Hindawi Publishing Corporation
EURASIP Journal on Wireless Communications and Networking
Volume 2011, Article ID 685219, 22 pages
doi:10.1155/2011/685219

Research Article

Secure Rateless Deluge: Pollution-Resistant Reprogramming and Data Dissemination for Wireless Sensor Networks

Yee Wei Law,[1] Yu Zhang,[2] Jiong Jin,[1] Marimuthu Palaniswami,[1] and Paul Havinga[3]

[1] *Department of Electrical and Electronic Engineering, The University of Melbourne, Parkville, VIC 3010, Australia*
[2] *School of Computer Science, Northwestern Polytechnical University, Xi'an, Shaanxi 710072, China*
[3] *Pervasive Systems Group, Faculty of EEMCS, University of Twente, P.O. Box 217, 7500 AE Enschede, The Netherlands*

Correspondence should be addressed to Yee Wei Law, ywlaw@unimelb.edu.au

Received 24 February 2010; Revised 22 June 2010; Accepted 19 July 2010

Academic Editor: Damien Sauveron

A network reprogramming protocol is made for updating the firmware of a wireless sensor network (WSN) *in situ*. For security reasons, every firmware update must be authenticated to prevent an attacker from installing its code in the network. While existing schemes can provide authentication services, they are insufficient for a new generation of network coding-based reprogramming protocols like Rateless Deluge. We propose Secure Rateless Deluge or Sreluge, a secure version of Rateless Deluge that is resistant to pollution attacks (denial-of-service attacks aimed at polluting encoded packets). Sreluge employs a neighbor classification system and a time series forecasting technique to isolate polluters, and a combinatorial technique to decode data packets in the presence of polluters before the isolation is complete. For detecting polluters, Sreluge has zero false negative rate and a negligible false positive rate. TOSSIM simulations and experimental results show that Sreluge is practical.

1. Introduction

There are occasions when new applications need to be installed, or when existing applications need to be modified or patched in a WSN. Given the difficulty of physically redeploying sensors, this requires the new firmware to be securely and remotely transmitted to all the nodes in the network.

In the core of a secure network, reprogramming protocol (also known as secure code update or secure code distribution in the literature) is a data dissemination protocol that distributes the new firmware to all the nodes in the network in an energy-efficient manner. In the beginning, the base station broadcasts a command (CMD) globally to announce the availability of the new firmware (which can be generalized as any data). The basic epidemic-like workflow established by SPIN [1] then takes place: (i) a node with the new firmware broadcasts an advertisement (ADV) locally; (ii) the receivers of the advertisement compare the advertised version number with their existing version number, and send requests (REQ) for the new firmware if their version number is lower; (iii) the advertiser starts sending data to the requesters; (iv) the receivers of the data in turn become the advertisers. Starting from Deluge [2] (the classic benchmark protocol in the literature), the firmware is disseminated page by page. After successfully collecting a page, the page is written to a nonvolatile memory. After successfully collecting all the pages, a node's next action depends on the semantics of CMD. For example, if the CMD is "disseminate," the node's job is done; if the CMD is "disseminate and reboot," the node then reboots using the new firmware (these are two existing commands in TinyOS 2.x). The problem of securing the protocol boils down to authenticating the CMD, ADV, REQ, and data packets, and addressing the threat of denial-of-service (DoS) attacks in every step of the protocol. In this work, we are primarily concerned with the authentication of data packets and the mitigation of DoS attacks during the process.

In the recent years, the technique of network coding [3] has been applied to improve the dissemination of data packets, resulting in such protocols as AdapCode [4] and Rateless Deluge [5] (more precisely, these protocols use random linear codes). Like Deluge, these protocols still divide the new firmware into pages and divide a page into

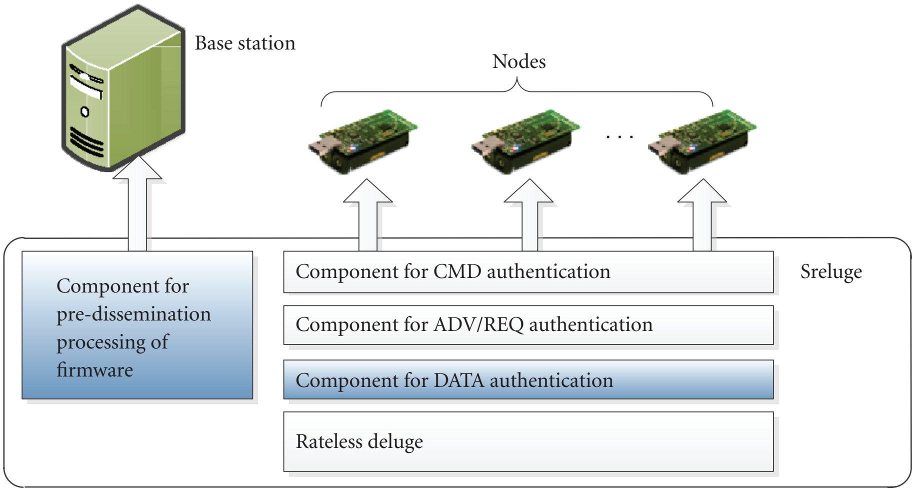

Base station
Nodes
· · ·
Component for pre-dissemination processing of firmware
Component for CMD authentication
Component for ADV/REQ authentication
Component for DATA authentication
Rateless deluge
Sreluge

Figure 1: Components of Sreluge: shaded components are our contribution; unshaded components are drop-in replacements from the literature. For CMD authentication, Sreluge can utilize either Elliptic Curve Cryptography with the proper DoS countermeasure or a multiple-time signature scheme (see Section 2.1). For ADV/REQ authentication, Sreluge utilizes a pairwise key predistribution scheme—either deterministic or probabilistic—to achieve authenticated local broadcast and authenticated local unicast (see Section 5.2).

packets. The advantage of network coding-based protocols is that they are resilient to packet loss. For example, at a packet loss rate of 5%, Rateless Deluge transmits three times less data packets than Deluge does [5]. Resilience to packet loss is a much desirable feature in disruptive environments (e.g., marine environment) where loss of connectivity occurs frequently. There is hence much incentive in using protocols like Rateless Deluge for reprogramming in these environments. For reprogramming protocols that are not based on network coding, several security schemes have been proposed. The technique common to these proposals is the use of a hash function that binds a later packet to an earlier packet which ultimately leads to a digital signature signed by the base station, to facilitate authentication. This technique works on uncoded data packets, while in network coding-based reprogramming protocols, data packets are transmitted as random linear combinations of the original data packets. In other words, although this existing technique can be used to authenticate *uncoded* data packets, rogue nodes can pollute *encoded* data packets, thereby executing the kind of DoS attack known as *pollution attack* in the literature [6].

Our contribution is Sreluge, a security-enhanced version of Rateless Deluge that is resistant to pollution attacks. Figure 1 highlights our specific contribution in terms of the components that make up Sreluge. Sreluge employs a neighbor classification system and a time series forecasting technique to isolate attackers, and a combinatorial technique to decode data packets in the presence of attackers before the isolation is complete. TOSSIM simulations show that when 20% of the nodes in a 27-degree, 100-node network are polluters, using Sreluge, dissemination on average takes roughly twice as long to finish as when there is no attack; a node transmits twice as many REQ and data packets, and performs 50% more decoding per page. Experimental results using Crossbow IRIS motes show that the overhead of Sreluge in an 11-node network is insubstantial. Sreluge is equally useful in cases where nonmalicious but faulty nodes "inadvertently" feed corrupted data packets to their neighbors.

This paper is organized as follows. Section 2 discusses related work. Section 3 details the attacker model on which this proposal is based. Section 4 describes the central problem in a more formal setting. Sections 5 and 6 describe the algorithms of Sreluge in detail. Analysis is provided in Section 7. In Section 8, both simulation and experimental results are presented. Finally, we conclude in Section 9 and list some future work.

2. Related Work

Our main contribution is in the *authentication of data packets* under the threat of pollution attacks. For completeness, we also discuss related work for the authentication of CMD, ADV, and REQ packets. As Sreluge performs a limited form of intrusion detection and response during reprogramming, we also discuss some relevant work in this area.

2.1. Authentication of CMD Packets.

A CMD packet contains a command, meta-information about the new firmware, cryptographic commitment for the ensuing data packets, and a signature of the packet itself. The CMD packet is authenticated by verifying the signature. There are two kinds of signature schemes as follows.

(i) Unlimited-Time Signature Schemes. These schemes can be used to sign an unlimited number of messages and are usually based on public-key cryptography. Among public-key cryptosystems, Elliptic Curve Digital Signature Algorithm (ECDSA) is as yet known to be the most secure and efficient for resource-constrained devices, and is thus commonly prescribed [7–12]. Performance results of TinyECC [13] show that a signature verification time of 2 seconds is

achievable at a ROM cost of 14.4 KB and a RAM cost of 1.6 KB on a modest MSP430F1611 MCU. An execution time in the order of seconds, however, means that an attacker can launch DoS attacks by generating random signatures that nonetheless need to be verified. One solution is Seluge's *message-specific puzzle* [11, 14], which provides "weak pre-authentication" of a signature. Tan et al. [15] propose a minor variation of message-specific puzzle.

Wang and Kulkarni's scheme [16] uses message authentication codes, and so in principle can sign an unlimited number of messages, but it has the following problems: (i) the signature size increases with the network size; (ii) it only takes a small number of nodes to be captured to compromise a large proportion of the key pool.

(ii) Multiple-Time Signature Schemes. These schemes can only be used to sign a fixed number of distinct messages, but they use less code space and require less computation for signature verification. The disadvantage is that these schemes give longer signatures and their security is lower compared to ECC-based schemes. Notable examples are Krontiris and Dimitriou's [17] and Ugus et al. [18, 19].

Tan et al.'s scheme [20] can also be classified in this category. They propose using a separate one-way hash chain for each hop group (group of nodes with the same hop count from the base station), based on the assumption that all members of the same hop group receive the same messages at once. In practice, this assumption is difficult to uphold. Moreover, their scheme has the following disadvantages: (i) the network topology must be determined prior to deployment; (ii) uncertainties arise when the hop count of a node from the base station changes, as would happen when some nonleaf nodes fail or die of battery exhaustion.

Sreluge can utilize either kind of signature scheme.

2.2. Authentication of ADV/REQ Packets. To prevent external attackers from forging ADV/REQ, Zhang et al. [21] has specifically designed a scheme based on the Combined Public Key cryptosystem (international patent number WO/2006/074611), but it is computationally expensive and the security of the patented cryptosystem is uncertain.

There is a simpler approach. First, note that ADV packets should be sent in *authenticated local broadcast* channels whereas REQ packets should be sent in *authenticated local unicast* channels. The best known way to achieve authenticated local broadcast is for a node to establish a *cluster key chain* with its neighbors [22]. Establishing this cluster key chain can be done with most *pairwise key predistribution protocols*, which can be either deterministic (e.g., [22–24]) or probabilistic (e.g., [25–27]). Sreluge is compatible with either kind of pairwise key predistribution schemes. In Section 5.2, we will describe how we can use an example of either kind to establish a cluster key chain between a node and its neighbors. Authenticated local unicast is achieved by simply using the pairwise key between two neighboring nodes.

2.3. Authentication of Data Packets. Given a stream of data packets, we can authenticate each packet independently, for example, by signing each packet. However, this approach tends to be inefficient. Instead of signing each packet, Itani et al. propose PETRA [28], a scheme that "authenticates" each packet independently using a Bloom filter. In PETRA, the base station registers all data packets with a Bloom filter, signs and broadcasts the Bloom filter (essentially a bit-vector) globally. Each node authenticates the Bloom filter and waits for subsequent data packets. Each incoming data packet is checked against the Bloom filter. The probability that a data packet has not been registered with the Bloom filter and yet passes the check is called the *false positive rate*. Contrary to their suggestion, the false positive rate is not negligible, especially at high data packet count.

Instead, the mainstream approach is based on Gennaro and Rohatgi's algorithm [29]: given a stream of packets, every packet except the first is authenticated using the hash appended to the preceding packet, and the first packet is authenticated using a signed hash. Mathematically, let H represents a one-way hash function, then the original packets $P_1, \ldots, P_n$ are transformed into $\mathrm{Sig}(h_0)$, $P_1 \| h_1$, $P_2 \| h_2, \ldots$, P_n ($h_i = H(P_{i+1} \| h_{i+1})$, for all $i \in \{0, \ldots, n-2\}$; $h_{n-1} = H(P_n)$) before dissemination. Lanigan et al.'s Sluice [7] uses this method to authenticate a stream of pages. The page is the basic unit of data transfer introduced by Deluge: data is divided into pages and a page is divided into packets. Dutta et al.'s [8] and Nilsson et al.'s [10] schemes are similar but operate on the packet level instead of the page level. These schemes require packets within a page to arrive sequentially. Using Deng et al.'s hash tree construction [9], the data packets within a page can arrive out of order, giving the advantage of shorter dissemination time. Instead of deriving a hash tree for each page, in Seluge, Hyun et al. [11] derive a Merkle hash tree only for the first page, and append the hash of the ith data packet of the jth page to the ith data packet of the $(j-1)$th page. This represents an improvement over Deng et al.'s scheme in terms of overall communication overhead. Despite the progress in secure reprogramming, none of the aforementioned schemes can be used as is for network coding-based reprogramming because, in the aforementioned schemes, the packets of a page are transmitted as is without encoding.

The security issues of network coding have long been recognized. Among the so-called secure network coding schemes listed in Table 1, only the first two deal with recovery from pollution, so only these two are relevant. However, these two schemes assume an attacker can only inject its packets at a known and fixed low rate, while in reality, an attacker can inject its packets at a variable rate and at a rate that is comparable to the capacity of a receiver.

It is seen that by modifying the coding scheme itself—as secure network coding schemes do—little headway can be made against more powerful attackers. Later proposals focus on implementing cryptographic schemes on top of network coding. Homomorphic signature [36] and homomorphic hash function [37] are too resource-intensive for WSNs because they make extensive use of modular exponentiations. Agrawal and Boneh's homomorphic MAC [38] operates in a small finite field, and is hence more efficient. However, the resultant message authentication tag is considerably

Table 1: Secure network coding schemes.

	Resilience to eaves dropping	Pollution detection	Recovery from pollution
Jaggi et al. [30]	N	N	Y
Ngai and Yang [31]	Y	N	Y
Ho et al. [32]	N	Y	N
Tan and Médard [33]	Y	N	N
Cai and Yeung [34]	Y	N	N
Lima et al. [35]	Y	N	N

long for little security (e.g., 49 bytes for 8-bit security). Yu et al.'s scheme [39] detects pollution by attaching multiple authentication tags to a message which amount to considerable overhead. The scheme also allows polluted packets to travel a small number of hops before detection, which absolutely must not happen during reprogramming. We also add that the schemes mentioned in this paragraph are all about detection of pollution, and are not concerned with recovery from pollution.

