TRAIL: Topology Authentication in RPL

Heiner Perrey, Martin Landsmann, Osman Ugus
Dept. Informatik, HAW Hamburg
first.last@haw-hamburg.de

Matthias Wählisch
Inst. Informatik, FU Berlin
m.waehlisch@fu-berlin.de

Thomas C. Schmidt
Dept. Informatik, HAW Hamburg
t.schmidt@haw-hamburg.de

Abstract

The IPv6 Routing Protocol for Low-Power and Lossy Networks (RPL) was recently introduced as the new routing standard for the Internet of Things. Although RPL defines basic security modes, it remains vulnerable to topological attacks which facilitate blackholing, interception, and resource exhaustion. We are concerned with analyzing the corresponding threats and protecting future RPL deployments from such attacks. In this paper, we derive and evaluate TRAIL, a generic scheme for topology authentication in RPL. TRAIL solely relies on the basic assumptions of RPL that (1) the root node serves as a trust anchor and (2) each node interconnects to the root as part of a hierarchy. Using proper reachability tests, TRAIL scalably and reliably identifies any topological attacker with little cryptographic efforts.

Keywords

IoT, routing security, mobile security, performance

1 Introduction

RPL [1] has been designed as an efficient and scalable routing protocol for low-power and lossy networks (LLN). It promises to reduce the overall power consumption by minimizing the control traffic, which is a major requirement for the energy constrained devices envisioned in the future Internet of Things (IoT). Such tiny intercommunicating devices like sensor nodes used in (home) automation, smart grids or surveillance systems are expected to massively populate our environment soon.

RPL constructs one or several tree topologies oriented towards a single root node. Each node in the RPL routing graph has a *rank* derived from its parent relationship that describes the topological distance to the root. Every node joining the topology calculates a higher rank than its parent, lower ranks are used for default upstream. This proactive organization leads to a Destination Oriented Directed Acyclic Graph (DODAG) topology, from which RPL is able to detect and remove inconsistencies reactively.

Control traffic in this topology consists of *DODAG Information Objects* (DIOs). A DIO advertises parameters and constraints for a specific DODAG that is uniquely identified by a version number. A node uses the information obtained from a DIO to select a parent node, compute its rank and join the DODAG, from which it inherits an upward route towards the root node. An optional upwards advertisement of *Destination Advertisement Objects* (DAO) generates downward-oriented routes to children of a subtree. Depending on the mode of operation, these routes are either maintained and stored at each node (*storing mode*), or forwarded to the root node and collected there (*non-storing mode*). The integrity of distribution trees is essential for RPL, as an inconsistent hierarchy will lead to traffic redirections and a loss of routes to the root. In addition, RPL will attempt to cure tree deficiencies by reorganization, and a node that will hold up failures of the routing hierarchy may trigger repeated reconfigurations that drain resources of the network.

RPL offers basic protection against external topology attacks [1]. However, as nodes may be captured and security keys can be extracted from them, the RPL topology is threatened by various attacks from inside the network [2]. The rank of a node and the DODAG version number are focal attributes in the topology. Known attacks are foremost based on them. A false rank of a node forges the relative topological distance to the root and disarranges the hierarchy. An inconsistent version breaks the reference to the topological graph and causes the network to reconfigure. Corresponding protections are not part of the current RPL specifications.

As major countermeasure, VeRA [3] has been proposed to fix these two classes of vulnerabilities by adding reverse hash chaining to DIO messages. Receivers shall be enabled to verify the advertised hierarchy. However, we could show that VeRA remains vulnerable to rank attacks by forgery and replay. In this paper, we present a more generic approach to solve the problem of topology authentication in RPL. Leaving aside the complexity of VeRA, our remaining work concentrates on TRAIL (Trust Anchor Interconnection Loop) that discovers and isolates bogus nodes while they attack the RPL routing hierarchy. TRAIL is derived of first hand principles and resolves the issues of topological infringements.

The remainder of this work is structured as follows. Section 2 discusses the problem of securing RPL, common attacks and related work. Section 3 introduces and proves TRAIL, our generic solution for topology authentication. TRAIL is thoroughly evaluated in Section 4. Finally, we conclude in Section 5 and look out on future work.

