New Strategies for Revocation
in Ad-Hoc Networks

Tyler Moore, Jolyon Clulow, Shishir Nagaraja, and Ross Anderson

Computer Laboratory, University of Cambridge
15 JJ Thomson Avenue, Cambridge CB3 0FD, United Kingdom
`firstname.lastname@cl.cam.ac.uk`

Abstract. Responding to misbehavior in ad-hoc and sensor networks is difficult. We propose new techniques for deciding when to remove nodes in a decentralized manner. Rather than blackballing nodes that misbehave, a more efficient approach turns out to be *reelection* – requiring nodes to secure a majority or plurality of approval from their neighbors at regular intervals. This can be implemented in a standard model of voting in which the nodes form a club, or in a lightweight scheme where each node periodically broadcasts a 'buddy list' of neighbors it trusts. This allows much greater flexibility of trust strategies than a predetermined voting mechanism. We then consider an even more radical strategy still – *suicide attacks* – in which a node on perceiving another node to be misbehaving simply declares both of them to be dead. Other nodes thereafter ignore them both. Suicide attacks, found in a number of contexts in nature from bees to helper T-cells, turn out to be more efficient still for an interesting range of system parameters.

Keywords: credential revocation, sensor networks, key management.

1 Introduction

The last ten years have seen the invention and deployment of a range of systems which organize themselves out of a collection of nodes in order to perform some task. Peer-to-peer systems emerged in the late 1990s, first as a means of resisting censorship on the Internet [1] and then as a mechanism for file-sharing. Communications technologies such as WiFi, Bluetooth and Homeplug support short-range networking of disparate devices in home and office environments, and may allow larger networks to be assembled opportunistically. Sensor networks then came along – networks assembled from large numbers of low-cost nodes that could be scattered into an area of interest to perform some task such as surveillance or environmental monitoring. We describe such communications strategies generically as 'ad-hoc networking'.

One consequence for these new technologies is that management by central authority is being discarded in favor of decentralized mechanisms to improve efficiency and robustness. Here, tasks are distributed amongst member nodes which cooperate to provide services and reach decisions. Another key feature

of wireless ad-hoc networks is their support for mobile devices. Devices move while remaining connected to the network, breaking links with old neighbors and establishing fresh links with new devices. Finally, resource constraints limit the tools available to the protocol designer: at most a minority of nodes have the capability to create digital signatures or store large amounts of data; symmetric cryptography is preferred for establishing and maintaining key material.

There are various threats to ad-hoc networks, of which the most interesting and important is probably *node subversion*. A node in a military sensor network may be reverse-engineered by the enemy, and replaced by a malicious node that knows its key material and can thus impersonate it. A participant in a peer-to-peer network may be forced to hand over his keys to an enforcement agency. Subverted nodes can perform a number of attacks on the network, for example, decrypting messages, injecting false data and manipulating decentralized operations such as voting. Thus they must be identified and removed quickly.

In this paper, we seek to address this revocation problem. We are primarily concerned with ad-hoc networks where a minority of nodes can be subverted, and with mechanisms whereby a subverted node can be efficiently removed. Existing strategies using certificate revocation lists and certification authorities are inappropriate for the requirements of these new systems. Instead, we propose lightweight revocation mechanisms suited to decentralized and mobile networks.

In Section 2, we review existing work. We then propose new distributed mechanisms for deciding when to remove bad nodes: reelection in Section 3 where nodes cast positive votes rather than negative ones, and suicide attacks in Section 4 where a node unilaterally decides to remove a bad node at the expense of its own participation on the network. In Section 5, we compare the performance and security of each scheme before concluding in Section 6.

2   Background

2.1   System Model

There are four basic events in the life cycle of an ad-hoc network: *pre-deployment*, *initialization*, *operation* and *revocation*. In pre-deployment, the network owner (if one exists) programs nodes with key material. For instance, symmetric keys are often pre-loaded onto sensor nodes [2,3,4,5]. Nodes are then deployed and initialized; during this phase they establish keys with their neighbors. When nodes are mobile, this is a continuing process rather than one-off. At any stage, one or more nodes may find another misbehaving, and this may prompt a decision to remove the bad node from the system.

Good nodes adhere to their programmed strategy including algorithms for routing and revocation. The attacker can compromise a small minority of nodes. A bad node can communicate with any other node, good or bad. Bad nodes may have access to the keys of all other bad nodes, whom they can therefore impersonate if they wish. They do not execute the authorized software and thus do not necessarily follow protocols to identify misbehavior, revoke other bad nodes, vote honestly or delete keys shared with revoked nodes.

We consider the two main threat models in the literature. Under the *conservative* model, a global, active adversary is present from the start of deployment; networks thus require a pre-deployment phase where nodes are assigned keys. Under the *relaxed* threat model, the opponent can monitor at most a small minority of communications during the initialization phase [6,7]. This means that no, or fewer, keys need to be pre-loaded; instead, nodes set up link keys with neighbors immediately after deployment. In each case, however, the focus is on the ease and cost of security maintenance after deployment.