The recent years have seen schemes that are not only more lightweight, but also include recovery mechanisms. Buttyán et al. [6] address the problem of recovering data from coding-based distributed storage systems that are subjected to pollution attacks. In their problem setting, however, downloading data has negligible cost, while in our case, to receive more data packets is costly. Dong et al.'s DART [40, 41] emulates the delayed key disclosure mechanism of μTESLA [42]. Like μTESLA, DART requires the nodes to be time-synchronized. In the ith time interval, a sender sends encoded packets to a receiver. In the $(i + 1)$th interval, the sender sends a checksum generated and signed by the base station, corresponding to the earlier encoded packets, to the receiver. While time synchronization is not too stringent a requirement, in Sreluge, we take the shortcut approach of communicating the checksums (which in our case are cryptographic hashes) to the receivers in the beginning of the protocol, thus allowing us to avoid time synchronization and to use a signature only once. We acknowledge that DART addresses a more difficult problem where the total number of packets is not known in advance. Bohli et al. [43] address the problem of authenticating LT-coded data packets in Synapse [44]. LT codes [45], like random linear codes in Rateless Deluge, are also a kind of rateless codes. As it stands, Bohli et al.'s protocol is a direct competitor to Sreluge, just as Synapse is a direct competitor to Rateless Deluge. As their work is discovered after the completion of this paper, and as they cannot disclose their source code, we must defer a comparison of their protocol to Sreluge to a later work.

2.4. Intrusion Detection and Response. It has been suggested that reactive defense schemes (such as Sreluge) are a promising approach to securing network coding-based protocols [46]. Reactive schemes are best exemplified by intrusion detection (and response) systems (IDS). An essential part of these systems is rating the reputation or trustworthiness of a node's neighbors. Ganeriwal et al.'s RFSN [47] model reputation values as Beta-distributed random variables and update the probability density function (pdf) of the random variables using Bayesian inference based on the outcome of binary events (e.g., neighbor either did or did not forward a packet as requested). Sun et al. [48] offer more alternatives for modelling trust. Many more schemes are described in Lopez et al.'s survey [49].

Due to the brief duration of reprogramming, it is an overkill to build an entire IDS into Sreluge. Rather, it is more beneficial to take the results of a reprogramming session as input to an enclosing IDS. In Section 6, we will give an example of such interaction.

3. Attacker Model

An attacker is computationally bounded. This is a standard cryptographic assumption, implying even if the attacker has access to supercomputers, its computing power is at most polynomial. Most key management schemes in the WSN literature are built on this assumption.

The base station is the only trusted entity in the network. An attacker may introduce its own nodes in a network, or it may capture, compromise, and reintroduce existing nodes in the network, as sensor nodes are generally not tamper-resistant. When a node is captured, its cryptographic keys may be exposed to the attacker. These cryptographic keys may allow the compromised nodes to rejoin the network later and become *internal/insider attackers*. These internal attackers can establish secure channels even with uncompromised nodes. We call internal attacker nodes that send out invalid data packets or corrupt data packets in transit *polluters*.

Pollution attacks are the focus of this paper. Attacks on the physical layer (e.g., jamming), data link layer (e.g., smart jamming), or network layer (e.g., packet dropping) are not considered, because countermeasures against these attacks are considered complementary to this work.

4. Problem Description and Key Definitions

According to the attacker model, internal attackers may fabricate control packets (CMD, ADV, REQ) and/or data packets. Fake control packets waste bandwidth and energy whereas fake data packets can corrupt the whole page a recipient is decoding. Both represent a form of DoS attack, but a fake data packet can destroy a whole page of at least ψ packets—that is equivalent to an effort ratio of ψ^{-1}, a “sound investment” for the attacker—and is thus a more

Table 2: List of frequently used symbols.

ψ	Minimum number of packets to receive per page ($\psi > 1$)
Ψ	Number of packets to receive for a page ($\psi \le \Psi \le \Psi_{\max}$)
$H(\cdot)$	A collision-resistant hash function
α_i	Packet ID of the ith encoded packet
f	*Packet ratio*; see Definition 2
l	Maximum bit-length of data payload
P	Number of pages
$\lvert \cdot \rvert$	(i) Bit-length of $\cdot$ or (ii) number of elements in set $\cdot$
w	Number of hashes that can fit in a packet
$\{\cdot\}_K$	Authenticated encryption of $\cdot$ using key K
$[\cdot]_K$	Message authentication code of $\cdot$ using key K
$\Vert$	Concatenation operator
$\mathcal{S}$	Set of suspected nodes
$\mathcal{B}$	Set of identified polluters
$\dot{s}$	Set of prime suspect ($\lvert \dot{s} \rvert \equiv 1$)
$\mathcal{I}$	Set of acquitted suspects
$\mathcal{A}$	Set of suspects investigation of which has been abandoned
N	Total number of nodes
τ	Fraction of polluters among neighbors

potent form of DoS attack. This more potent kind of attacks, called pollution attacks, is the main topic of this paper. It is worth noting that *energy-efficient link-layer jamming* is another potent form of DoS attack, but it is a topic that we have already addressed at great length in [50]. For the definition of symbols in the ensuing discussion, please refer to Table 2.

In the original Rateless Deluge (and naturally in Sreluge), every page of packets are encoded before transmission. To send a page consisting of packets $X_1, \dots, X_\psi$, the packets are encoded as $Y_1, \dots, Y_\psi$ according to

$$\begin{bmatrix} H(\alpha_1) & H^2(\alpha_1) & \cdots & H^\psi(\alpha_1) \\ H(\alpha_2) & H^2(\alpha_2) & \cdots & H^\psi(\alpha_2) \\ \vdots & & \ddots & \\ H\left(\alpha_\psi\right) & H^2\left(\alpha_\psi\right) & \cdots & H^\psi\left(\alpha_\psi\right) \end{bmatrix} \begin{bmatrix} X_{1,j} \\ X_{2,j} \\ \vdots \\ X_{\psi,j} \end{bmatrix} = \begin{bmatrix} Y_{1,j} \\ Y_{2,j} \\ \vdots \\ Y_{\psi,j} \end{bmatrix}. \quad (1)$$

In (1), $X_{i,j}$ denotes the jth byte of X_i and α_i is the packet ID corresponding to Y_i. By populating the matrix on the left-hand side with pseudorandom numbers, the matrix is rendered almost surely nonsingular. Decoding $Y_1, \dots, Y_\psi$ to $X_1, \dots, X_\psi$ is by means of Gaussian elimination. Denote by q the size of the field where $X_{i,j}$ and $Y_{i,j}$ are defined, then the probability of successful decoding is given by [5]

$$\frac{(q^\psi - 1)\prod_{i=1}^{\psi-1}(q^\psi - q^i - 1)}{q^{\psi^2}}. \quad (2)$$

We plot (2) against ψ with a field size of $q = 2^8$ (for efficiency) in Figure 2, which shows that ψ should preferably be at least 6 for this field size. On the other hand, to limit computational overhead, it is best if ψ does not exceed 20 [5]. To summarize, $6 \le \psi \le 20$ for practicality.

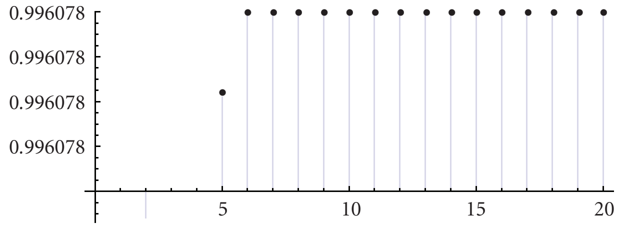


Figure 2: Probability of successful decoding versus ψ with $q = 2^8$.

A receiver must receive at least ψ packets before it can decode the entire page, because otherwise the system of linear equations in (1) is unsolvable. If any of the received packets are polluted, the whole page becomes invalid. If the receiver knows which of its neighbors are polluters, then it only has to discard all the packets from them. The problem is when an invalid page is contributed by more than one neighbor, the receiver would not know which packets are invalid, let alone which senders are rogue.

Like most other security countermeasures, the solution proposed here starts with the evaluation of the severity of an attack. The metric to use in this case is the probability of page corruption. To derive this probability, we first formally define *page-round*, *decode-round*, and *packet ratio*.

Definition 1. A **page-round** transpires whenever a page is received and decoded. When a page has exactly ψ packets, exactly one **decode-round** is required to decode the page, if the page is actually decodable. When a page has more than ψ packets, say Ψ packets, at least one but at most $\binom{\Psi}{\psi}$ decode-rounds are required to decode the page, if the page is actually decodable.

Definition 2. Suppose that in the ith page-round, a node collected Ψ_i packets. Among the Ψ_i packets, x_i packets are from neighbor u, then the **packet ratio** of neighbor u in the ith page-round, denoted by $f_u(i)$, is $f_u(i) = x_i/\Psi_i$.

The packet ratio should be understood as a random variable. Denote A_i as the event that the page is polluted in the ith page-round; Ψ_i as the number of packets received for the ith page-round; define $F_i \triangleq \sum_{u \in \{\text{good neighbors}\}} f_u(i)$. Then,

$$\begin{aligned} \Pr(A_i \mid \Psi_i = \psi) &= \Pr(F_i < 1), \\ \Pr(A_i \mid \Psi_i > \psi) &= \Pr\left(F_i < \frac{\psi}{\Psi_i}\right). \end{aligned} \quad (3)$$

Furthermore,

$$\begin{aligned} \Pr(F_i < 1) &= \Pr\left(F_i < \frac{\psi}{\Psi_i}\right) + \Pr\left(\frac{\psi}{\Psi_i} \le F_i < 1\right) \\ &\ge \Pr\left(F_i < \frac{\psi}{\Psi_i}\right), \end{aligned} \quad (4)$$

$$\therefore \Pr(A_i \mid \Psi_i = \psi) \ge \Pr(A_i \mid \Psi_i > \psi).$$

In other words, besides trying to isolate the polluters, we should set $\Psi_i > \psi$ in the presence of polluters. Setting $\Psi_i > \psi$ potentially increases the number of decode-rounds, so it should be done with discretion. The objective is $F_i \geq \psi/\Psi_i$ or $\Psi_i \geq \psi/F_i$. By limiting Ψ_i to $\Psi_i \leq \Psi_{\max}$, we have

$$\frac{\psi}{F_i} \leq \Psi_{\mathrm{i}} \leq \Psi_{\max}. \tag{5}$$

Recall F_i is the combined packet ratio of the good neighbors, hence $1 - F_i$ is the combined packet ratio of the polluters. If we can find F_T such that $F_T \geq 1 - F_i$ or $1/F_i \leq 1/(1 - F_T)$, and set $\Psi = \psi/(1 - F_T)$, then (5) is satisfied.

To identify the polluters among the neighbors, there are two opposing ways: one being the most memory-efficient and the other the most time-efficient. In the most memory-efficient approach, we use one buffer for decoding and concentrate on one sender in one page-round. Whenever a page consisting of packets solely from the sender fails verification, the sender is labelled a polluter. In the most time-efficient approach, we use n buffers for decoding (assuming n is the number of neighbors), with each of the buffers dedicated to a sender. When any of the buffers fills up, we decode the buffer and if the decoded page fails verification, we label the corresponding sender a polluter. Due to the resource limitation of a typical sensor node, and the fact that TinyOS 2.x deprecates the use of dynamic memory allocation, we adopt the most memory-efficient approach for Sreluge, that is, using only one buffer to identify one polluter in one page-round. We note though that it is possible to adopt a middle-ground approach where we use a fixed small number of buffers to identify several polluters in parallel, but this will be explored in a future work.

5. Sreluge: Predissemination

This section describes what happens prior to the dissemination of a new firmware.

5.1. On the Base Station. The firmware to be disseminated is divided into pages (whose size is not necessarily the same as the page size of a Flash memory). For example, the Deluge implementation in TinyOS 2.x allocates 1024 bytes for each page. Sreluge allocates $l\psi$ bits per page, where l is the maximum data payload size in bits and ψ is the number of packets per page. Pages are indexed from 1. From the ith page, the hash h_i is derived. The goal is to pack the hashes h_i $(i = 1, \ldots, P)$ in as few packets as possible, with the added constraint that each of these packets is independently authenticable. Following Deng et al.'s naming convention [9], let us call these packets *index packets*. An example of how index packets are constructed from 10 pages is shown in Figure 3.

The key idea is constructing a Merkle hash tree [51] and determining the height of the tree. Denote by L the height of the tree. Denote by w the number of hashes that can fit in a packet, so $w = \lfloor l/|H(\cdot)| \rfloor$. By inspecting Figure 3, we can see that L is the minimum value such that $2^L \geq \lceil \psi/(w - L) \rceil$. The root of the tree is signed. This construction is identical to Seluge's [11], except that in our case the leaves are hashes of pages, not hashes of packets.

5.2. On the Nodes. In order for a node u to send authenticated messages to its neighbor v, u needs to establish a pairwise key K_{uv} with v. A REQ from u to v is sent as $\mathrm{REQ}\|[REQ]_{K_{uv}}$.

Once K_{uv} is established, setting up an authenticated local broadcast channel from u to its neighbors is straightforward. First, u generates a cluster key chain $A_{u,n}, A_{u,n-1}, \ldots, A_{u,0}$, where $A_{u,i} = H(A_{u,i+1})$. Then, u sends the key chain commitment $A_{u,0}$ to v encrypted with K_{uv}. An ADV from u to its neighbors is broadcast as $\mathrm{ADV}\|i\|[ADV\|i]_{A_{u,i}}$. Authentication is successful if $H^{(i-j)}(A_{u,i}) = A_{u,j}$ for some $j < i$, and $[\mathrm{ADV}\|i]_{A_{u,i}}$ is valid.

We now describe how either a deterministic key predistribution scheme called LEAP+ [22] or a probabilistic key predistribution scheme by Eschenauer and Gligor [25] can be used to establish K_{uv}.