International Conference on Embedded Wireless Systems and Networks (EWSN) 2016
15–17 February, Graz, Austria

ISBN: 978-0-9949886-0-7

2 RPL Security Challenges & Related Work

RPL constructs a reverse path forwarding hierarchy by announcing tree parameters in the downward direction, starting from the root node. A node that successfully joined the tree advertises its rank towards its potential children in so called DIO messages, while unconnected nodes select as parent the neighbor of lowest rank, i.e., in closest position to the root. Following this algorithm, a fully connected acyclic, hierarchical graph is created in compliance to wireless reachability. Each of such DODAGs is associated with a unique version number to survey consistency.

RPL specifies secured control plane messages for authenticity, integrity, and optional confidentiality [1]. Even though these basic security features defend against external attackers [4], RPL remains unprotected against adversaries from inside the network [3, 5]. Capturing a node and extracting security credentials enables an attacker to gain access to the control plane and to modify the routing topology. The rank and the version number are the key information for defining the structure of the routing system. The essential challenge for securing the routing topology thus is to protect rank and version number from any unwanted modification. Strong identity-based end-to-end authentification as introduced in [6] could defend a RPL routing system against internal modifications. However, its inherent complexity prevent this from being a standard solution. Next, we introduce the core attacks against the RPL topology and the assumptions made on the attacker.

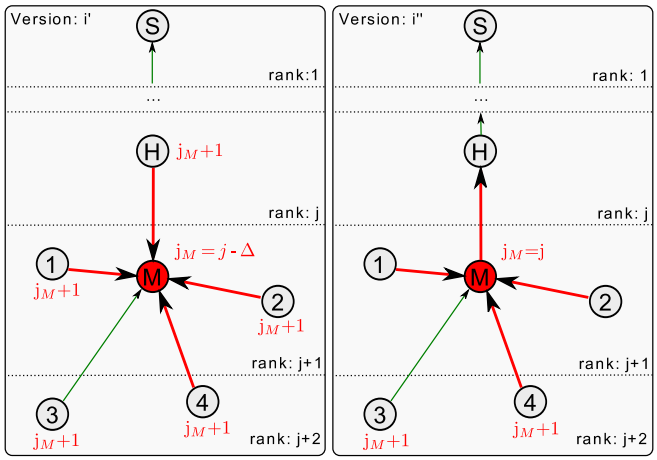
2.1 Attacker Model

We assume the presence of one or multiple attackers that physically captured and compromised multiple, arbitrary nodes on the network. The attacker has access to all available keys on the captured nodes, which include all information for joining and participating in the DODAG without restrictions. The compromised nodes are successfully integrated in the network and are thus authorized to transmit authenticated messages. Furthermore, the attacker is limited by the resources and constraints of the captured nodes. Hence, we assume that the attacker cannot install directed antennas or create multiple identities [7] to seemingly use several malicious nodes with one physical interface or to establish out-of-band channels. The attacker aims at maximizing his impact on the network, for example by attracting as much traffic as possible for eavesdropping or sink-holing, or by affecting the operational conditions of as many nodes as possible.

2.2 Topology Attacks

Version Number Attacks: The version number of the DODAG is increased by the root node, whenever a global repair is needed. This occurs, if inconsistencies cannot be repaired locally. In a version number attack [3], an attacker illegally increases the version number of the DODAG. Publishing a higher version number will lead to a reconstruction of the RPL topology. This either serves as a preparation for a following attack such as on the rank, or can be repeatedly executed to disturb the network and drain the resources of nodes.

Rank Spoofing Attack: In a rank spoofing attack [3], a malicious node propagates an incorrect rank to change its position in the routing tree. Commonly, an attacker will choose a lower rank to improve its position in the hierarchy and achieve larger impact on the network. In response to forged rank advertisements, neighboring nodes select the attacker as parent and forward traffic towards it. Fig. 1(a) visualises the topological manipulations caused by a strict rank decrease. The attacker M propagates the lowest rank of the vicinity and attracts all its neighbors. In this example, the parent node H is also attracted by the malicious node M, which creates a sinkhole. Node 3 correctly selects M as parent, but unknowingly propagates the illegal rank downtree. Thus, 3 and its parents potentially attract even more children and increase the number of nodes that forward traffic towards the attacker M.