Another important threat is the *Sybil attack* [8,9], in which an opponent denies service by causing large numbers of malicious nodes to join the network. A Sybil variant is node replication [10], where many copies of a subverted node are introduced. These stratagems have been used to attack peer-to-peer systems. We focus on networks in which the Sybil attack can be contained, perhaps by having a cost of entry, or perhaps by using initially trustworthy nodes whose subversion requires a finite effort of reverse engineering.

2.2   Dealing with Bad Nodes

Three stages are required to revoke a bad node: detecting misbehavior, deciding whether to revoke, and implementing punishment. While each stage is important, in this paper we focus on the decision whether to revoke an accused node.

Deciding when to remove a node is complicated by two factors. First, detection mechanisms rarely produce evidence that is universally non-repudiable. When such mechanisms do exist (e.g., geographic packet leashes [11] for detecting wormholes and node replication detection in sensor networks [9]), they require extensive use of costly signed messages. Furthermore, situations where a bad node is forced into self-incrimination are limited. It can be hard to get hanging evidence against a node that drops occasional messages because messages vanish for many reasons unconnected with malice. More typically, evidence is gathered which is non-repudiable to a single party. For example, a message authentication code (MAC) generated with a key shared between two nodes guarantees authenticity to the other node. Detection mechanisms of this type include temporal packet leashes [11], Sybil attack [8] detection by querying for possessed keys [9] and distance-bounding protocols [12,13,14]. Still other mechanisms rely on evidence that is entirely repudiable (e.g., the wireless monitoring scheme Watchdog [15] where nodes promiscuously listen to their neighbors' routing actions). Repudiable evidence enables bad nodes to falsely accuse good nodes. Hence, it would be foolish to design a simple decision mechanism that revokes any node accused of misbehavior following a single unsubstantiated claim of impropriety.

The second factor hindering the design of decision mechanisms is that untrusted nodes, not central authorities, are often in the best position to detect misbehavior. If node A accuses node B of making inconsistent statements about its location and B denies making them, a trusted base station can only determine one of them is misbehaving. Hence a distributed decision mechanism is required, and existing proposals for collective decision-making have been voting-based. Threshold voting is a natural choice to implement revocation as it conceptually

distributes the decision-making process while taking into account the observations of others. Once the number of votes cast exceeds the specified threshold, then the target node is deemed to be malicious and revoked from the network. Any such *blackballing* scheme must deal with a number of key issues: which nodes are eligible to vote, how individual votes are verified, how votes are tallied and how the outcome of a vote is verified. (In the absence of global non-repudiation, the Byzantine generals problem means that, in general, we need a majority of $\frac{2}{3} + \epsilon$ rather than $\frac{1}{2} + \epsilon$ of good nodes; we are concerned in this paper with applications in which the proportion of wrongdoers is much less than $\frac{1}{3}$.)

Once a decision is reached, the bad node is punished. Typically, bad nodes are kept from interacting with good nodes by instructing every node to delete all keys shared with the bad node [2,3,6]. Alternatively, nodes could be implicitly removed by routing around the bad node [15] or by maintaining a blacklist.

2.3   Existing Decision Mechanisms

In [3], Chan, Perrig and Song propose a distributed revocation mechanism for sensor networks using the random pairwise key-predistribution scheme, where nodes sharing a pre-assigned pairwise key can vote to remove a node. Their scheme is extended and generalized in [16]. Here, each node B that shares a pairwise key with A is assigned to the set of *participants of A*, V_A. While the average number of voting members v is significantly larger than the number of neighbors in direct communication range, tying voting eligibility to key predistribution avoids the difficulties in accurately determining neighbors post-deployment.

Every node A is assigned a unique revocation secret rev_A, which is divided into secret shares and given to every $B \in V_A$ along with an authentication value for the revocation secret, $h^2(\text{rev}_A)$. Nodes vote for another's removal by revealing their share. If enough shares are revealed, then rev_A can be reconstructed and the hash $h(\text{rev}_A)$ is broadcast throughout the network. Every node $B \in V_A$ deletes its key shared with A upon verifying the broadcast. Only nodes eligible to vote against a revoked node are loaded with the authentication value for the revocation secret. Any path keys established between a non-voting node and a revoked node are not removed since the non-voting node cannot verify the revocation secret. Thus, authentication values for revocation secrets should be loaded onto every node; [17] describes an efficient $1 + v \log n$ solution.

One problem with voting by revealing secret shares is that cast votes are permanent; a slow trickle of votes against a node over its lifetime is equivalent to a burst in a short period. To avoid stale votes, Chan et al. create T revocation sessions each with a unique revocation secret $\text{rev}_{A,i}, i \in \{1, \ldots, T\}$ and associated shares $\text{rev}_{A,i,B}$; thus a revealed share only counts as a vote for a single period i.