LEAP+. Before deployment, every node gets an initial key K_{IN}. At boot-up, every node u computes (i) an individual key $K_u = [ID_u]_{K_{IN}}$; (Here, we are using a MAC, such as CBC-MAC, as a pseudorandom function [52].), and (ii) a cluster key chain $A_{u,n}, A_{u,n-1}, \ldots, A_{u,0}$. u establishes K_{uv} with v, for all $v \in \{u\text{'s neighbors}\}$ using the following protocol:

$$\begin{aligned} u \longrightarrow * :&\quad ID_u \\ v :&\quad K_{uv} \longleftarrow [ID_u]_{K_v} \\ v \longrightarrow u :&\quad \{ID_v \,||\, ID_u\}_{K_{uv}} \\ u :&\quad K_v \longleftarrow [ID_v]_{K_{IN}}, \qquad K_{uv} \longleftarrow [ID_u]_{K_v}. \end{aligned} \tag{6}$$

Some time after the above protocol, K_{IN} is erased. This scheme is only suitable for static networks.

Eschenauer-Gligor. Before deployment, every node gets m keys randomly chosen from a key pool of n keys. During neighbor discovery, every pair of neighbors can find at least one common key at a probability of $1 - \binom{n-m}{m}/\binom{n}{m}$. Using the common key(s), two neighbors can establish a pairwise key. If there is no common key between two neighbors u and v, u and v can go through a common neighbor w who has common key(s) with both u and v, to establish a pairwise key. This scheme is suitable for both static networks and networks with limited mobility.

Note that with these authentication channels in place, an internal attacker will still be able to send authentic *yet* invalid data packets, but absolutely no external attackers can send their packets through. Moreover, an internal attacker could, after compromising sufficient keys, impersonate any node in the network. The amount of effort required of the attacker depends on the resilience of the key management scheme used. This topic is extensively studied in the key management literature (e.g., see [22, Section 3.6], for LEAP+ and [53, Section 4.2], for Eschenauer-Gligor), and will not be pursued here.

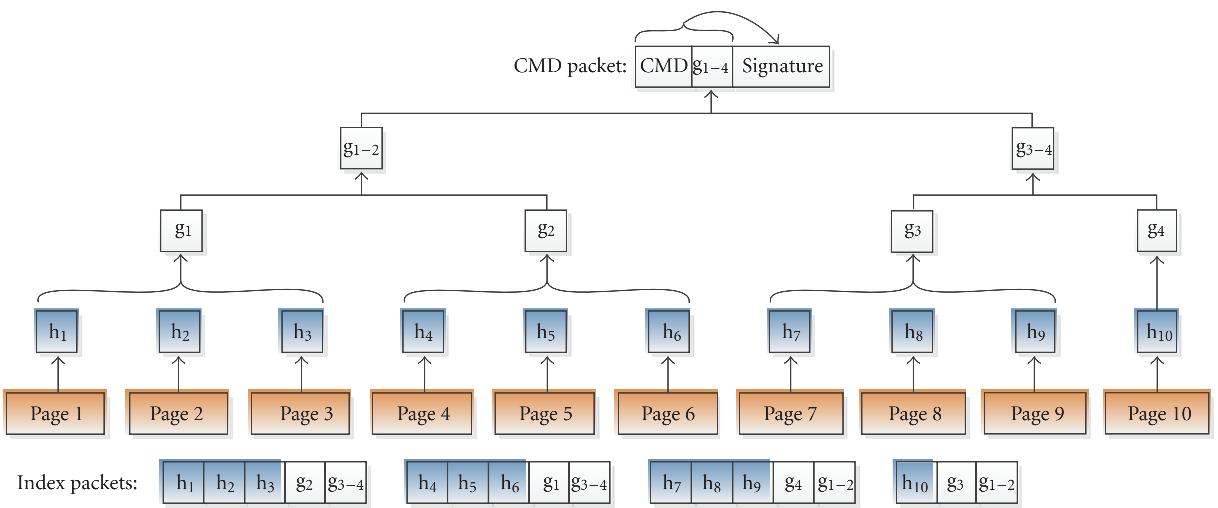


Figure 3: Construction of individually authenticable index packets using the Merkle tree from 10 pages.

6. Sreluge: Dissemination

To start disseminating the new firmware, the base station sends a CMD packet consisting of the root of the Merkle hash tree and a signature of the root (Figure 3). The signature is used to authenticate the root, and the root in turn is used to authenticate the index packets. For example, when the first index packet of Figure 3 arrives, it is partitioned into h_1, h_2, h_3, g_2, and g_{3-4}. If $H(H(H(h_1\|h_2\|h_3)\|g_2)\|g_{3-4})$ equals the root, then the first index packet is authenticated. The ith authenticated index packet is used to authenticate the ith page later.

Following the index packets, the base station sends the pages in the form of encoded data packets. For each page, a minimum of ψ packets, encoded according to (1), are to be received. Pollution attacks happen when some of these encoded packets are maliciously modified. Note that unlike the index packets which are individually authenticable, these encoded packets are not. It is only by failing to verify a decoded page using its corresponding hash that we can deduce the presence of polluted encoded packets in the page. The core of Sreluge is a pollution detection engine that partitions the neighbors of a node into five sets:

(i) $\mathcal{S}$: nodes that are suspected to be polluters;

(ii) $\mathcal{B}$: nodes that have been blacklisted as polluters;

(iii) $\mathcal{A}$: nodes that were suspected but investigation of which has been abandoned because Sreluge timed out waiting for their packets;

(iv) $\mathcal{I}$: innocent nodes that have been acquitted and are no longer suspected;

(v) Unclassified: nodes that do not belong to any of the above categories.

The basic idea (see Figure 4) is that whenever a decoded page fails verification, the neighbors that contributed to that page are placed in $\mathcal{S}$. A *prime suspect*, defined as a single-element set $\dot{s} \subseteq \mathcal{S}$, is picked based on some criteria, and a page is solicited only from $\dot{s}$. If the page from $\dot{s}$ passes verification later, then $\dot{s}$ is put into $\mathcal{I}$ and removed from $\mathcal{S}$. If however the page fails verification, $\dot{s}$ is put into $\mathcal{B}$ and removed from $\mathcal{S}$. If however after some timeout, the page expected of $\dot{s}$ still has not fully arrived, the investigation of $\dot{s}$ is abandoned, that is, $\dot{s}$ is placed in $\mathcal{A}$ and removed from $\mathcal{S}$.

After the firmware is successfully received, Sreluge could pass the results, in the form of $\mathcal{B}$, to an enclosing IDS. For example, suppose the enclosing IDS is based on RFSN, then a neighbor is associated with two parameters: α and β, which specify a Beta distribution function. The IDS will change (α, β) to $(\alpha, \beta + 1)$ if the corresponding neighbor $\in \mathcal{B}$, or to $(\alpha + 1, \beta)$ if otherwise. This way, Sreluge does not cause the permanent isolation of a falsely identified polluter due to a small but finite false positive rate (see Section 7.2.1).

Implementing Sreluge requires modifying the following components of Rateless Deluge: (i) the decode procedure, (ii) the data processing procedure—the procedure that processes incoming data packets, and (iii) the data request procedure—the procedure that sends REQ packets. These modifications are described in detail in the following subsections.

6.1. The Decode Procedure. Encoded data packets are collected in the $\Psi_{\max}$-size *c-buffer*, and decoded in the ψ-size *d-buffer*. The procedure `decode` is invoked after Ψ packets have been collected. It is paramount to note that before any pollution is detected, $\Psi = \psi$, but once pollution is detected, Ψ is varied between ψ and $\Psi_{\max}$ to satisfy (5). We will give more detail on the adjustment algorithm in Section 6.1.3.

The main task of decode (see Procedure 1) is to decode each combination of ψ packets until one of the following happens:

(i) decoding fails due to linear dependence, and there are not enough potentially linearly independent packets left in *c-buffer* (lines 10–15);

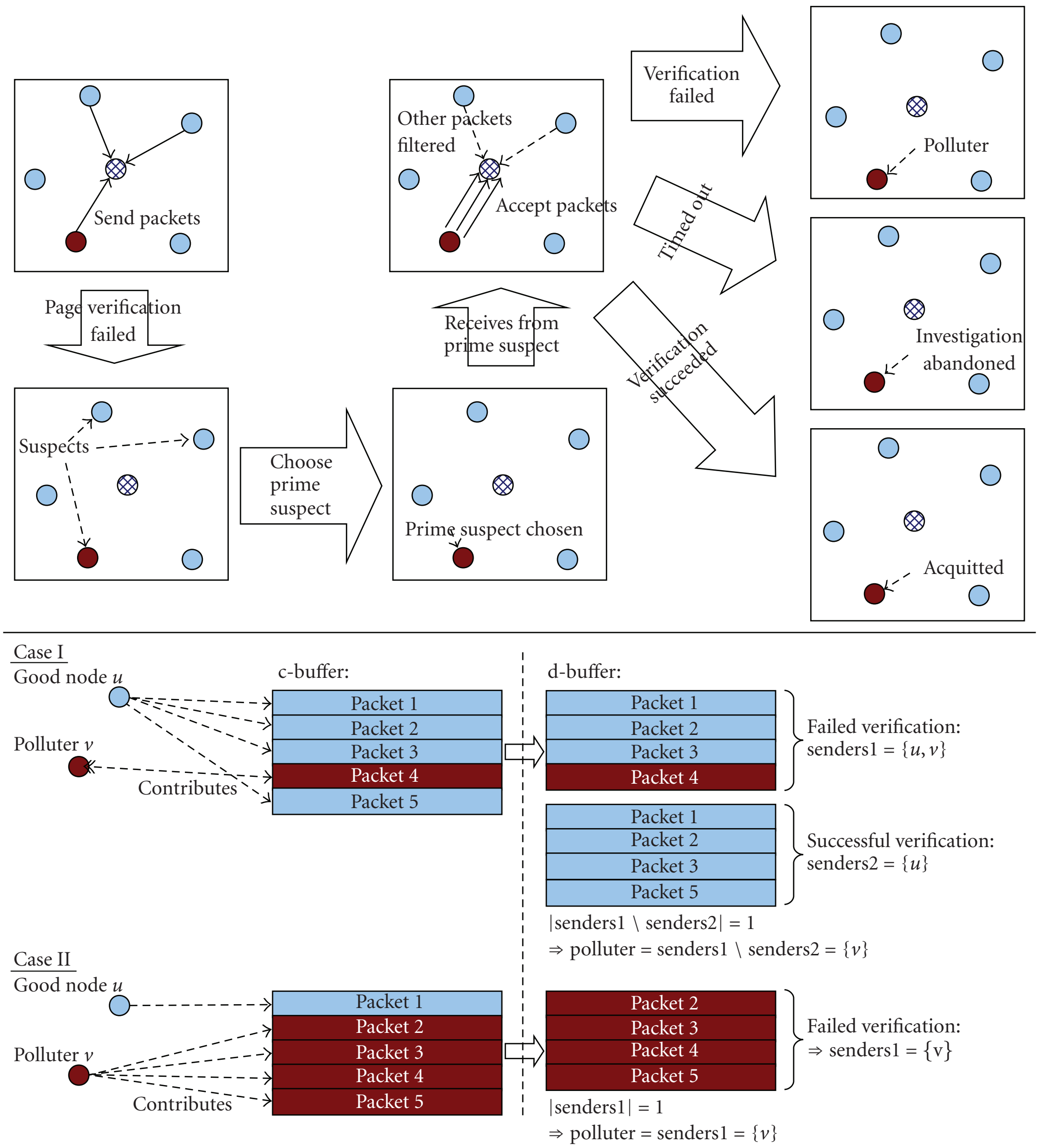


Figure 4: (Top) A simplified view of neighbor classification upon detection of page pollution. (Bottom) Two ways of identifying a polluter, with $\psi = 4$ and $\Psi = 5$ in this example.

(ii) decoding succeeds but verification fails, and there are not enough potentially intact packets left in *c-buffer* (lines 16–26);

(iii) decoding succeeds and verification succeeds (lines 27–35).

At the end of `decode`, the variable `decodeStatus` is assigned one of three values. Depending on the value of `decodeStatus`, Sreluge takes the following action.

`LinearDependence`. The linearly dependent packets in *d-buffer* are discarded and more packets are set to be collected.

`VerificationFailure`. The algorithm is flowcharted in Figure 5. The main idea is that if a polluter has been found, then a new prime suspect is chosen and Ψ is adjusted. If a polluter has not been found, then more suspects are added to $\mathcal{S}$, *before* a new prime suspect is chosen and Ψ is adjusted. The three procedures: `findAddSuspects`, `newPrimeSuspect`, and `adjustMaxNumPktsRecvd` will be discussed in more detail later.

`VerificationSucceess`. The algorithm is flowcharted in Figure 6. If there is already a prime suspect, the prime suspect is acquitted. Then, a new prime suspect is chosen and Ψ is adjusted. If there has not been a prime suspect, Ψ is simply set to ψ, on the optimistic assumption that there will subsequently be no pollution. The verified page is written to the Flash memory.

6.1.1. *Procedure `findAddSuspects`.*

This purpose of this procedure (see Procedure 2) is to identify the senders to be

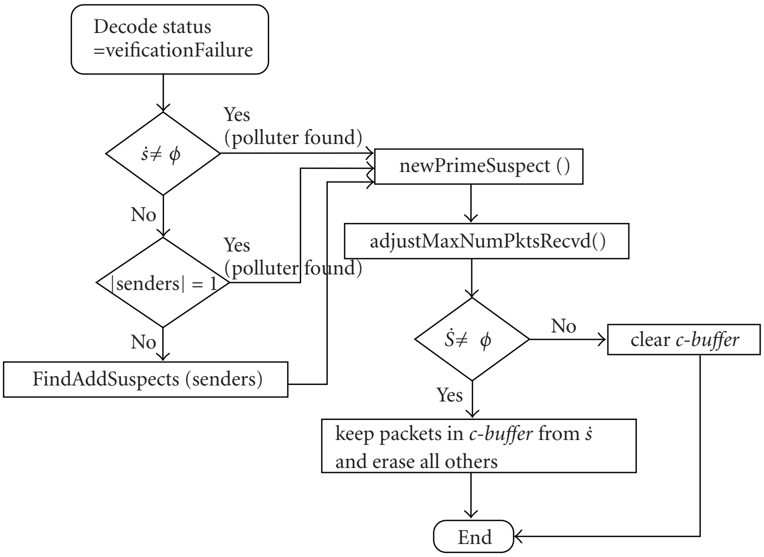


Figure 5: Flowchart for handling the case `decodeStatus = verificationFailure`. $\dot{s}$ is the set of prime suspect. "senders" refer to the senders of the packets in *c-buffer*.