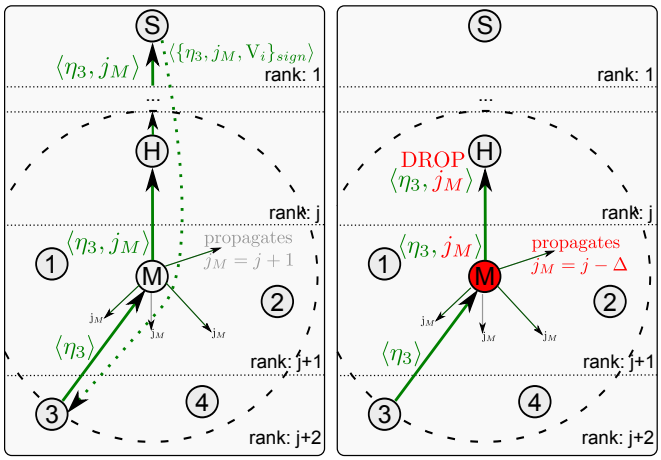
(a) Topology after a rank spoofing (b) Topology after a replay attack

Figure 1. RPL topology (a) with rank spoofing. The attacker M propagates a rank j_M falsely decreased by Δ, and thereby incorrectly attracts nodes 1, 2, 4, and the parent node H, which creates a sinkhole. (b) visualizes a replay of the parent rank, only attracting nodes 1, 2, and 4 with intact upstream to H.

Rank Replay Attack: An attacker who learned a valid rank from a (potential) parent may replay this value in its own advertisements and pretend to run at one hierarchy level above the proper value. This special case of a rank spoofing will not disconnect the attacker from the root as visualised in Fig. 1(b). In contrast to arbitrary rank forgery, the replay allows a malicious node to re-use a proper rank, even if rank verification schemes apply. We will show in the following section that present protection schemes are vulnerable to rank replay attack.

2.3 Related Work

Recent work has classified the different attacks on RPL [8], but only limited work has addressed the security of the RPL routing system. A security threat analysis for LLNs by the IETF [4] focuses on potential threats and attacks. However, the analysis solely proposes generic countermeasures to the described attacks. Some attempts have been made to deal with topology attacks [9, 3, 10, 11, 12]. VeRA addresses the rank and version number attacks by adding a rank and version control obtained from hash chaining [3]. While successfully



(a) Successful Rank Validation (b) Failed Rank Validation

Figure 2. TRAIL SINGLE RANK VALIDATION: Node 3 initiates the rank validation by sending a nonce, η, to parent *M*. In (a) *M* announces its true rank. The message arrives at the root and is singed. In (b) *M* uses a forged rank, hence the message is dropped.

mitigating a version number attack, the VeRA approach is still subject to two topology attacks [13]. The first attack is a general *rank spoofing*, which allows an attacker to pretend any rank and therefore any position in the DODAG. The second attack is a *rank reply attack*, which allows an attacker to claim one level closer to the root by replaying its parent rank. An extended version of the present paper [14] analyzes these vulnerabilities in detail and derives countermeasures.

Our work goes beyond mitigating sinkhole attacks. By performing generic topological tests, we inquire on the integrity of the routing hierarchy, identify and isolate individual attackers.

3 TRAIL – Trust Anchor Interconnection Loop

We introduce *TRAIL*, our generic approach to detect and prevent topological inconsistencies. In contrast to the previous approaches, each node is enabled to validate its upward path to the root and to detect rank spoofing on it. Our test furthermore identifies the largest sub-DODAG(s) affected by non-monotonous rank order. Having learned such inconsistency, the root of that sub-DODAG may either trigger a local repair, or disconnect its malicious sub-tree and rely on alternate paths. In the following, we treat ranks as monotonously increasing integers. It is noteworthy that any RPL rank function is monotonous and can be reverted to an integer chain.

3.1 TRAIL Idea: Path Validation

The key idea of TRAIL is to validate upward paths to the root using a round trip message. Without relying on encryption chains as in VeRA(++), a node can conclude rank integrity from a recursively intact upward path.

A child node that received a rank advertisement from its parent initiates a *positive attestation* of the rank as follows. It sends a test message with a random nonce η upwards to its parent. The parent adds its rank j and forwards the test message $\langle j, \eta \rangle$ upstream towards the root. At each intermediate hop, the receiving upper node verifies that (a) the rank in the test message is higher than its own, and (b) the rank of the sending node lies in between the rank of the test message and its own. If a rank violation is observed, the test message is discarded and the sub-DODAG gets either disconnected or a local repair is started (see Fig. 2). The test message eventually arrives at the root, which adds the current version number to the test message and signs for its way back to the initiating client. Before forwarding, every node verifies whether the signed message contains the scribed rank j that is larger than its own rank. A violation stops the propagation of the message. On reception, the client verifies the signature, matches its nonce, and obtains evidence of the current version number and the rank advertised by its parent. As the rank announcement had consistently travelled to the root, no honest node on the path had observed a rank violation and the upstream is valid. A child not receiving the reply, continues without positive attestation of its parent. It may choose another upstream, if available, or apply additional measures for transport security.