To recap, each node A is loaded with information to do the following:

1. **Vote against each node** $B \in V_A$: Secret share $\text{rev}_{B,i,A} \forall B \in V_A, i \in \{1, \ldots, T\}$) (storage cost vT)
2. **Prove to all** $B \in V_A$ **that vote is valid:** $\log v$ path-authentication values for each vote $\text{rev}_{B,i,A}$ (storage cost $vT \log v$)

3. **Verify votes from others:** Merkle tree roots $\forall B \in V_A, i \in \{1, \dots, T\}$ (storage cost vT)
4. **Verify revocation secrets:** Authentication values for each revocation secrets $h^2(\text{rev}_{B,i})\forall B \in N, i \in \{i, \dots, T\}$ (storage cost nT)

However, voting schemes are slow, expensive and prone to manipulation. They are often susceptible to false accusations, collusive attackers and Sybil attacks; they can result in a delayed attack response between a node starting to misbehave and a revocation order being issued; they do not cope well with node mobility and churn; they may require that at least some nodes can do public-key cryptography; and they impose high storage and communications overhead.

In Sections 3 & 4, we propose new decision mechanisms with the aim of improving security and performance. We use Chan et al.'s blackballing decision mechanism as a basis for comparison.

3   Reelection

Existing proposals for decision and punishment require action by the honest members of the network to remove misbehaving nodes. For example, all nodes must follow the voting, blacklisting and key removal procedures to prevent a malicious node from rejoining the network. This represents a significant computational and communications burden shared by all honest nodes and shirked by malicious ones. In contrast, we propose a mechanism that turns the computational liability on its head by requiring additional effort for honest nodes to continue participating on the network but no effort to remove malicious devices.

We propose a system where a node, on joining the network and periodically thereafter, must demonstrate that it is still authorized to be on the network. Revocation becomes preventing a bad node from renewing its membership. Conceptually, this corresponds to a voting scheme with positive votes instead of negative ones: good nodes reelect each other to the club once in each time period.

We first present a robust protocol for remaining on the network using threshold-secret-sharing mechanisms. Since threshold schemes can be too expensive for peanut processors, we then propose a lightweight reelection mechanism using hash operations exclusively.

3.1   Reelection for Semi-capable Devices

We define a *network access token* $\text{access}_{A,i}$ that allows node A onto the network during time period $i \in \{1, \dots, T\}$. A must present the token $\text{access}_{A,i}$ to its neighbors to continue interacting with them. Tokens are created using a hash chain where $\text{access}_{A,i-1} = h(\text{access}_{A,i})$, for $i = 1, \dots, T$. The end-of-chain authentication value $\text{access}_{A,0}$ is distributed to every voting member $B \in V_A$, which can authenticate $\text{access}_{A,i}$ for time period i by verifying that $\text{access}_{A,0} = h^{(i)}(\text{access}_{A,i})$.

Each token $\text{access}_{A,i}$ is divided into v shares using a (k, v) threshold-secret-sharing scheme. The shares are distributed to the voting members $B \in V_A$. In

particular, B is assigned shares $\text{access}_{A,i,B}$ for each $i = 1, \dots, T$. Responsibility for reconstructing tokens rests with A, which asks its voting members for their shares. So B casting $\text{access}_{A,i,B}$ is an affirmation of A's honesty rather than a claim of impropriety. Hence, the threshold of votes k may be larger than for black-balling, as more positive votes are required than negative ones. Note that if the voting members are those pre-assigned a pairwise key (as in Chan et al.'s black-balling scheme),then nodes should delete any voting shares for non-neighbors following neighbor discovery. Alternatively, we could reduce the average number of voting members v by choosing the voting set upon deployment.

Nodes must store additional information to verify transmitted votes and tokens. To verify received votes, node A can store a hash of each share $h(\text{access}_{A,i,B})$ for each $B \in V_A$ and $i = 1, \dots, T$. To authenticate reconstructed tokens, the owner creates a hash tree where the leaf pre-images are the end-of-chain authentication values $\text{access}_{A,0}$ for each $A \in N$. Each node A stores the tree's root-authentication value, its own end-of-chain authentication value $\text{access}_{A,0}$ and the $\log n$ path-authentication values required to authenticate $\text{access}_{A,0}$.

Here is the reelection protocol for a node during time period i:

1. $A \longrightarrow * : A, i$
2. $B \longrightarrow A : A, B, \text{access}_{A,i,B}$
3. $A \longrightarrow * : A, i, \text{access}_{A,i}, \text{access}_{A,0}, \text{path-authentication values}$
4. $\quad * : \quad$ verify $h^{(i)}(\text{access}_{A,i}) = \text{access}_{A,0}$, verify $\text{access}_{A,0}$

A asks each neighbor B for its share $\text{access}_{A,i,B}$ (step 1). If k voting neighbors cooperate (step 2), then A can reconstruct $\text{access}_{A,i}$, which is then broadcast to A's neighbors (step 3). The neighbors verify $h^{(i)}(\text{access}_{A,i}) = \text{access}_{A,0}$ (step 4).

If node B wishes to vote against A, then it simply deletes the stored shares $\text{access}_{A,i,B}, i = 1, \dots, T$. Once all of the node's neighbors minus k have done so, A can no longer reconstruct tokens. Revocation is final and absolute: even if the adversary subsequently compromises all neighbor nodes, it cannot reconstruct the tokens. The basic method can be trivially modified to temporarily punish A by deleting a subset of the tokens for a number of time periods.