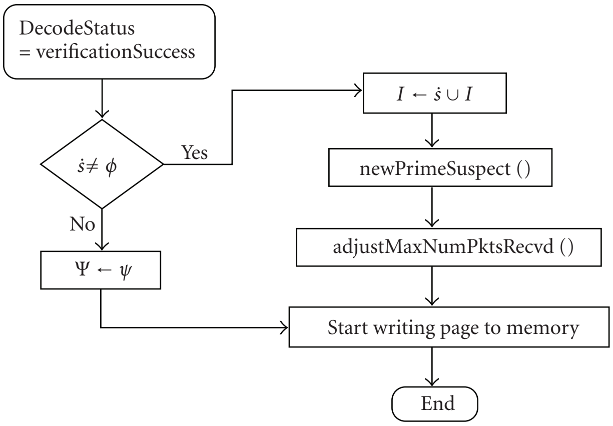


Figure 6: Flowchart for handling the case `decodeStatus = verificationSuccess`. $\dot{s}$ is the set of prime suspect. $\mathcal{I}$ is the set of acquitted ex-suspects.

classified in $\mathcal{S}$. Denote by senders the set of senders who contributed to a polluted page. Any member of `senders` that is not already in either $\mathcal{S}$, $\mathcal{B}$, $\mathcal{I}$, or $\mathcal{A}$ is admitted to $\mathcal{S}$. If there is no such member, then any member of `senders` that is not already in $\mathcal{S}$, $\mathcal{B}$, or $\mathcal{I}$ but maybe in $\mathcal{A}$ is added. Recall that $\mathcal{A}$ is the set of neighbors investigation of which had been abandoned due to Sreluge timing out waiting for their packets. We take the opportunity to reopen investigation of these neighbors here.

6.1.2. Procedure `newPrimeSuspect`. The purpose of this procedure (see Procedure 3) is to pick a prime suspect among the suspects. The logic of the procedure revolves around the concept of the packet ratio (see Definition 2). To obtain a verdict on the prime suspect as soon as possible, we hope to get ψ packets from it in the shortest time possible. Clearly, the higher the packet ratio of the prime suspect, the more packets we can get from it within a fixed amount of time. Hence, the rationale of `newPrimeSuspect` is to choose the suspect with the highest packet ratio *for the next page-round*. The problem is future packet ratios are not known in advance and thus must be forecast.

To cater for the possibility that the forecast might be wrong and the packet ratio might turn out to be zero, we need to set a time-out. Denote by $\widehat{f}(i+1)$ the packet ratio of the prime suspect forecast for the $(i+1)$th page-round. If $\widehat{f}(i+1)$ is larger or equal to a *threshold value* f_T, then we will have to collect at most $1/f_T$ packets in order to receive the first packet from the prime suspect. In other words, if we set a timer (called `probationTimer` in `newPrimeSuspect`) to $1/f_T$, then the first packet from the prime suspect should arrive before the timer fires. The `probationTimer` is decremented for every packet received (see line 6 of Procedure 4).

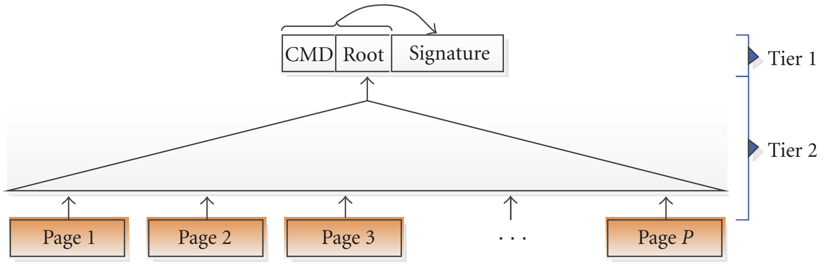


Figure 7: Reduction of security proof.

Based on the analysis above, the challenges are (i) to forecast the suspects' packet ratios for the next page-round, and (ii) to determine a suitable value for f_T.

We first present the solution for challenge (i). The packet ratios of a sender are a discrete-valued stochastic process that depends on many factors, for example, the underlying medium access control protocol, the topology, the radio environment, the transfer rate of the Flash memory, and so forth. The complexity forbids the derivation of a comprehensive analytical model. However, with respect to the standard TinyOS 2.x protocol stack, the time series analysis and simulation-based study in Section 8.1 suggest that the forecast $\widehat{f}(i+1) = f(i)$ is good enough.

As the solution for challenge (ii), we need a suitable value for f_T. A value for f_T should be high enough so that we do not need to wait for too long before we collect enough of the prime suspect's packets, but low enough so that we do not miss out actual polluters with low packet ratios. Simulation results in Section 8.2 suggest that $f_T = 0.5$ is a reasonable tradeoff.

6.1.3. Procedure `adjustMaxNumPktsRecvd`*.* The purpose of this procedure is to adjust Ψ according to whether there is a prime suspect:

(i) When there is a prime suspect, packets will be received solely from the prime suspect, hence $\Psi \leftarrow \psi$.

(ii) When there is no prime suspect, packets will be received from polluters and nonpolluters alike. To satisfy (5), Ψ should be set to $\psi/(1 - F_T)$, where F_T is at least as large as the combined packet ratio of the polluters. We set $F_T = |\mathcal{S} \cup \mathcal{A}| f_T$, because (i) $|\mathcal{S} \cup \mathcal{A}| f_T$ tends to overestimate the combined packet ratio of the polluters, thereby satisfying the requirement for F_T; (ii) this facilitates a termination proof for Sreluge in the form of Proposition 1.

There are two special cases that need to be taken care of:

(1) when $\psi/(1 - |\mathcal{S} \cup \mathcal{A}| f_T) > \Psi_{\max}$;

(2) when $\psi/(1 - |\mathcal{S} \cup \mathcal{A}| f_T) < 0$.

These cases happen when there are many suspects, in which case the necessary value for Ψ would be too high. For these cases, we set $\Psi = \psi$.

6.2. The Data Processing Procedure. Upon the reception of a data packet, the procedure `filterData` is invoked. The main tasks of `filterData` (see Procedure 4) are to filter packets from blacklisted polluters when there is no prime suspect (lines 1–4) and to filter packets not sent by the prime suspect when there is a prime suspect (lines 5–18).

6.3. The Data Request Procedure. Before a REQ is sent, the destination node has to be decided, and obviously, the destination cannot be any member of $\mathcal{B}$. If there is a prime suspect, then the REQ is sent to the prime suspect. Otherwise, the REQ is sent to the neighbor with the highest forecast packet ratio that is not a member of $\mathcal{B}$. Due to the simplicity of the algorithm, the pseudocode of this procedure is omitted.

7. Analysis of Sreluge

A complete analysis would involve all the components in Figure 1, but it is only meaningful to focus on the components contributed by this work, that is, the component for pre-dissemination processing of firmware and the component for data authentication. The two components are analyzed together, and the analysis consists of two parts: cryptographic properties (Section 7.1) and noncryptographic properties (Sections 7.2.1 and 7.2.2).

7.1. Cryptographic Properties. In informal terms, the security objective is to prevent an attacker from substituting a firmware page(s) with its own page(s). The security proof is reduced to the security proof of the signature schemes labelled Tier 1 and Tier 2 in Figure 7. In formal terms, the security objective is that both Tier 1 and Tier 2 signature schemes are *existentially unforgeable under known-message attacks* [54, Definition 1.5]. Usually, a signature scheme is required to satisfy a stronger security notion: existentially unforgeable under (adaptive) chosen-message attacks [54, Definition 1.6]. However, since the base station is trusted in the attacker model, an attacker cannot adaptively choose messages to sign, so an attacker can at most only launch known-message attacks.

As a candidate for the Tier 1 signature scheme, ECDSA has been proven existentially unforgeable under chosen-message attacks in a generic group model by Brown [55]. While the generic group model used in Brown's proof is

```
// Precondition: packets have been collected in c-buffer
// Output: decodeStatus
senders 1 ← $\emptyset$;
senders 2 ← $\emptyset$;
decodeStatus ← verifcationFailure
for $i \leftarrow 1$ to $\binom{\Psi}{\psi}$ do
    Choose a fresh combination of $\psi$ packets in c-buffer
    Copy the packets from c-buffer to d-buffer
    Decode the packets in d-buffer
    if decoding failed due to linear dependence then
        $x \leftarrow$ |linearly dependent packets in d-buffer|
        If $\Psi - x < \psi$ then
            decodeStatus ← linearDependence
            break
        end if
    else if decoding succeeded but verification failed then
        // Case II in Figure 4
        senders1 ← senders of packets in d-buffer
        if |senders1| = 1 then
            $\mathcal{B} \leftarrow senders1 \cup \mathcal{B}$
            x ← |packets in c-buffer not from senders1|
            if $x < \psi$ then
                decodeStatus ← verificationFailure
                break
            end if
        end if
    else if decoding and verification succeeded then
        // Case I in Figure 4
        senders2 ← senders of packets in d-buffer
        if senders1 $\neq \emptyset$; and |senders1 \ senders2|= 1 then
            $\mathcal{B} \leftarrow$ |senders1 \ senders2| $\cup\ \mathcal{B}$
        end if
        decodeStatus ← verificationSuccess
        break
    end if
end for
```

Procedure 1: Decode.

```
candidates ← senders \ ($\mathcal{S} \cup \mathcal{B} \cup \mathcal{I} \cup \mathcal{A}$)
if |candidates| = 0 then
    candidates ← senders \ ($\mathcal{S} \cup$ B $\cup\ \mathcal{I}$)
    $\mathcal{S} \leftarrow$ candidates $\cup\ \mathcal{S}$
end if
```

Procedure 2: FindAddSuspects (senders).

```
If $\mathcal{S} = \emptyset$ and $\dot{s} = \emptyset$ then
    $x \leftarrow$ member of $\mathcal{S}$ whose $\hat{f}_x(i+1)$ is the highest
    if $\hat{f}_x(i+1) \geq f_T$ then
        $\dot{s} \leftarrow \{x\}$
        probationTimer ← $1/f_T$
        $\mathcal{S} \leftarrow \mathcal{S} \setminus \dot{s}$
    end if
end if
```

Procedure 3: newPrimeSuspect.

invalid for supersingular elliptic curves [56], the model is valid for the curves defined in the standard [57]. There are currently no formal security proofs for Krontiris and Dimitriou's [17] and Ugus et al.'s [18] multiple-time signature schemes.

As a candidate for the Tier 2 signature scheme, the Merkle tree has been proven existentially unforgeable under chosen-message attacks [58]. Therefore, it can be concluded here that if ECDSA is used at Tier 1 and the Merkle tree is used at Tier 2, then the component for pre-dissemination processing of firmware and the component for data authentication of Sreluge are provably secure.

7.2. Noncryptographic Properties. In terms of noncryptographic properties, our main concern is the resilience of Sreluge against the following DoS attacks.

(i) DoS Attacks on the Index Packets. Since the index packets can be received out of order, they are more resilient to DoS attacks than if they have to be received in order such as in [9].

(ii) DoS Attacks on the Encoded Date Packets. As explained, pollution is a major threat due to the nature of network coding. Since Sreluge functions like an intrusion detection system during the dissemination phase, we are interested in the *false negative rate* and *false positive rate* of the algorithm for identifying polluters. Another important goal is to prove the *total correctness* of the algorithm. These three properties ensure that, provided the underlying cryptographic framework is secure, no polluter can escape detection and no polluter can cause Sreluge to execute indefinitely. The analysis of these properties is the focus of the remainder of this section.

7.2.1. False Negative and False Positive Rates. A false negative is when a node fails to identify a polluter whereas a false positive is when a node mistakes a good neighbor as a polluter.

Theorem 1. *Sreluge has zero false negative rate and a false positive rate that is upper-bounded by*

$$1-(1-\mathrm{BER})^{l\max(\Psi-\psi,\psi)}, \tag{7}$$

where BER *is the bit error ratio, and other symbols are as defined in Table 2.*

Proof. There are two cases when a node is labelled as a polluter (see Figure 4). We discuss these two cases separately.

Case I. This case is handled by line 31 of procedure decode. Denote by S_{bad} the set of packets that decode into a page that fails verification and by S_{good} the set of packets that decode into the correct page. Then, the packets in $S_{\mathrm{bad}} \setminus S_{\mathrm{good}}$ must be responsible for the pollution. Moreover, if all of $S_{\mathrm{bad}} \setminus S_{\mathrm{good}}$ originated from a single sender, there are two possibilities: (i) either the sender is a polluter or (ii) the sender is good but some of its packets are unintentionally corrupted.

(i) The Sender Is Indeed a Polluter. Sreluge correctly identifies it as a polluter, so there is no false negative.

(ii) The Sender Is Actually Good. Denoting by E_i the event that $|S_{\mathrm{bad}} \setminus S_{\mathrm{good}}| = i$; by BER the bit error ratio; by l the number of bits in a data payload. Then, the probability of this false positive is

$$\begin{aligned}\mathrm{FPR}_{\mathrm{I}} &= \sum_{i=1}^{\Psi-\psi} \Pr(E_i)\left[1-\Pr(\text{packet is intact} \mid E_i)^i\right]\\ &= \sum_{i=1}^{\Psi-\psi} \Pr(E_i)\left[1-(1-\mathrm{BER})^{li}\right]\\ &\le \left[1-(1-\mathrm{BER})^{l(\Psi-\psi)}\right]\sum_{i=1}^{\Psi-\psi}\Pr(E_i)\\ &= 1-(1-\mathrm{BER})^{l(\Psi-\psi)}.\end{aligned} \tag{8}$$

Case II. This case is handled by line 20 of procedure decode. Again, there are two possibilities as follows.

(i) The Sender Is Indeed a Polluter. Sreluge correctly identifies it as a polluter, so there is no false negative.

(ii) The Sender Is Actually Good. Denoting by BER the bit error ratio and by l the number of bits in a data payload, the probability of this false positive is

$$\begin{aligned}\mathrm{FPR}_{\mathrm{II}} &= 1-\Pr(\text{packet is intact})^{\psi}\\ &= 1-(1-\mathrm{BER})^{l\psi}.\end{aligned} \tag{9}$$

Meanwhile, the combined false positive rate is

$$\begin{aligned}\mathrm{FPR} &= \mathrm{FPR}_{\mathrm{I}}\cdot\Pr(\text{Case I})+\mathrm{FPR}_{\mathrm{II}}\cdot\Pr(\text{Case II})\\ &= (\mathrm{FPR}_{\mathrm{I}}-\mathrm{FPR}_{\mathrm{II}})\Pr(\text{Case I})+\mathrm{FPR}_{\mathrm{II}}.\end{aligned} \tag{10}$$

Since (10) is linear in Pr(Case I), the maximum of *FPR* is either when $\Pr(\text{Case I}) = 0$ or when $\Pr(\text{Case I}) = 1$. Therefore,

$$\mathrm{FPR} \le \max(\mathrm{FPR}_{\mathrm{I}},\mathrm{FPR}_{\mathrm{II}}) = 1-(1-\mathrm{BER})^{l\max(\Psi-\psi,\psi)}. \tag{11}$$

The implication of the above analysis is that the false negative rate is zero, and the false positive rate is negligible in practice. □

7.2.2. Total Correctness. An algorithm is said to satisfy total correctness if given a precondition, the algorithm is guaranteed to terminate and the resulting state satisfies the postcondition. In our case, the precondition is that after discounting polluters, a node is at least 1-connected to the base station whereas the postcondition is that a node eventually receives all the pages.