After all nodes have applied this test recursively down the hierarchy with success, it is assured that none of the nodes has a parent that illegally lowered its rank. The highest ranked node that unsuccessfully performs the test identifies the root of the largest sub-DODAG affected by rank spoofing. It should be noted, though, that a directly connected chain of k malicious nodes can secretly replay rank values $k-1$ times so that they are counted in the test as one node. However, this costly attack does not decrease rank values of the attackers, but solely extends the wireless reach of the malicious group and cannot be observed without surveillance of the wireless geometry.

As every node in the network needs to inquire with the root individually, the overhead in messages and signature processing grows linearly with the network size. Hence, this simple scheme of path validation suffers the obvious drawback of scalability. In the following, we will present an aggregated scheme that keeps messages per node and signature computation constant.

3.2 Scalable Path Validation

3.2.1 Rank Attestation Scheme

The path validation can be turned into a scalable procedure by aggregating all client-specific inquiries into a single message exchange. Starting from the leaf nodes of a DODAG, we design a convergecast that reaches up to the root. The root node receives and signs a single, converged request that serves as a universal path attestation message when distributed downtree via multicast.

After a leaf node $N_{l,k}$ of the DODAG has received the rank advertisement of its parent (and discovered that it has no further children), it issues a nonce $\eta_{l,k}$ to its parent. The parent node collects the nonces $\{\eta_{l,k}\}_k$ of all children and writes them into a single array element. For space efficiency, the parent combines the nonces in a Bloom filter [15]. Note that this Bloom filter can be very short, as the number of entries is limited by the number of children per node. This array element containing a single Bloom filter is sent upstream to the grandparent and saved by the node.

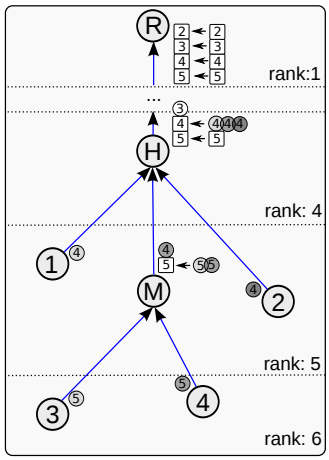


Figure 3. TRAIL RANK ATTESTATION

From each of its children, the grandparent receives such an array of Bloom filters together with an individual nonce. It should be noted that these arrays need not be of equal lengths, as the tree may be unbalanced. The grandparent aligns every array on the position below the child node rank and merges the entries of equal index using the scalable Bloom filter technique of Almeida et al. [16]. In detail, the grandparent node extracts all first index elements $A_i(1)$, merges them and writes the result to a new output array B at the index 2 (incremented by one). In general, $\{A_i(k)\}_i$ are merged into $B(k+1)$, if existent. Finally, the node adds the Bloom filter that aggregates all nonces of its immediate children to the array element $B(1)$ forwards the array B upwards together with its own nonce and saves both B and its nonce.

As depicted in Fig. 3, in proceeding this way stepwise towards the root, an array is created whose index represents the rank and whose values are merged Bloom filters of all nonces issued at a specific rank. Thereby array elements are of variable length, each accommodating the concatenated Bloom filters as generated according to the shape of the tree. Additionally every node on the path saves the array and nonce they forward for latter validation. The root node adds the current version number and signs the data structure consisting of the Bloom filter array and the version number. Thereafter, the signed data is distributed via multicast down the tree.

On the reception, each node can verify the version, and the rank of its parent. It accesses the corresponding array element to match its nonce in the Bloom filter and verifies that no further array element contains the same nonce. Finally, it verifies that the signed Bloom filter array does not contain less nonces than the previously saved array. Note that the probability of a false positive hit can be chosen sufficiently low when configuring the Bloom filter. A successful match testate that ranks have increased monotonically from the root downwards and that the array and contained nonces have not been manipulated or reordered. Whenever the matching fails, monotonic rank order has been violated on the upward path from the current node to the root. The highest ranked node detecting such violation forms the root of an inconsistently connected sub-DODAG. Any node experiencing such inconsistency may choose another upstream, if available, or apply additional measures for transport security.

3.2.2 *Security Proof*

We show that a malicious node cannot improve its rank by modifying the data structure, and that improper modifications are detected in the verification phase.