Nodes must wait to delete a revealed neighbor's share until the following round; otherwise an attacker could ask for a neighbor's share so that the intended node does not observe the response. Also, note that step 1 is optional; any node loaded with secret shares for a node A can reveal the share without being asked. Dropping this broadcast step means that nodes must continuously listen for neighbors revealing their shares.

To recap, each a node A is loaded with information to do the following:

1. **Token share for each node** $B \in V_A$**:** Secret share $\text{access}_{B,i,A} \forall B \in V_A$, $i \in \{1, \dots, T\}$ (storage cost vT)
2. **No need to prove token shares to** $B \in V_A$**:** (storage cost 0)
3. **Verify received shares:** Hash values of all token shares for A (storage cost vT)
4. **Prove to all that token is valid:** Root-authentication value and path-authentication values (storage cost $1 + \log n$)

3.2   Lightweight Reelection with Buddy Lists

Reconstructing secret shares can be too demanding for devices with peanut processors. In addition, the effort involved in pre-assigning, swapping and storing vT shares per node may be unattractive in some applications. Also, in some applications we might want to use diverse strategies: risk-averse nodes might shun a neighbor as soon as one of its other neighbors had done so, while more relaxed nodes might continue to do business with any node that was still supported by two of its neighbors. In some applications, one might want a diverse population of risk-averse and risk-loving nodes, so that the network performed well in normal times but still performed acceptably under serious attack. It therefore makes sense to disentangle the voting mechanism as far as possible from the strategy.

We therefore consider a lightweight reelection mechanism that is general enough to support diverse strategies. The central idea is that nodes periodically transmit a *buddy list* of approved neighbors across their local neighborhoods. Since many node neighbors overlap, they can cross-reference received lists to determine whether enough nodes have also approved their buddies. If so, they continue to interact with the nodes during the next time period. The definition of 'enough' is made independently of the protocol mechanism described here.

Approved buddy lists are authenticated using Guy Fawkes-style [18] hash chains: upon deployment, node A distributes a key authentication value $K_{A,0} = h^{(T)}(\text{seed}, A)$ to its neighbors. Buddy lists are signed with a session authentication key $K_{A,i} = h^{(T-i)}(\text{seed}, A)$ during time period i, and $K_{A,i}$ is not revealed until the start of period $i+1$. Here is the protocol:

1. $A \longrightarrow * : k_{i-1}, \text{access}_{A,i}(\text{buddies}) = \langle A, i, \text{buddies}, \text{HMAC}_{K_i}(A, i, \text{buddies}) \rangle$
2. $\quad * : \quad$ Verify $\text{access}_{A,i-1}(\text{buddies})$, delete offending neighbor's keys

Each node A broadcasts a list of approved neighbors $\text{access}_{A,i}(\text{buddies})$, where buddies is a set of approved node identifiers. Notably, no pre-assigned storage or topological information is required, yet buddy lists work even under the conservative threat model. They also support extremely general strategies for maintaining a network's trusted membership. Nodes' risk aversion could change over time, according to news from other nodes, or as part of an evolutionary game; one could even implement dynamic games similar to Conway's game of 'Life'. Separating trust strategies from the underlying protocol, and implementing it using lightweight and purely local mechanisms, is the strength of this option.

4   Suicide Attacks

Doing revocation by blackballing has turned out to be complex and costly. Matters were improved by a move to reelection, whereby each node had to persuade a quorum of its neighbors to support its continued membership at regular intervals, and still further by the buddy-list mechanism. Here we introduce a radical, even simpler and in some ways even cheaper method: suicide.

Decisions are much simpler if a single node can decide. Should a node believe another has misbehaved, then it can carry out punishment. The trouble with

this approach is that a malicious node can falsely accuse legitimate ones; the solution is to make punishment costly. If a node determines another node has cheated, there is no more convincing way to let its neighbors know than to be prepared to die to certify the fact. (The many echoes in pre-modern human societies range from ancient feuds, through medieval trial by combat, to the duels of eighteenth-century Europe.) We discussed suicide as a strategy in [19]; here we describe implementations and ways to mitigate abuse. We present three cases in order of increasing complexity: where a central trusted authority is available; using limited asymmetric cryptography without access to a trusted authority; and using only conventional cryptography without access to a trusted authority.

4.1   Suicide Using a Central Authority

The simplest way to implement suicide attacks is using a central authority such as a base station. Upon detecting a node M engaging in some illegal activity, node A sends a *suicide note* $\text{suicide}_{A,M}$ with the identities of both A and M to the base station authenticated by the pairwise unique key shared between the node and the base station. The base station S confirms that node A is entitled to revoke node M and informs the other nodes in the network by sending either individually authenticated messages or a single TESLA-authenticated message [20]. Note that the decision mechanism remains distributed: it is the nodes, not the authority, who decide when to revoke each other since nodes are better positioned to detect misbehavior than far-away base stations.