It is trivial to show that if a node has only good neighbors and the collective transmission rate of the good neighbors is nonzero, then the node will eventually receive all pages and Sreluge running on the node will eventually terminate. If a

```
if $\dot{s} = \emptyset$ then
    if sender $\in \mathcal{B}$ then drop packet
    else accept packet
    end if
else
    probationTimer $\leftarrow$ probationTimer $-$ 1 // Dec. timer
    if sender $\in$ B or sender $\notin \dot{s}$ then
        drop packet
    else if sender $= \dot{s}$ then
        accept packet
        probationTimer $\leftarrow 1/f_T$ // Reset timer
    end if
    if probationTimer $=$ 0 then
        A $\leftarrow \dot{s} \cup$ A
        clear c-buffer
        newPrimeSuspect()
    end if
end if
```

Procedure 4: FilterData(packet).

node has one polluter among its neighbors, Proposition 1 says that Sreluge will eventually terminate with the node successfully receiving all pages. Proposition 1 depends on Definition 3.

Definition 3. The packet ratio f is *oscillatory around* f_T if it can be represented by a two-state Markov process, with states $A : f \geq f_T$ and $B : f < f_T$, and transition matrix

$$\begin{bmatrix} \Pr(A \mid A) & \Pr(B \mid A) \\ \Pr(A \mid B) & \Pr(B \mid B) \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}. \qquad (12)$$

None of the packet ratio processes in Figure 9 is oscillatory around any value of f_T. We can safely assume that the packet ratio processes are generally not oscillatory around any reasonable value of f_T. It is *theoretically* feasible for an attacker to manipulate its packet ratio such that the packet ratio is oscillatory around a specific f_T, but in practice this requires the attacker to have continuous and absolute influence on the packet ratios of the good neighbors, which is difficult to achieve.

Proposition 1. Suppose a node has one polluter among its neighbor and at least one good neighbor with positive transmission rate. Provided the packet ratio of the polluter is not oscillatory around f_T, Sreluge will eventually terminate with all pages successfully received and verified.

Proof. For the proof, let us label the node running Sreluge as v and the polluter as u. Figure 8 shows the finite state automaton representing Sreluge. Sreluge would not terminate if it is trapped in some loop in Figure 8. We first note that the transitions c and d do not contribute to any loop because in states 3 and 4, $\dot{s}$ is eliminated from further consideration. We then note that there are three potential loops: (i) *a-e-h*, (ii) *b-g*, and (iii) *a-e-h-b-g*.

Consider first the possibility of infinite loop *a-e-h*. Transition e is triggered by the condition $f_u|_2^5 < f_T$. After transition h, in state 1, the condition $\hat{f_u} = f_u|_2^5 < f_T$ (recall that in Section 6.1.2, we defined $\hat{f_u}$ to be the most recent value of f_u) triggers transition b instead of a. Hence, the infinite loop *a-e-h* is not possible. Using the same reasoning, the infinite loop *b-g* can be shown to be impossible.

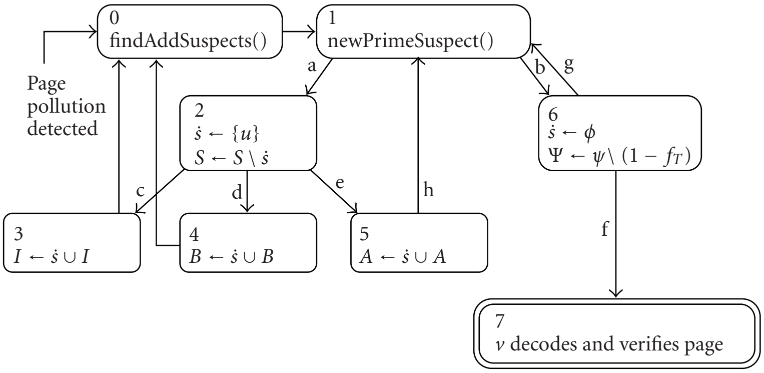
Consider now the possibility of infinite loop *a-e-h-b-g*. Given the packet ratio f_u is *not* oscillatory around f_T, we can write the transition matrix of the Markov process describing f_u as

$$\begin{bmatrix} \Pr(A \mid A) & \Pr(B \mid A) \\ \Pr(A \mid B) & \Pr(B \mid B) \end{bmatrix} = \begin{bmatrix} \pi_{11} & \pi_{12} \\ \pi_{21} & \pi_{22} \end{bmatrix}, \qquad (13)$$
$$\pi_{11} + \pi_{12} = 1, \quad \pi_{21} + \pi_{22} = 1, \quad \pi_{12}\pi_{21} < 1,$$

where the states A and B of the Markov process correspond to the conditions $f_u \geq f_T$ and $f_u < f_T$, respectively. The infinite loop *a-e-h-b-g* necessitates that the process alternates between states A and B. The probability of this happening indefinitely is $\lim_{i \to \infty}(\pi_{12}\pi_{21})^i$, but $\pi_{12}\pi_{21} < 1 \Rightarrow \lim_{i \to \infty}(\pi_{12}\pi_{21})^i = 0$, so while the loop *a-e-h-b-g* is possible, it cannot occur indefinitely. □

The implication of the above analysis is that u will eventually be labelled as a polluter (in state 4), or abandoned (in state 5). In the former case, all polluted packets from u will be filtered and as long as the collective transmission rate of the good neighbors is nonzero, enough intact packets will be received. In the latter case, at least ψ in Ψ packets will be intact, and the ψ packets can be decoded and successfully verified. Note that u will never be labelled as innocent (in state 3) because according to Theorem 1, Sreluge has zero false negative rate.

Proposition 1 addresses the case where there is one polluter. The case where a node has multiple polluters among


0
findAddSuspects()
1
newPrimeSuspect()
Page
pollution
detected
2
$\dot{s} \leftarrow \{u\}$
$S \leftarrow S \setminus \dot{s}$
6
$\dot{s} \leftarrow \phi$
$\Psi \leftarrow \psi \setminus (1 - f_T)$
3
$I \leftarrow \dot{s} \cup I$
4
$B \leftarrow \dot{s} \cup B$
5
$A \leftarrow \dot{s} \cup A$
7
v decodes and verifies page
a
b
c
d
e
f
g
h

a: $\hat{f}_u \geq f_T$
b: $S = \phi \vee \hat{f}_u < f_T$
c: $f_u |_2^3 \geq f_T$ and verification succeeded
d: $f_u |_2^4 \geq f_T$ and verification failed
e: $f_u |_2^5 < f_T$
f: $f_u |_6^7 \leq f_T$
g: $f_u |_6^1 > f_T$
h: none

Definitions:
$\hat{f}_u \triangleq$ the forecast value of the next f_u
$f_u |_i^j \triangleq$ the value of f_u between states i and j

FIGURE 8: A finite state automaton for proving the total correctness of Sreluge, where u is the polluter and v is the node running Sreluge.

its neighbors requires a more thorough proof using formal logic, which will be given in a later work.

8. Simulation and Experimental Results

Simulations are performed using TOSSIM Live [59] of TinyOS 2.x. The advantage of using TOSSIM is that large scale network dynamics can be observed more easily, and the code can be ported to actual sensor nodes with little to no modification.

Two classes of polluters are simulated as follows.

(i) Basic polluters behave like normal nodes except that they send out invalid data packets. A Basic polluter can be implemented, for example, by capturing a normal node and setting the memory address of the outgoing data packet to an arbitrary value.

(ii) Aggressive polluters are more aggressive than Basic polluters, in that (i) they advertise the falsehood that they possess all the pages, and (ii) they reply to requests immediately with invalid data packets. Implementing an Aggressive polluter requires more effort from the attacker. Most likely, the attacker would need to program its own node, and copy the necessary cryptographic keys from a captured node to its own node. Basic polluters are significantly easier to implement than Aggressive polluters, so we deem Basic polluters a more prevalent threat.

50 and 100 nodes are distributed in a 40 m × 40 m area in 12 random topologies and a 19-page firmware is disseminated. The networks with 50 nodes have an average network degree of 13 whereas the networks with 100 nodes have an average network degree of 27. Radio parameters (as defined in [60]) are configured as in Table 3. Sreluge-specific parameters are configured as such: $\psi = 8$, $\Psi_{\max} = 1.5\psi$ (i.e., $\psi \leq \Psi \leq 1.5\psi$). ADV, REQ, and data packets are fixed in size and are 14, 10, and 29 bytes long, respectively. The three cases $\tau = 0\%$, $\tau = 10\%$, and $\tau = 20\%$ are simulated. Four performance metrics are used.

TABLE 3: Radio parameters used in the simulations.

Path loss exponent	4.7
Shadowing s.d.	3.2
d_0	1.0
$PL(d_0)$	55.4
Noise floor	−105.0
S_{11}	0.9
S_{12}	−0.7
S_{21}	−0.7
S_{22}	1.2
White Gaussian noise s.d.	4.0

(i) Number of REQ Packets Transmitted per Page per Node. In the ideal case, and in the absence of attacks, a node sends $1/n$ request packet per page.

(ii) Number of Data Packets Transmitted per Page per Node. A data packet is typically much longer than a request packet, hence this metric is necessary. In the ideal case, and in the absence of attacks, a node sends ψ/n data packets per page.

(iii) Number of Decode-Rounds per Page per Node . In the ideal case, and in the absence of attacks, a node decodes each page once.

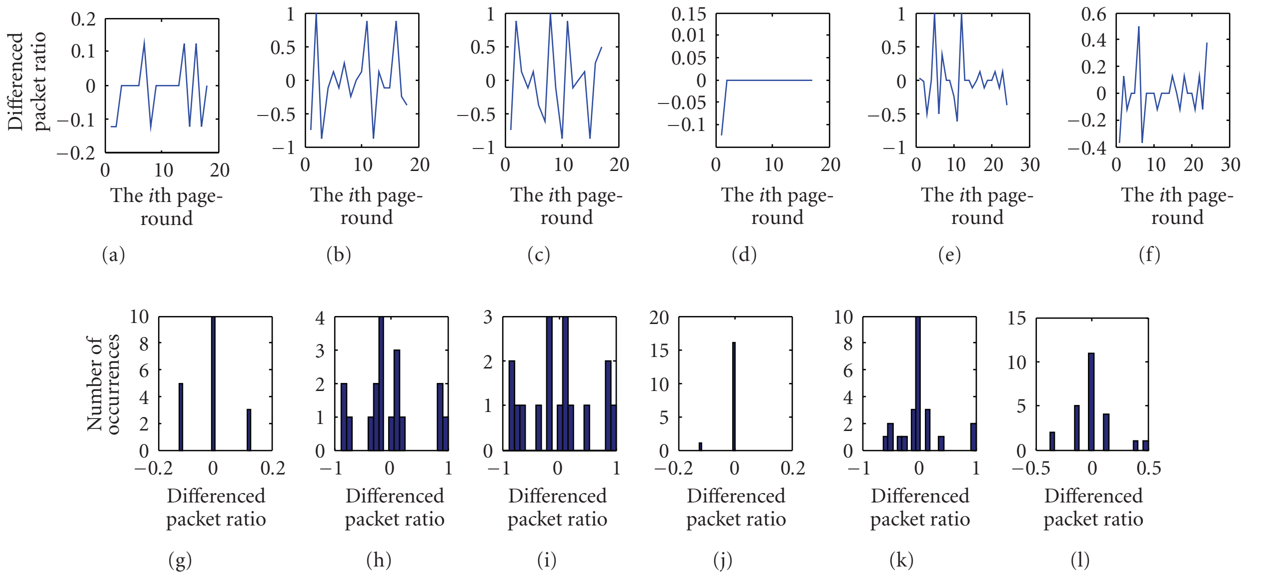


Figure 9: (a)–(f) Six randomly sampled packet ratio processes plotted against page-round, after first-order differencing. (g)–(l) Histograms of the differenced packet ratio processes appear Gaussian for most cases. The best ARIMA models for the processes are found to be ARIMA(6,1,2), ARIMA(0,1,1), ARIMA(0,1,3), ARIMA(0,1,0), ARIMA(0,1,1), and ARIMA(0,1,1), respectively; chosen among the candidates ARIMA(p,1,q), $p, q \in \{0, \dots, 6\}$, based on Akaike's information criterion.

