving the disseminated object.

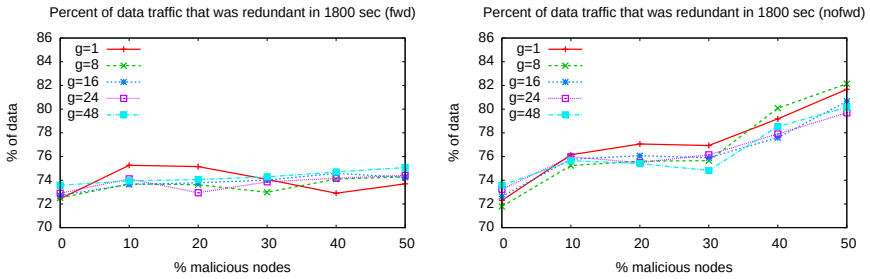

Figure 9: Fraction of total data traffic that was redundant for 1800 seconds of simulated time.

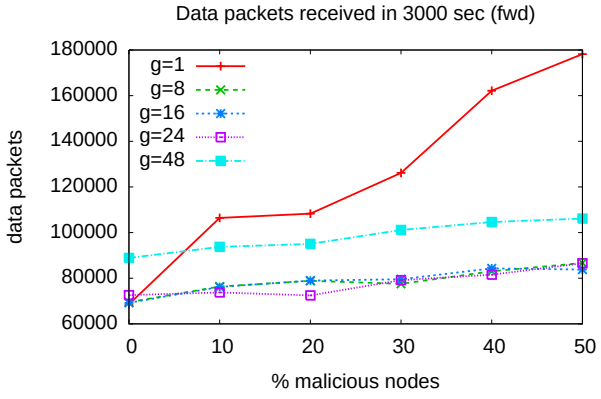

Figure 10: The number of data packets received by of the all nodes in the network during dissemination.

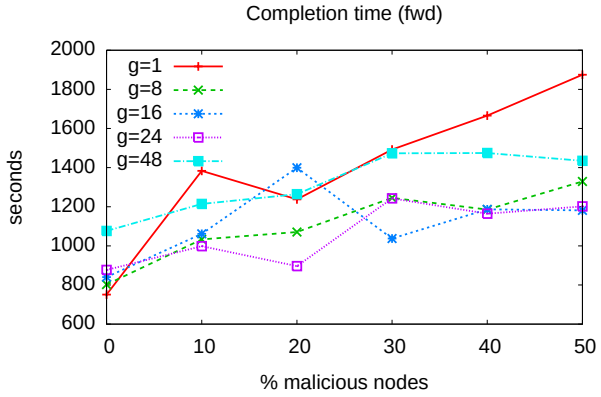

Figure 11: The dissemination latency for the entire network.

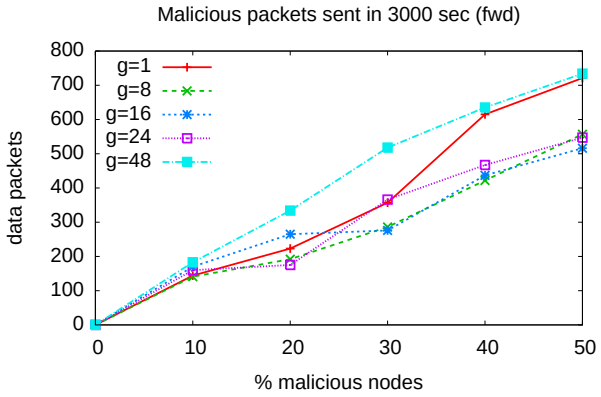

Figure 12: The number of malicious packets sent.

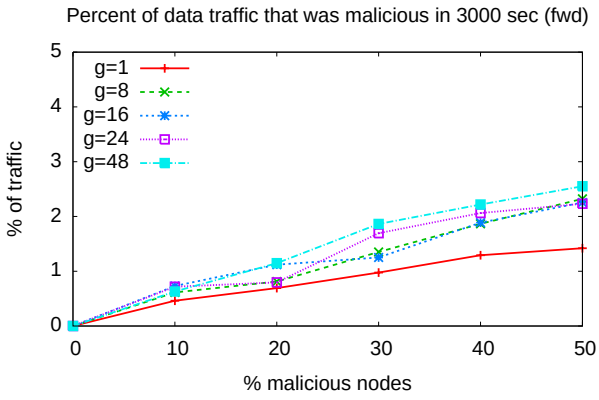

Figure 13: The percentage of packets that were malicious.