4.2   Distributed Suicide Using Signatures

Nodes may not have access to a trusted base station; instead node A broadcasts a signed note $\text{suicide}_{A,M}$ with the identities of both A and M. The other nodes in the network verify the signature and, if correct, revoke both A and M by deleting all keys shared with them and/or adding both identities to a blacklist.

Public key cryptography works when nodes are sufficiently capable. The owner generates a new public-private key pair for each node and signs the public key. The key pair, certificate and owner's public key are stored on the node. When a node issues a suicide note, it broadcasts its public key certificate along with the suicide note for other nodes to verify the public key and suicide note.

In constrained devices, one-time signatures using only pseudo-random functions may be substituted [21]. Each node is pre-loaded with a single private signing key and the associated public key: this key might be certified by the network owner, or a hash of the key might be the device's name, depending on the deployment model and computational constraints. Nodes verifying a signed suicide note must be able to authenticate the public key. Thus the owner constructs a hash tree with the public keys as leaves suitably ordered to tie a node's identity to its position in the tree. Each node stores the root-authentication value and the $\log n$ path-authentication values required to verify their own public key. The path-authentication values are subsequently broadcast along with the signed suicide note.

4.3   Flypaper and Trolling Attacks

One challenge for a decentralized suicide scheme is ensuring that multiple nodes do not issue suicide notes for a single misbehaving node. In a *flypaper attack*, a malicious node in a fixed location presents widely observable misbehavior to attract many simultaneous suicides. A base station S can trivially resolve multiple suicide offers for the same node by accepting just one of them:

1. A :   detects M misbehaving
2. $A \longrightarrow S : A, M, \mathrm{HMAC}_{K_{AS}}(\mathrm{suicide}_{A,M})$
3. S :   verify signature, wait for duplicates
4. $S \longrightarrow B : A, M, \mathrm{HMAC}_{K_{SB}}(\mathrm{suicide}_{A,M}) \ \forall B \in N$
5. $*$:   verifies signature, deletes keys shared with A, M, adds to blacklist

In a decentralized scheme, where each node must be able to reach a decision independently, two precautions can mitigate a flypaper attack. First, a node can wait a random back-off period ($0 \leq t_\mathrm{r} < t_\mathrm{max}$) before transmitting an offer. If it observes another suicide note for the same node while waiting for its own timer to expire, the node abandons its offer in favor of the already-published one. If its timer does expire, the node transmits a suicide message. Larger values of t_max lower the probability of a collision at the expense of slower revocation. This back-off can significantly reduce the number of simultaneous transmissions; however, duplicate offers are still possible if a second timer expires before the first transmitted suicide message is received by the second node.

To address this possibility, a tie-breaking mechanism is required. If loose time synchronization exists in the network, nodes can append a timestamp to their signed suicide message. Nodes then wait long enough for all offers to be broadcast (t_{bcast}) and honor the suicide note with the earliest timestamp. Alternatively, time synchronization can be avoided by using a using a random number transmitted along with each suicide message. However, using time stamps to resolve conflicts is more efficient since earlier offers are likely to propagate faster. One consequence when using one-time signatures is that we must now store Q key pairs per node, or generate signing keys on the fly from a secret (using a modified hash chain or stream cipher encryption of a secret). Here is the distributed protocol for two nodes A and B detecting M misbehaving:

1a. A :   detect M misbehaving; start random timer t_r
1b. B :   detect M misbehaving; start random timer $t_{r'}$
2. A :   Timer t_r expires (assuming $t_r < t_{r'}$)
3. $A \longrightarrow * : A, M, t_A, \{\mathrm{suicide}_{A,M}, t_A\}_{K_A^{-1}}$
4. $*$:   waits t_{bcast} for earlier offers, verifies signature, deletes keys shared
                 with A, M and adds them to blacklist

Trolling is where a node presents itself in several locations, either re-using identities (node replication) or presenting different ones (Sybil). This could be done with the aid of collusive malicious nodes that present the same misbehaving identity in multiple locations. Alternatively, a powerful transmitter or flying

over the area achieves the same effect. We have assumed in this paper that other mechanisms exist for detecting and preventing Sybil and node-replication attacks. However, our multiple-offer resolution mechanism addresses trolling with re-used identities even when node replication detection is not available, provided the network is connected and a long enough time-out is used to allow multiple offers notes to traverse the network.

If the adversary is capable of partitioning the network, then a single malicious node can kill multiple good nodes either by issuing different suicide notes in each partition, or by misbehaving in different partitions with the aim of prompting multiple suicide notes. The number of honest nodes affected is proportional to the number of partitions. A potential countermeasure is *resurrection*: once the network is reconnected, several suicide attacks on a single node can be converted into the resurrection of all but the first sacrificed node along with revoking the replicated node. In this case, suicide notes would have to be stored, and a blacklist operated in preference to deleting keys. (Note that if revocation is reversible, then all the mechanisms compared in this paper become more complex.)