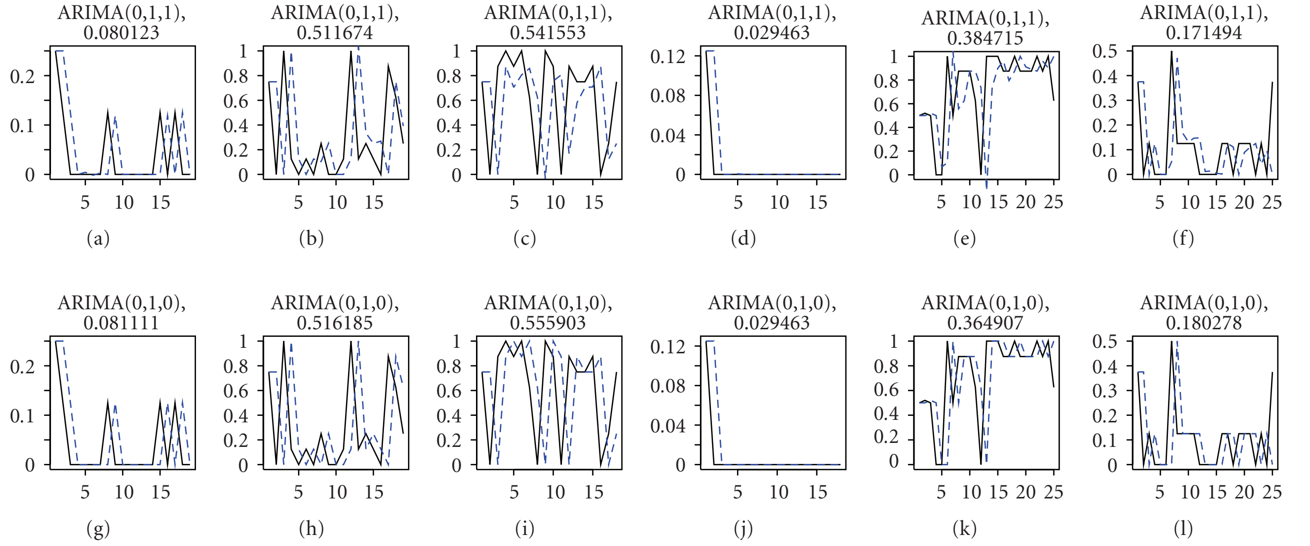
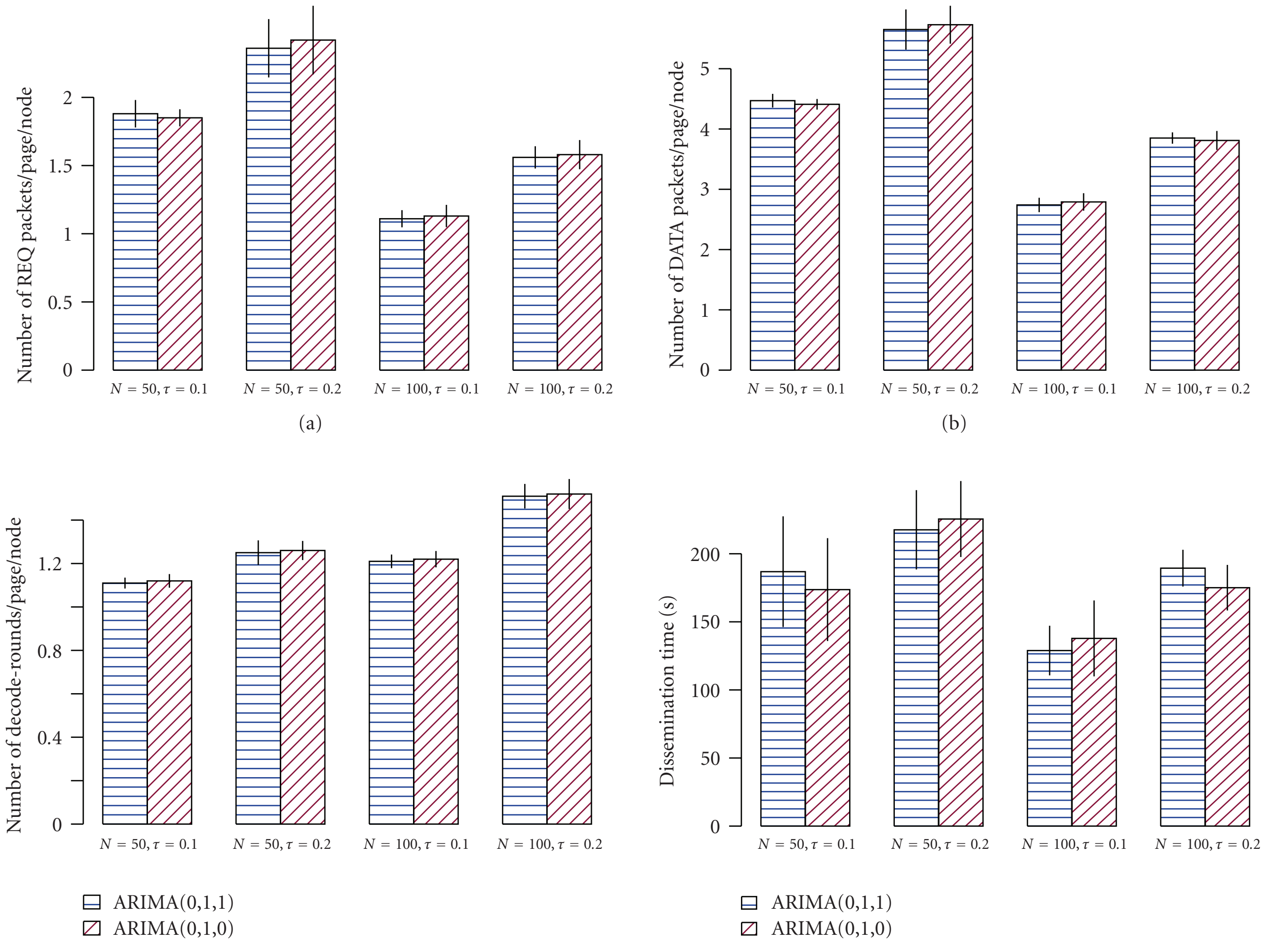


Figure 10: Forecast results of the six randomly sampled packet ratio processes from Figure 9. The top and bottom rows are based on models ARIMA(0,1,1) and ARIMA(0,1,0), respectively. Solid lines represent the original time series whereas dashed lines represent the results of *real-time* forecasting with a forecast horizon of 1. Root mean square errors between the actual time series and the forecast time series are given above each graph.

Table 4: Comparison of Sreluge to Rateless Deluge ("R.D.") when $\tau = 0\%$. Figures are mean values.

	50 nodes		100 nodes	
	R.D.	Sreluge	R.D.	Sreluge
REQ/page/node	1.47	1.47	0.82	0.80
Data/page/node	3.59	3.56	2.01	1.97
Decoding/page/node	1.00	1.01	1.00	1.00
Dissemination time	128.17	123.75	79.08	73.83



(a)

(b)

(c)

(d)

Figure 11: Comparison of forecast models for packet ratios. $f_T = 0.5$.

(iv) Dissemination Time . This metric considers the amount of time in seconds required for the disseminated firmware to reach all the nodes in the network, but does not take into account the time required for the nodes to reboot themselves using the new firmware. Note that the timing results here are only rough estimates as TOSSIM does not simulate computation delay.

For the simulations, key predistribution and distribution of index packets are not implemented; 10-byte-long hashes derived from the pages are hard-coded. Assuming both Rateless Deluge and Sreluge implement the same key predistribution scheme and index packet scheme, then the overhead for both Rateless Deluge and Sreluge in this regard would be the same, and the comparison between the two protocols given here would not be far off.

When $\tau = 0\%$, Sreluge behaves like Rateless Deluge, as evidenced by Table 4. This confirms that Sreluge does not incur unnecessary overhead when there is no attack. Table 4 also shows that higher network density quickens dissemination in a pollution-free network. In the ensuing subsections, results will be given

(1) to compare forecast methods of the packet ratio, assuming Basic polluters;

(2) to compare different values of f_T, assuming Basic polluters;

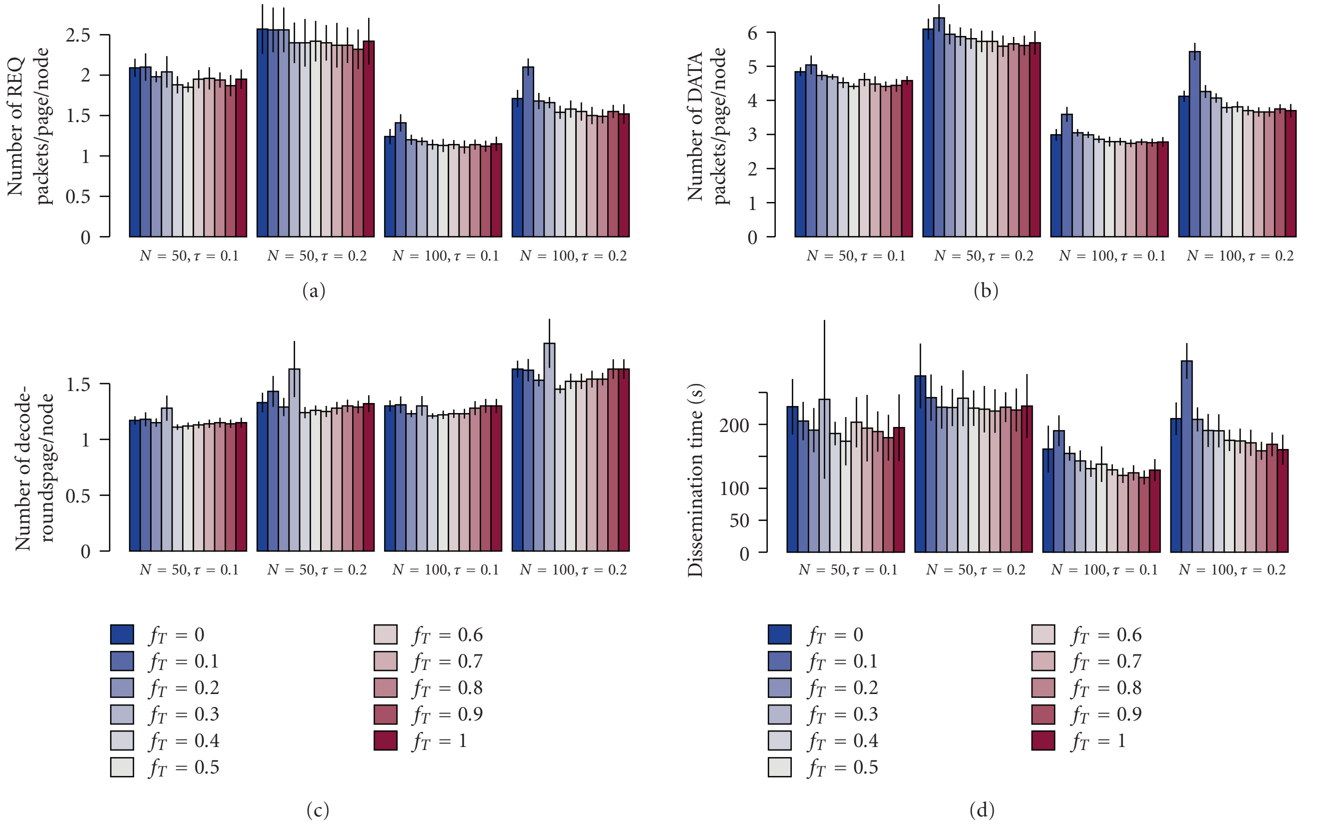
(3) to compare Sreluge with a prescient countermeasure called Prescient Sreluge, assuming Basic polluters;

(4) to compare the effects of Basic and Aggressive polluters on the performance of Sreluge.

Experimental results using Crossbow IRIS motes will also be given.

8.1. Comparison of Forecast Methods of the Packet Ratio.

In procedure `newPrimeSuspect`, packet ratios need to be forecast. By simulations, we find that first-order differenced packet ratio processes are mostly stationary and may well be modelled using moving average processes. For example, among the six randomly sampled packet ratio processes in Figure 9, *after first-order differencing*, the best model for half of the processes is found to be the first-order



(a)

(b)

(c)

(d)

FIGURE 12: Comparison of different values of f_T. ARIMA(0,1,0) is used as the forecast model for packet ratios.

moving average, MA(1). For the ensuing discussion, we employ the standard notation ARIMA(p,d,q) to denote the *autoregressive integrated moving average* time series, $f(t)$, defined by the following [61]:

$$\phi_p(B)(1-B)^d f(t) = \mu + \theta_q(B)\epsilon_t \tag{14}$$

where

$$\begin{aligned} B^i f(t) &= f(t-i) \quad \forall i = 0,1,\dots,t, \\ \phi_p(B) &= 1 - \phi_1 B - \cdots - \phi_p B^p, \\ \theta_q(B) &= 1 + \theta_1 B + \cdots + \theta_q B^q, \\ \epsilon_t &\sim N(0,\sigma^2). \end{aligned} \tag{15}$$

B is called the lag operator and μ is called the deterministic trend term when $d \geq 1$. Using this standard notation, we say that for half of the randomly sampled packet ratio processes in Figure 9, the best model is found to be ARIMA(0,1,1). While it is inconclusive whether ARIMA(0,1,1) is the best model for the majority of packet ratio processes besides these six, it is beyond the capability of standard sensor nodes to perform real-time model selection as well as parameter estimation for high-order ARIMA models.

Reusing the packet ratio processes from Figure 9, we further compare ARIMA(0,1,1) to ARIMA(0,1,0) in terms of their accuracy in forecasting packet ratios in real-time.

When the model is ARIMA(0,1,1), we use the *method of moments* [62, page 125] for parameter estimation and the *innovations algorithm* [62, page 115] for forecasting. We use these algorithms instead of the higher-accuracy Kalman filter because the latter is much more resource-intensive.

When the model is ARIMA(0,1,0), we first write down the equation for the model:

$$f(t) - f(t-1) = \mu + \epsilon_t, \tag{16}$$

which is essentially a first-order autoregressive process, AR(1). We then apply the *Wiener-Kolmogorov prediction formula* [61, page 80] for AR(1):

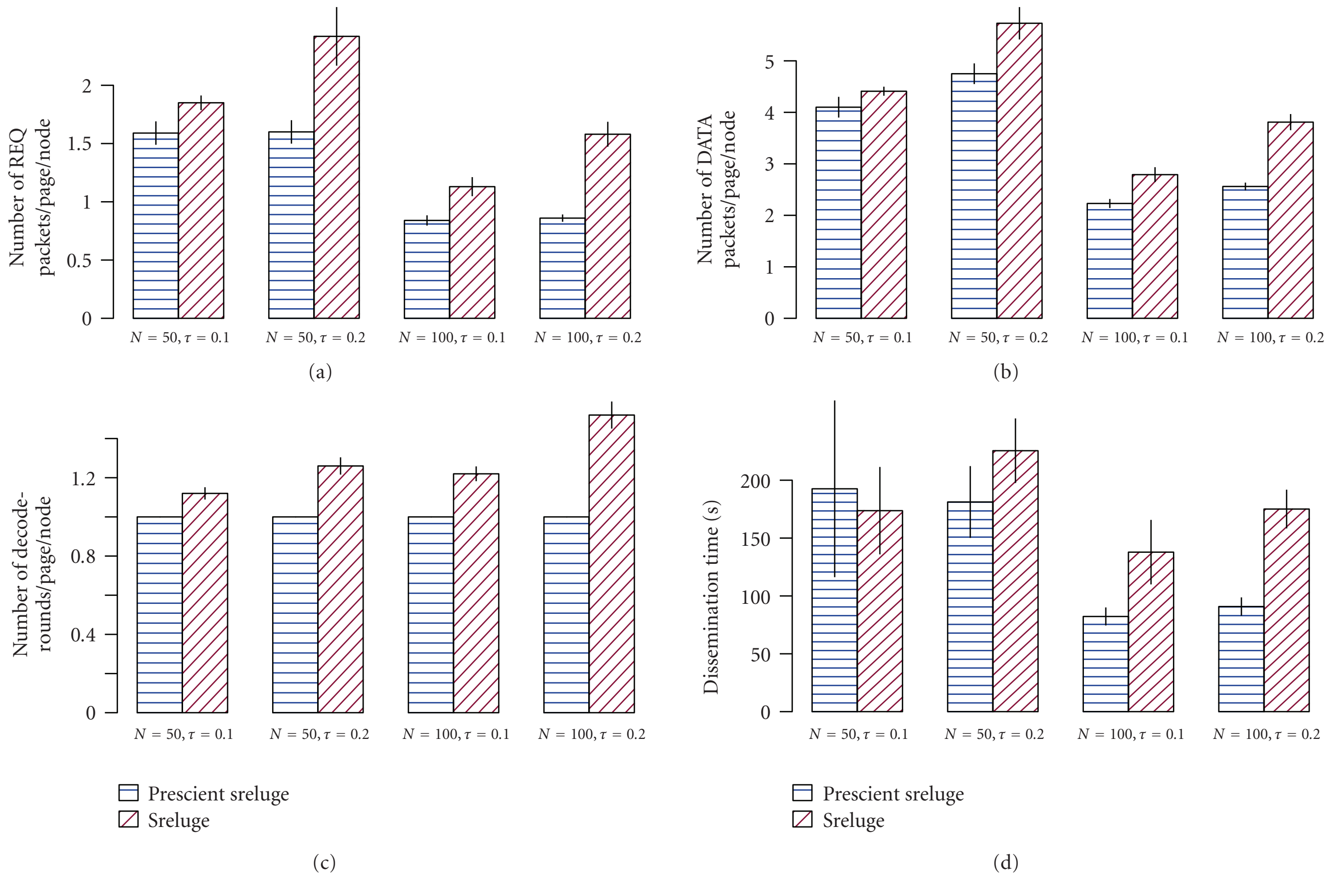
$$E[f(t+s) \mid f(t), f(t-1), \dots] = \mu + \phi_1^s [f(t) - \mu]. \tag{17}$$

Substituting $\phi_1 = 1$ as implied by (16), and $s = 1$ for a forecast horizon of 1, in (17), we have

$$E[f(t+1) \mid f(t), f(t-1), \dots] = f(t) \tag{18}$$

which means the forecast for the next sample is simply the current sample. We must note that the Wiener-Kolmogorov prediction formula is based on an infinite number of observations, so it is only used for asymptotic approximation.