4.4   Extensions: Probabilistic Suicide and Suicide Pacts

Suicide may not be well suited to detection mechanisms that identify malicious nodes with less than high confidence. Yet the basic mechanism can be extended to cope with uncertainty. Suppose a node detects behavior that is probably malicious, and can assign a probability to this (e.g., bad with $p = 0.7$).

One solution is for a node to maintain a running total for each node it can observe. When the total exceeds a specified threshold (e.g., $\sum p_i \geq 1$), a suicide attack is triggered. A stateless alternative is for the node to attack with probability p. One limitation of these approaches is that each node operates in isolation, gaining no benefit from the collective knowledge of its neighbors.

We can modify the suicide offers to include probability p as $\text{offer}_{A,M}, t_A, p,$ $\{\text{offer}_{A,M}, t_A, p\}_{K_A^{-1}}$. Thus revocation decisions can be made on collective knowledge of uncertain observations, and the nodes participating in this decision might be thought of as having entered into a *suicide pact* against the suspect. Deciding which member of the pact has to carry out the suicide attack should reflect the probability claimed in the suicide offer, whether based on a weighted, verifiable coin toss, or simple probabilistic attack in the second round of the protocol.

5   Analysis and Comparison

5.1   Storage and Communication Costs

A comparison of the storage costs is presented in Table 1. Each column represents the tasks discussed in Sections 2.3 & 3.1.

Reelection is more efficient than blackballing in terms of storage: access shares ('positive votes') need only be verified by one node, and only one node (the target node) need store the authentication information for the recovered token. Thus, storage costs for reelection are $O(vT + \log n)$, compared to $O(vT + v \log n)$ for

Table 1. Node storage costs for alternative schemes

Node storage for	1.	2.	3.	4.
Blackballing	vT	$v \log Tv$	vT	$1 + v \log n$
Reelection	vT	0	vT	$1 + \log n$
Suicide (sym.)	$O(Q)$	$Q \log nQ$	0	1
Suicide (asym.)	1	1	1	1

blackballing. However, reelection arguably may require more, shorter time periods (and hence larger values for T) since a revoked node does not immediately lose access to the network but only at the end of the current time period. As with blackballing, we can also reduce v using a weakened threat model.

Both blackballing and reelection increase storage well beyond the initial costs of key distribution. This is not easily borne, particularly for large networks of constrained devices. In contrast, suicide using one-time signatures is not affected by the number of keys that are pre-assigned. Here, a node only needs the ability to transmit a very small number (Q) of offers. But the size of each public and private key for one-time signatures can be very large, requiring two hash values per signed bit. Suicide using asymmetric cryptography requires far less storage as nodes keep only their own private and public keys (small when elliptic curve cryptography is used) as well as the owner's public key and certificate.

Table 2 shows the respective communication costs associated with each scheme. Reelection is unique in that there is a fixed cost per session when a nodes asks for and receives its access shares. This requires a node to broadcast a request to its immediate neighbors, while the k shares can be returned as unicast messages. The reconstructed token is also broadcast. However, communication costs do not increase as nodes get revoked. In contrast, the communication costs of blackballing increase with the number of votes cast. $k + 1$ locally broadcast votes are required to remove a node, where each vote comprises the vote and $\log v$ path-authentication values. These messages may need to be forwarded by other nodes since we are not guaranteed that all voting members are within communication range of each other. The final revocation order must be broadcast across the network (along with $\log n$ path authentication values) to ensure that everyone revokes the malicious node. Since votes are only valid in a given session, it is possible for up to k votes to be cast each session without revoking a node. Thus, blackballing is more efficient than reelection under low rates of misbehavior; reelection fares better when more attackers are present.

Asymmetric suicide is noted for its low communication costs (only one network-wide broadcast), though these energy savings are offset by increased computational expense due to the use of signatures when compared to symmetric-key-based schemes. Of course, voting schemes using asymmetric cryptography faces even higher costs since signatures are required for every vote.

Suicide using one-time signatures can face high communication costs, particularly if the public key must be transmitted along with the signature instead of being pre-loaded onto devices. In fact, recent work has demonstrated that one-time-signature schemes perform only slightly better than elliptic curve

Table 2. Communication costs for alternative schemes

	# sessions	Setup per session	Comm. per rev.	Min accrued comm. without revocation	Max comm. without rev.
Blackballing	T	0	$k + 1$ broadcasts	0	kT
Reelection	T	k unicasts +2 broadcasts	0	$2kT$ unicasts $+T$ broadcasts	$2kT$ unicasts $+T$ broadcasts
Suicide (both)	1	0	1 broadcast	0	0

algorithms when considering both the communication and computational over-head [22]. Thus, the limited asymmetric cryptographic operations needed by suicide may be preferable to the higher complexity and storage requirements imposed by one-time signature schemes.

Suicide does offer other advantages over any voting-based scheme, however. Suicide does not suffer from the problem of stale votes nor a delay before the revoked node is removed from the network. It also requires fewer, less restrictive assumptions. Specifically, suicide places no restrictions on node mobility for normal or compromised nodes. It also places no topological restrictions, such as requiring nodes to have a minimum number of neighbors.

On the other hand, suicide does require good nodes to value the social welfare of the network over individual utility. This condition is reasonable whenever the nodes are deployed by a single entity (e.g., a sensor network deployed on a battlefield) but may be less so when nodes are individually controlled (e.g., a peer-to-peer file-sharing system) [23].

5.2   Denial-of-Service Attacks

Suicide enables precision denial-of-service attacks since adversaries can remove any node. Network topology differences increase the importance of some nodes due to their location or number of neighbors. Even unsophisticated attackers can wreak havoc by taking out high-value nodes with low-value-node suicides.

But suicide is arguably less susceptible to DoS attacks than threshold voting schemes. Threshold voting schemes become totally vulnerable once the attacker gains sufficient numerical advantage (exceeding the threshold) in a region. Here the adversary can vote out all good nodes in the area. This is a particular concern when devices are mobile as an attacker can use the minimum number of compromised devices, moving them around the network and ejecting good nodes unchallenged. Suicide, by contrast, bounds the maximum amount of damage a set of malicious nodes can do but means that twice as many nodes are removed from the network – one good node for every bad node.

5.3   Quantifying Suicide Abuse

While protections against flypaper and trolling attacks ensure only one good node is sacrificed per bad node, reconciling several suicide notes due to these
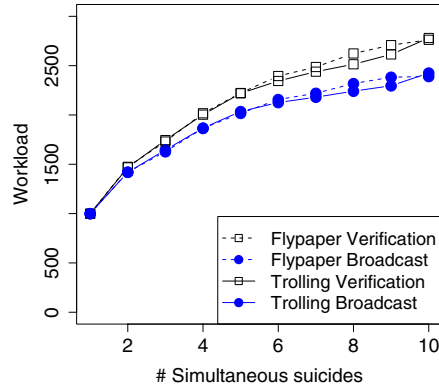
Fig. 1. Workload versus simultaneous suicides

attacks triggers increased communication and computational complexity. An attacker may still attempt flypaper or trolling attacks aiming to consume resources (e.g., battery life or network capacity) by forcing multiple offers to be resolved.

We quantify the increased workload due to these attacks compared to the transmission of a single suicide note. We use simulation since this analysis probabilistically depends on the topology of the network as well as the random back-off period. We consider three scenarios of increasing complexity: normal operation where only a single suicide note is issued; a simple collision where two suicide notes are issued simultaneously; and a trolling attack where misbehavior is presented to nodes across the network simultaneously in the absence of node replication detection to trigger multiple suicide offers. We compute two quantifiable measures: the number of broadcasts attributed to all suicide offers and the number of signature verifications attributed to all suicide offers.

We simulated a wireless network comprised of 1000 nodes uniformly distributed over a plane, where the communication radius of nodes ensures an average of 60 immediate neighbors in communication range. We averaged our results using 10 iterations on a network sample. Each suicide note is embedded with a timestamp. A node re-broadcasts a received suicide note to its immediate neighbors if it is either the first one received, or has the earliest timestamp. In this way, suicide notes are propagated throughout the network until the one with the earliest timestamp completely dominates the network.

When one suicide note is broadcast, every node in the network broadcasts and verifies once. Two simultaneous suicide notes increases the workload by approximately 50%, a manageable rise. But what is the effect of additional simultaneous suicides? Figure 1 plots workload (number of broadcasts and verifications) during trolling and flypaper attacks as a function of the number of simultaneously issued suicide notes. As expected, the computational and communication burden increases. Notably, the function is mostly marginally decreasing, so that most additional suicides increase the workload less than previous ones. At most,

resource consumption attacks increase system workload by a small multiple; thus we conclude they are not as dangerous a threat as originally feared.

6   Conclusions

A major challenge for ad-hoc networks is how to remove nodes that are observed to be behaving badly. Existing threshold voting proposals for node revocation enfranchise too many of the wrong nodes, undermining their efficiency and security. They are susceptible to manipulation, particularly if nodes are mobile.

So we switched from voting against bad nodes to a protocol where good nodes reelect each other to the club at regular intervals. Reelection reduces storage costs by shifting the responsibility of verifying votes from a node's neighbors to the node itself. This is a significant improvement, but simple reelection remains infeasible for severely constrained devices. We then proposed a lightweight reelection mechanism requiring no pre-assigned storage and using just hash operations: each node broadcasts a buddy list of trusted neighboring nodes locally at each time period. This can support a much wider range of membership strategies and is significantly cheaper. However (like the other mechanisms we have discussed) it still has some communications costs, and it may be less effective where there are many mobile nodes or an uneven network topology.

We then showed that the most effective way of doing revocation in general ad-hoc networks is the suicide attack. A node observing another node behaving badly simply broadcasts a signed message declaring both of them to be dead. This is cheap; it scales well; it is not affected much by mobility; and it works across interesting parameter ranges. Such strategies are well known in nature, from bees attacking an intruder to the operation of helper T-cells in the immune system. They even find an echo in some human societies, such as the dueling culture of the eighteenth century and the US Wild West. We believe that suicide attacks are attractive for a wide range of distributed system applications.