The results in Figure 10, obtained offline using R [63], show that in all cases, forecasting using ARIMA(0,1,0) is only slightly worse than using ARIMA(0,1,1). While the results in Figure 10 are obtained offline, the results in Figure 11 are obtained online, which show that the slight degradation in forecast accuracy of ARIMA(0,1,0) does not result in



(a) (b) (c) (d)

Figure 13: Comparison of Sreluge with Prescient Sreluge. For regular Sreluge, ARIMA(0,1,0) is used as the forecast model for packet ratios; $f_T = 0.5$.

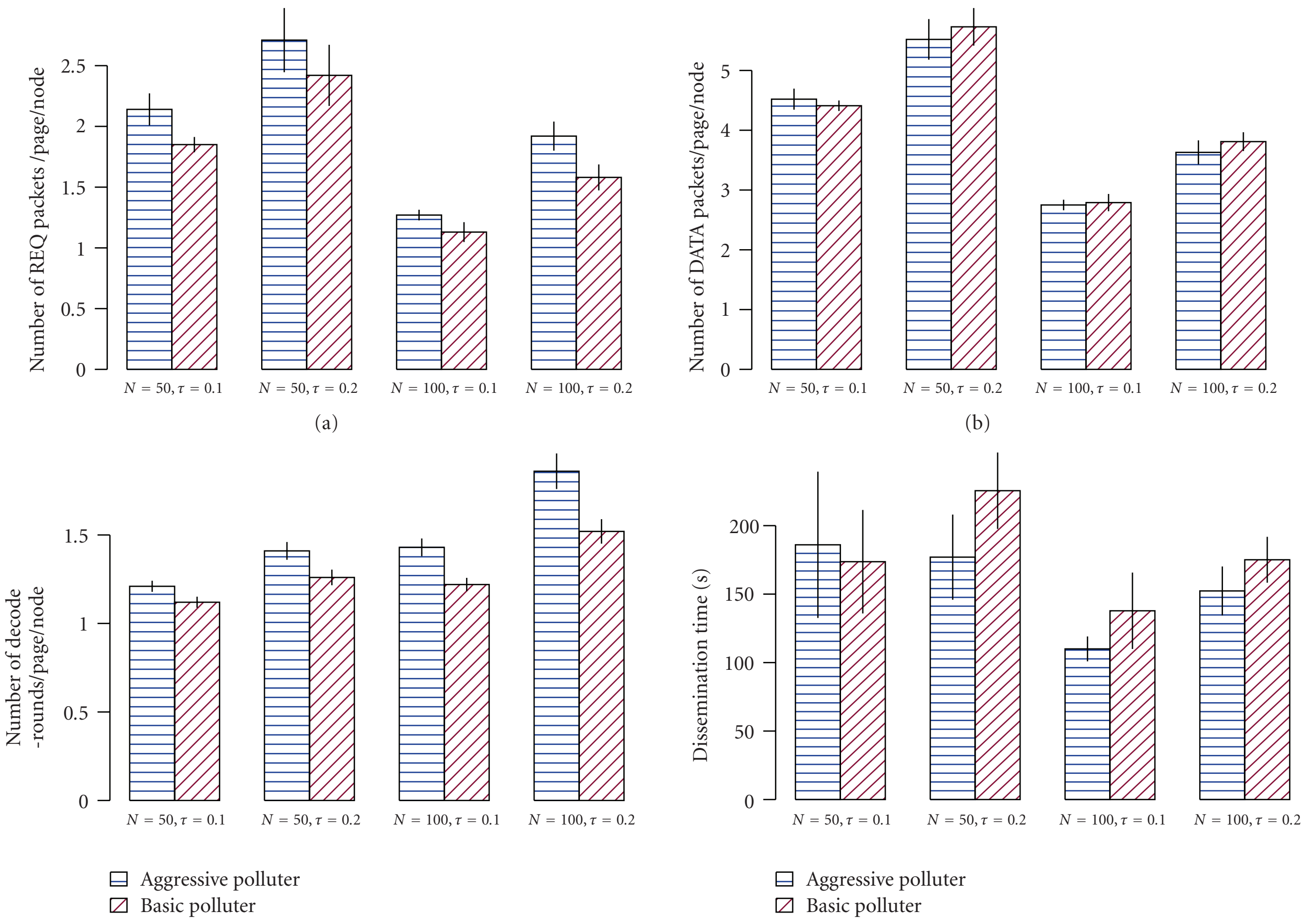
any practical difference in the four performance metrics. Consequently, we adopt the simpler ARIMA(0,1,0) as the model, and in the interest of resource conservation, $\hat{f}(i+1) = f(i)$ as the forecast formula.

8.2. Comparison of Different Values of f_T. In procedure `newPrimeSuspect`, f_T is used as the threshold for choosing a prime suspect. A suitable value is chosen via simulations. For the simulations, 11 values ranging from 0.0 to 1.0 are used. When $f_T = 0.0$, the procedure `newPrimeSuspect` always finds a prime suspect. When $f_T = 1.0$, the procedure almost never finds a prime suspect. The results in Figure 12 show that the number of REQ/data packets and the dissemination time generally manifest a downward trend against f_T whereas the number of decode-rounds generally increases with f_T. As such, $f_T = 0.5$ presents a reasonable tradeoff.

8.3. Comparison with Prescient Sreluge. Sreluge is a reactive countermeasure that tries to identify polluters. If the good nodes knew in advance which neighbors of theirs are polluters, then the overhead could be drastically reduced. With this prescience, the nodes will drop all ADV and data from the polluters. We call this prescient countermeasure *Prescient Sreluge*. This is not a realistic protocol, but as a comparison to Sreluge, it sheds light on how close Sreluge is to a solution with prior information. Figure 13 shows that, in the worst case (100 nodes with 20% polluters), Sreluge requires 100% more REQ packets, 50% more data packets, 50% more decode-rounds, and 100% more dissemination time.

8.4. Comparison of Basic and Aggressive Polluters. Figure 14 shows that Aggressive polluters make Sreluge send more REQ packets and perform more decoding. However, Aggressive polluters neither make Sreluge send more data packets, nor make Sreluge spend more time on dissemination. This surprising outcome can be explained by the fact that as the Aggressive polluters start polluting data earlier than the Basic polluters do, they tend to be blacklisted earlier than the Basic polluters do as well. With pollution blocked earlier, the session can finish earlier. Since the Aggressive polluter does not cause more "damage" than the Basic polluter does for the amount of effort that is required of the attacker, the Basic polluter should be considered a more prevalent threat.

Consistent with intuition, Figures 11–14 show that a higher percentage of polluters increases all four metrics. On the other hand, for the same percentage of polluters, higher network density results in less REQ/data packets, shorter dissemination time, but more decode-rounds.



(a)

(b)

(c)

(d)

Figure 14: Comparison of the effects of Basic and Aggressive polluters on the performance of Sreluge. ARIMA(0,1,0) is used as the forecast model for packet ratios. $f_T = 0.5$.

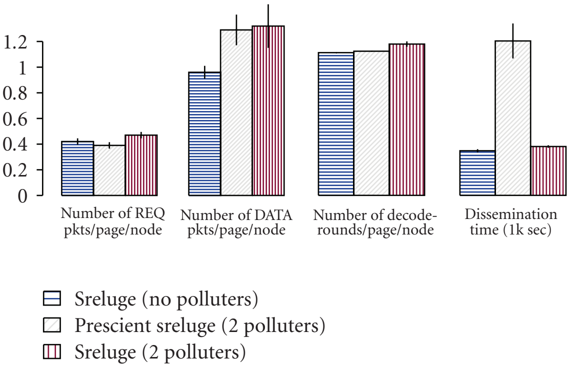


Figure 15: Experimental results comparing Sreluge to Prescient Sreluge in the presence of polluters, and to Sreluge in the absence of polluters

8.5. Experimental Results. The objective of the experiments is to compare the performance of Sreluge under attack, and Sreluge not under attack as well as the performance of Prescient Sreluge under attack. For the experiments, 11 Crossbow IRIS motes running Sreluge on top of the CVS version of TinyOS 2.x are used—one of which serves as the base station, five of them are placed one hop away from the base station, and the remaining five are placed two hops from the base station. All nodes in the same hop distance from the base station are interconnected. Packet length is set to 33 and the transmission power is set to the minimum to minimize interference and the area of the network. A TinyOS image of 191 pages is disseminated by the base station. To simulate attacks, two Basic polluters ($\tau = 20\%$, not counting the base station) are placed in the first hop of the network. For each scenario, the experiment is carried out five times. The results are shown in Figure 15. Note that due to practical constraints, this experimental network is much sparser than the simulated networks, and hence the results here are not directly comparable to the simulation results in the previous sections.

Figure 15 shows that the overhead of Sreluge in the face of a 20% attack is insubstantial compared to the case when there is no attack at all. Intuition may suggest that all the performance metrics will be lowest when there is no attack, and highest when there is attack and Sreluge (not

Prescient Sreluge) is used. However, Figure 15 turns out to be conflicting with intuition in terms of the number of REQ packets and in terms of dissemination time. According to observation, polluters tend to saturate the medium with ADV packets when Prescient Sreluge is used, consequently the normal nodes have less opportunity to send REQ packets and it takes a much longer time for the dissemination to finish. During the course of the experiments, it is found that Flash-write failures (`BlockWrite.write` returns `FAIL`) on the IRIS motes are more common than expected, but neither the official Deluge nor the official Rateless Deluge code handles this exception because they are targeted at the TelosB platform. Moreover, interference-induced packet corruption happens more frequently than expected, such that a simple 2-byte CRC on the data link layer is no longer adequate. In view of these exceptions, improvements have been made to Sreluge to ensure Sreluge will only proceed after a successful Flash write, and interference-induced packet corruption is detected using a 4-byte SHA-1 hash (not to be confused with the hashes used for page verification).

On the IRIS platform, Sreluge compiles to 52078 bytes in the Flash memory, but the figure also includes the code and data structures for collecting experimental results. The compiler reports a RAM usage of 3131 bytes. Most of the RAM space is used for maintaining per-neighbour information. This figure is a result of hard-coding the following size limits: number of neighbors ≤ 40, $|\mathcal{S}| \leq 25$, $|\mathcal{B}| \leq 10$, $|\mathcal{A}| \leq 25$, and $|\mathcal{I}| \leq 10$. (recall that TinyOS 2.x deprecates dynamic memory allocation). The exact ROM and RAM figures would also vary slightly with the exact version of TinyOS 2.x used.

9. Conclusion and Future Work

To address the inadequacy of existing schemes for securing network coding-based reprogramming protocols, we propose Sreluge, a secure version of Rateless Deluge that is resistant to pollution attacks. Sreluge features a neighbor classification system and a time series forecasting technique to isolate polluters, and a combinatorial technique to decode packets in the presence of polluters before the isolation is complete. For detecting polluters, Sreluge has zero false negative rate and a negligible false positive rate. TOSSIM simulations show that when 20% of the nodes in a 27-degree, 100-node network are polluters, using Sreluge, dissemination on average takes roughly twice as long to finish as when there is no attack; a node transmits twice as many REQ and data packets, and performs 50% more decoding per page. Experimental results using Crossbow IRIS motes show that the overhead of Sreluge in a 11-node network is insubstantial. Sreluge is equally useful in cases where nonmalicious but faulty nodes “inadvertently” feed corrupted data packets to their neighbors.

The current limitation is that there is no analytical model that describes the negotiation dynamics (exchange of ADV/REQ/data) of Deluge (which Rateless Deluge and Sreluge inherit), and as a result Sreluge resorts to simulation-based study to forecast packet ratios. In fact, due to the lack of this model, most creators of post-Deluge reprogramming protocols rely on implementation results to make comparisons between Deluge and their own protocols. Sreluge currently investigates one prime suspect in one page-round. There are potential savings in bandwidth and energy if multiple prime suspects are investigated within a page-round, given enough memory. More investigation will be carried out in this direction.

Acknowledgments

This work is partly supported by the Australian Research Council under grant DP1095452 and the European Commission under the Contract no. INFSO-ICT-215923 (SENSEI). The authors would like to thank Andrew Hagedorn for sharing his Rateless Deluge source code; Chieh-Jan Liang, Razvan Musaloiu-E., Miklos Maroti, and Philip Levis for their help with TinyOS; Osman Ugus for his feedback. Lastly, the authors thank the anonymous reviewers for their immensely helpful reviews.

References

[1] W. R. Heinzelman, J. Kulik, and H. Balakrishnan, “Adaptive protocols for information dissemination in wireless sensor networks,” in *Proceedings of the 5th Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom ’99)*, pp. 174–185, ACM, 1999.