References

1. Anderson, R.: The eternity service. In: First International Conference on the Theory and Applications of Cryptology (PRAGOCRYPT) (1996)
2. Eschenauer, L., Gligor, V.D.: A key-management scheme for distributed sensor networks. In: 9th ACM Conference on Computer and Communications Security (CCS), ACM, pp. 41–47. ACM Press, New York (2002)
3. Chan, H., Perrig, A., Song, D.X.: Random key predistribution schemes for sensor networks. In: IEEE Symposium on Security and Privacy (S&P), pp. 197–213. IEEE Computer Society Press, Los Alamitos (2003)
4. Du, W., Deng, J., Han, Y.S., Varshney, P.K.: A pairwise key pre-distribution scheme for wireless sensor networks. In: 10th ACM CCS, pp. 42–51. ACM, New York (2003)
5. Liu, D., Ning, P.: Establishing pairwise keys in distributed sensor networks. In: 10th ACM CCS, pp. 52–61. ACM Press, New York (2003)

6. Zhu, S., Setia, S., Jajodia, S.: LEAP: efficient security mechanisms for large-scale distributed sensor networks. In: 10th ACM CCS, pp. 62–72. ACM Press, New York (2003)

7. Anderson, R.J., Chan, H., Perrig, A.: Key infection: Smart trust for smart dust. In: 12th IEEE International Conference on Network Protocols, IEEE Computer Society, pp. 206–215. IEEE Computer Society Press, Los Alamitos (2004)

8. Douceur, J.R.: The Sybil attack. In: Druschel, P., Kaashoek, M.F., Rowstron, A. (eds.) IPTPS 2002. LNCS, vol. 2429, pp. 251–260. Springer, Heidelberg (2002)

9. Newsome, J., Shi, E., Song, D.X., Perrig, A.: The Sybil attack in sensor networks: analysis and defenses. In: 3rd International Symposium on Information Processing in Sensor Networks, pp. 259–268. ACM Press, New York (2004)

10. Parno, B., Perrig, A., Gligor, V.D.: Distributed detection of node replication attacks in sensor networks. In: IEEE S&P, IEEE Computer Society, pp. 49–63. IEEE Computer Society Press, Los Alamitos (2005)

11. Hu, Y.C., Perrig, A., Johnson, D.B.: Packet leashes: A defense against wormhole attacks in wireless networks. In: 22nd IEEE INFOCOM, IEEE Computer Society Press, Los Alamitos (2003)

12. Brands, S., Chaum, D.: Distance-bounding protocols (extended abstract). In: Helleseth, T. (ed.) EUROCRYPT 1993. LNCS, vol. 765, pp. 344–359. Springer, Heidelberg (1994)

13. Hancke, G.P., Kuhn, M.G.: An RFID distance bounding protocol. In: IEEE Secure Comm., pp. 67–73. IEEE Computer Society Press, Los Alamitos (2005)

14. Capkun, S., Buttyan, L., Hubaux, J.P.: SECTOR: secure tracking of node encounters in multi-hop wireless networks. In: 1st ACM Workshop on Security of ad hoc and Sensor Networks, pp. 21–32. ACM Press, New York (2003)

15. Marti, S., Giuli, T.J., Lai, K., Baker, M.: Mitigating routing misbehavior in mobile ad hoc networks. In: 6th International Conference on Mobile Computing and Networking, pp. 255–265. ACM Press, New York (2000)

16. Chan, H., Gligor, V.D., Perrig, A., Muralidharan, G.: On the distribution and revocation of cryptographic keys in sensor networks. IEEE Transactions on Dependable Secure Computing 2(3), 233–247 (2005)

17. Moore, T., Clulow, J.: Secure path-key revocation for symmetric key predistribution schemes in sensor networks. In: 22nd IFIP TC-11 International Information Security Conference 2007 (to appear)

18. Anderson, R., Bergadano, F., Crispo, B., Lee, J.H., Manifavas, C., Needham, R.: A new family of authentication protocols. ACM SIGOPS Operating Systems Review (OSR) 32(4), 9–20 (1998)

19. Clulow, J., Moore, T.: Suicide for the common good: a new strategy for credential revocation in self-organizing systems. ACM SIGOPS OSR 40(3), 18–21 (2006)

20. Perrig, A., Canetti, R., Tygar, J.D., Song, D.X.: Ecient authentication and signing of multicast streams over lossy channels. In: IEEE S&P, pp. 56–73. IEEE Computer Society Press, Los Alamitos (2000)

21. Merkle, R.C.: A certified digital signature. In: Brassard, G. (ed.) CRYPTO 1989. LNCS, vol. 435, pp. 218–238. Springer, Heidelberg (1990)

22. Seys, S., Preneel, B.: Power consumption evaluation of efficient digital signature schemes for low power devices. In: IEEE International Conference on Wireless And Mobile Computing, Networking And Communications, pp. 79–86. IEEE Computer Society Press, Los Alamitos (2005)

23. Danezis, G., Anderson, R.: The economics of resisting censorship. IEEE Security& Privacy 3(1), 45–50 (2005)