[2] J. W. Hui and D. Culler, “The dynamic behavior of a data dissemination protocol for network programming at scale,” in *Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems (SenSys ’04)*, pp. 81–94, New York, NY, USA, November 2004.

[3] R. Ahlswede, N. Cai, S.-Y. R. Li, and R. W. Yeung, “Network information flow,” *IEEE Transactions on Information Theory*, vol. 46, no. 4, pp. 1204–1216, 2000.

[4] I.-H. Hou, Y.-E. Tsai, T. F. Abdelzaher, and I. Gupta, “AdapCode: adaptive network coding for code updates in wireless sensor networks,” in *Proceedings of the 27th IEEE Conference on Computer Communications (INFOCOM ’08)*, pp. 2189–2197, 2008.

[5] A. Hagedorn, D. Starobinski, and A. Trachtenberg, “Rateless Deluge: over-the-air programming of wireless sensor networks using random linear codes,” in *Proceedings of the 7th International Conference on Information Processing in Sensor Networks (IPSN ’08)*, pp. 457–466, IEEE Computer Society, 2008.

[6] L. Buttyán, L. Czap, and I. Vajda, “Securing coding based distributed storage in wireless sensor networks,” in *Proceedings of the 5th IEEE International Conference on Mobile Ad-Hoc and Sensor Systems (MASS ’08)*, pp. 821–827, October 2008.

[7] P. E. Lanigan, R. Gandhi, and P. Narasimhan, “Sluice: secure dissemination of code updates in sensor networks,” in *Proceedings of the 26th IEEE International Conference on Distributed Computing Systems (ICDCS ’06)*, pp. 53–63, IEEE Computer Society, 2006.

[8] P. K. Dutta, J. W. Hui, D. C. Chu, and D. E. Culler, “Securing the deluge network programming system,” in *Proceedings of the 5th International Conference on Information Processing in Sensor Networks (IPSN ’06)*, pp. 326–333, ACM, April 2006.

[9] J. Deng, R. Han, and S. Mishra, “Secure code distribution in dynamically programmable wireless sensor networks,” in

Proceedings of the 5th International Conference on Information Processing in Sensor Networks (IPSN '06), pp. 292–300, April 2006.

[10] D. K. Nilsson, U. Lindqvist, T. Roosta, and A. Valdes, "Key management and secure software updates in wireless process control environments," in *Proceedings of the 1st ACM Conference on Wireless Network Security (WiSec'08)*, pp. 100–108, ACM, 2008.

[11] S. Hyun, P. Ning, A. Liu, and W. Du, "Seluge: secure and DoS-resistant code dissemination in wireless sensor networks," in *Proceedings of the 7th International Conference on Information Processing in Sensor Networks (IPSN '08)*, pp. 445–456, IEEE Computer Socity, April 2008.

[12] A. Liu, P. Ning, and C. Wang, "Lightweight remote image management for secure code dissemination in wireless sensor networks," in *Proceedings of the 28th IEEE Conference on Computer Communications (INFOCOM '09)*, pp. 1242–1250, April 2009.

[13] A. Liu and P. Ning, "TinyECC: a configurable library for elliptic curve cryptography in wireless sensor networks," in *Proceedings of the International Conference on Information Processing in Sensor Networks (IPSN '08)*, pp. 245–256, IEEE Computer Society, 2008.

[14] P. Ning, A. Liu, and W. Du, "Mitigating DoS attacks against broadcast authentication in wireless sensor networks," *ACM Transactions on Sensor Networks*, vol. 4, no. 1, article 1, 2008.

[15] H. Tan, D. Ostry, J. Zic, and S. Jha, "A confidential and DoS-resistant multi-hop code dissemination protocol for Wireless Sensor Networks," in *Proceedings of the 2nd ACM Conference on Wireless Network Security (WiSec '09)*, pp. 245–252, March 2009.

[16] L. Wang and S. S. Kulkarni, "Authentication in reprogramming of sensor networks for mote class adversaries," in *Proceedings of the 21st International Parallel and Distributed Processing Symposium (IPDPS '07)*, pp. 1–8, March 2007.

[17] I. Krontiris and T. Dimitriou, "Authenticated in-network programming for wireless sensor networks," in *Ad Hoc Mobile Wireless Networks*, vol. 4104 of *Lecture Notes in Computer Science*, pp. 390–403, 2006.

[18] O. Ugus, D. Westhoff, and J.-M. Bohli, "A ROM-friendly secure code update mechanism for WSNs using a stateful-verifier T-time signature scheme," in *Proceedings of the 2nd ACM Conference on Wireless Network Security (WiSec '09)*, pp. 29–40, 2009.

[19] M. Rossi, N. Bui, G. Zanca, L. Stabellini, R. Crepaldi, and M. Zorzi, "SYNAPSE++: code dissemination in wireless sensor networks using fountain codes," *IEEE Transactions on Mobile Computing*. In press.

[20] H. Tan, S. Jha, D. Ostry, J. Zic, and V. Sivaraman, "Secure multi-hop network programming with multiple one-way key chains," in *Proceedings of the 1st ACM Conference on Wireless Network Security (WiSec '08)*, pp. 183–193, ACM, April 2008.

[21] Y. Zhang, X.-S. Zhou, Y.-M. Ji, Z.-Y. Fang, and L.-F. Wang, "Secure and DoS-resistant network reprogramming in sensor networks based on CPK," in *Proceedings of the 4th International Conference on Wireless Communications, Networking and Mobile Computing (WiCOM '08)*, pp. 1–5, October 2008.

[22] S. Zhu, S. Setia, and S. Jajodia, "LEAP+: efficient security mechanisms for large-scale distributed sensor networks," *ACM Transactions on Sensor Networks*, vol. 2, no. 4, pp. 500–528, 2006.

[23] S. A. Çamtepe and B. Yener, "Combinatorial design of key distribution mechanisms for wireless sensor networks," *IEEE/ACM Transactions on Networking*, vol. 15, no. 2, pp. 346–358, 2007.

[24] D. Sánchez and H. Baldus, "A deterministic pairwise key pre-distribution scheme for mobile sensor networks," in *Proceedings of the 1st International Conference on Security and Privacy for Emerging Areas in Communications Networks (SecureComm '05)*, pp. 277–288, September 2005.

[25] L. Eschenauer and V. D. Gligor, "A key-management scheme for distributed sensor networks," in *Proceedings of the 9th ACM Conference on Computer and Communications Security*, pp. 41–47, ACM Press, November 2002.

[26] W. Du, J. Deng, Y. S. Han, P. K. Varshney, J. Katz, and A. Khalili, "A pairwise key predistribution scheme for wireless sensor networks," *ACM Transactions on Information and System Security*, vol. 8, no. 2, pp. 228–258, 2005.

[27] D. Liu, P. Ning, and R. Li, "Establishing pairwise keys in distributed sensor networks," *ACM Transactions on Information and System Security*, vol. 8, no. 1, pp. 41–77, 2005.

[28] W. Itani, A. Kayssi, and A. Chehab, "PETRA: a secure and energy-efficient software update protocol for severely-constrained network devices," in *Proceedings of the 5th ACM International Symposium on QoS and Security for Wireless and Mobile Networks (Q2SWinet '09)*, pp. 37–43, ACM, 2009.

[29] R. Gennaro and P. Rohatgi, "How to sign digital streams," *Information and Computation*, vol. 165, no. 1, pp. 100–116, 2001.

[30] S. Jaggi, M. Langberg, S. Katti, T. Ho, D. Katabi, and M. Médard, "Resilient network coding in the presence of Byzantine adversaries," in *Proceedings of the 26th IEEE International Conference on Computer Communications (INFOCOM '07)*, pp. 616–624, IEEE, May 2007.

[31] C. K. Ngai and S. Yang, "Deterministic secure error-correcting (SEC) network codes," in *Proceedings of the IEEE Information Theory Workshop (ITW '07)*, pp. 96–101, September 2007.

[32] T. Ho, B. Leong, R. Koetter, M. Médard, M. Effros, and D. R. Karger, "Byzantine modification detection in multicast networks using randomized network coding," in *Proceedings of the IEEE International Symposium on Information Theory (ISIT '04)*, p. 144, IEEE, July 2004.

[33] J. Tan and M. Médard, "Secure network coding with a cost criterion," in *Proceedings of the 4th International Symposium on Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks*, pp. 1–6, IEEE, 2006.

[34] N. Cai and R. W. Yeung, "Secure network coding," in *Proceedings of the IEEE International Symposium on Information Theory*, p. 323, July 2002.

[35] L. Lima, M. Médard, and J. Barros, "Random linear network coding: a free cipher?" in *Proceedings of the IEEE International Symposium on Information Theory*, pp. 546–550, 2007.

[36] D. Charles, K. Jain, and K. Lauter, "Signatures for network coding," *International Journal of Information and Coding Theory*, vol. 1, no. 1, pp. 3–14, 2009.

[37] M. N. Krohn, M. J. Freedman, and D. Mazières, "On-the-fly verification of rateless erasure codes for efficient content distribution," in *Proceedings of the IEEE Symposium on Security and Privacy*, pp. 226–240, 2004.

[38] S. Agrawal and D. Boneh, "Homomorphic MACs: MAC-based integrity for network coding," in *Proceedings of the 7th International Conference on Applied Cryptography and Network Security*, vol. 5536 of *Lecture Notes in Computer Science*, pp. 292–305, Springer, 2009.

[39] Z. Yu, Y. Wei, B. Ramkumar, and Y. Guan, "An efficient scheme for securing XOR network coding against pollution attacks," in *Proceedings of the 28th Conference on Computer Communications (INFOCOM '09)*, pp. 406–414, April 2009.

[40] J. Dong, R. Curtmola, and C. Nita-Rotaru, "Practical defenses against pollution attacks in intra-flow network coding for wireless mesh networks," in *Proceedings of the 2nd ACM Conference on Wireless Network Security (WiSec '09)*, pp. 111–122, March 2009.

[41] J. Dong, R. Curtmola, and C. Nita-Rotaru, "Secure network coding for wireless mesh networks: threats, challenges, and directions," *Computer Communications*, vol. 32, no. 17, pp. 1790–1801, 2009.

[42] A. Perrig, R. Szewczyk, V. Wen, D. Culler, and J. D. Tygar, "SPINS: security protocols for sensor networks," in *Proceedings of the 7th Annual International Conference on Mobile Computing and Networking*, pp. 189–199, ACM, July 2001.

[43] J.-M. Bohli, A. Hessler, O. Ugus, and D. Westhoff, "Security enhanced multi-hop over the air reprogramming with fountain codes," in *Proceedings of the Conference on Local Computer Networks (LCN '09)*, pp. 850–857, 2009.

[44] M. Rossis, G. Zancas, L. Stabellini, R. Crepaldi, A. F. Harris III, and M. Zorzi, "SYNAPSE: a network reprogramming protocol for wireless sensor networks using fountain codes," in *Proceedings of the 5th Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks (SECON '08)*, pp. 188–196, June 2008.

[45] M. Luby, "LT codes," in *Proceedings of the 34rd Annual IEEE Symposium on Foundations of Computer Science*, pp. 271–282, IEEE Computer Society, November 2002.

[46] J. Dong, R. Curtmola, R. Sethi, and C. Nita-Rotaru, "Toward secure network coding in wireless networks: threats and challenges," in *Proceedings of the 4th IEEE Workshop on Secure Network Protocols (NPSec '08)*, pp. 33–38, October 2008.

[47] S. Ganeriwal, L. K. Balzano, and M. B. Srivastava, "Reputation-based framework for high integrity sensor networks," *ACM Transactions on Sensor Networks*, vol. 4, no. 3, article 15, pp. 1–37, 2008.

[48] Y. L. Sun, Z. Han, W. Yu, and K. J. R. Liu, "A trust evaluation framework in distributed networks: vulnerability analysis and defense against attacks," in *Proceedings of the 25th IEEE International Conference on Computer Communications (INFOCOM '06)*, pp. 1–13, 2006.

[49] J. Lopez, R. Roman, I. Agudo, and C. Fernandez-Gago, "Trust management systems for wireless sensor networks: best practices," *Computer Communications*, vol. 33, no. 9, pp. 1086–1093, 2010.

[50] Y. W. Law, M. Palaniswami, L. V. Hoesel, J. Doumen, P. Hartel, and P. Havinga, "Energy-efficient link-layer jamming attacks against wireless sensor network MAC protocols," *ACM Transactions on Sensor Networks*, vol. 5, no. 1, article 6, 2009.

[51] R. C. Merkle, "A certified digital signature," in *Proceedings on Advances in Cryptology (CRYPTO '89)*, pp. 218–238, Springer, 1989.

[52] M. Bellare, J. Kilian, and P. Rogaway, "Security of the cipher block chaining message authentication code," *Journal of Computer and System Sciences*, vol. 61, no. 3, pp. 362–399, 2000.

[53] J. Newsome, E. Shi, D. Song, and A. Perrig, "The Sybil attack in sensor networks: analysis & defenses," in *Proceedings of the 3rd International Symposium on Information Processing in Sensor Networks (IPSN '04)*, pp. 259–268, ACM, 2004.

[54] J. Katz, *Digital Signatures*, Springer, Berlin, Germany, 2010.

[55] D. R. L. Brown, "Generic groups, collision resistance, and ECDSA," *Designs, Codes, and Cryptography*, vol. 35, no. 1, pp. 119–152, 2005.

[56] N. Koblitz, and A. Menezes, "Another look at generic groups," *Advances in Mathematics of Communications*, vol. 1, no. 1, p. 13, 2007.

[57] Certicom Research, *Standards for Efficient Cryptography. SEC2: Recommended Elliptic Curve Domain Parameters*, 1st edition, 2000.

[58] L. C. C. García, "On the security and the efficiency of the Merkle signature scheme," Cryptology ePrint Archive: Report 2005/192, 2005.

[59] C. Metcalf, *TOSSIM Live: towards a testbed in a thread*, M.S. thesis, Colorado School of Mines, Golden, Colo, USA, 2007.

[60] M. Zuniga and B. Krishnamachari, "Analyzing the transitional region in low power wireless links," in *Proceedings of the 1st Annual IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks (SECON '04)*, pp. 517–526, IEEE, October 2004.

[61] J. D. Hamilton, *Time Series Analysis*, Princeton University Press, Princeton, NJ, USA, 1994.

[62] R. H. Shumway and D. S. Soffer, *Time Series Analysis and Its Applications With R Examples*, Springer, Berlin, Germany, 2nd edition, 2006.

[63] R Development Core Team, *R: A Language and Environment for Statistical Computing*, R Foundation for Statistical Computing, Vienna, Austria, 2008.