Assumptions. We rely on the attacker model specified in Section 2.1. In particular, we refer to an attacker that has no means to establish an out-of-band communication channel. A chain of k malicious neighbors is considered as one attacker with an extended wireless reach. Distributed attackers scattered among different hierarchy levels communicating out-of-band channel cannot be detected and are not considered in our model. However, non-collaborating attackers distributed in the topology are considered. Finally, we ignore the false positive rates on queries to bloom filters as they can be made arbitrarily small by choosing appropriate parameters.

PROOF. We consider the security of TRAIL in existence of i) multiple non-collaborating malicious nodes and ii) multiple malicious nodes with limited collaboration:

i) Multiple non-collaborating malicious nodes: Since the nodes are not allowed to collaborate, they can be considered as multiple single attackers. For simplicity, we provide the analysis for a single malicious node. A malicious node receiving a topology test message $\langle \eta, A \rangle$ from its child(ren) has the option to (1) not include its child(ren) in the message array or to not merge-and-forward the array A at all. It may as well (2) rearrange the array, and in particular include the nonces of its child(ren) at a wrong array position. It may (3) attempt to exclude itself from the attestation hierarchy by not submitting its nonce value to its parent. These four choices of malicious nodes will lead to the following conditions:

C1. By not forwarding the test nonces of its children or the attestation array, the malicious node causes its immediate detection. When receiving the signed attestation message of the root, the child(ren) of the malicious node will test for its nonces without success and detect the inconsistency.

C2. The best a malicious node can do to its children is writing nonces at the foreseen position. Any misplacement will move data of the children to a lower rank position and thus cannot be aligned with a malicious rank upgrade. Other rearrangements of the array will change the data positions for nodes lower in the tree. This implies that affected nodes are not within the wireless transmission range of the malicious node – they had chosen the better rank of the malicious node otherwise. As the malicious node cannot coordinate rank advertisements outside its wireless reach, nodes will remain unaware of their nonce moving to other rank positions. Nodes will thus search at the original rank position in the attestation message and corresponding tests will fail.

C3. If the malicious node withholds its own nonce, but cooperates in traversing the merged filter array, its honest parent will merge the data with data from its other children and insert at the proper position. Not delivering the nonce will simply lead to a Bloom filter that does not contain the nonce of the malicious node. Hence, an

malicious node causes nothing but excluding itself from the verification process.

ii) Multiple malicious nodes with limited collaboration: We mean by a limited collaboration that multiple attackers know in advance their position in the topology and the desired rank which they want to claim during an attack. This can be realized by configuring them accordingly during their deployment. Limited indicates that once they are deployed, those malicious nodes, which are not within each other's communication range, cannot communicate anymore. TRAIL mitigates such attacks as follows: A malicious node close to the root merges array elements on behalf collaborating malicious nodes lower in the topology that claim a false rank. Consequently, nonces of honest nodes that are affected by the rank spoofing, are moved to the correct array element. However, due to the malicious merging of array elements, these nonces exist multiple times. Such a duplicate either denotes a fraud or a false positive. Given a false positive rate of f, we detect the attack with probability $1-f$. Deleting nonces from filters will cause that an honest node on the path will detect the attack by comparing the forwarded array with the signed one.

In any of the cases, forgery will not comply to a rank decrease and will be detected, whenever it affects third party nodes. All parents of a malicious node will always exclusively write to the lower rank-test positions, which is the obvious protection from rank spoofing in this procedure. □

3.2.3 Details of the Bloom Filter

We use Bloom filters [15], a space-efficient random data structure, to reduce message lengths in our attestation scheme. A Bloom filter is defined as a bit-vector, v of m bit and represents a data set. By using k independent hash functions, each element of a set of $A = \{a_1, \ldots, a_n\}$ is mapped to k bits in v. By these means, the size of each input element is reduced to at most k bits. Due to randomized overlapping of bits from different elements, the size may be reduced even further, but this may return a false positive result of a query. Essentially, there is a linear relation between number of bits used for storing each element, and the false positive rate. Mitzenmacher [17] could show that properly designed Bloom filters can be compressed even further by about 30 % at a given false positive rate. Almeida et. al. [16] designed a scalable extension of Bloom filters that linearly add filter elements with increasing set sizes.

In TRAIL, we require tiny Bloom filters that store nonces from the children set of a single node. For a commonly small fanout of k nodes and a false positive rate below 1%, an appropriate bit-size m of the (compressed) Bloom filters can be estimated as $m = 6k$ [bits].

4 Performance Evaluation

To evaluate the performance of our RPL security scheme, we have implemented TRAIL authentication (attestation and announcement messages) as an extension to the existing RPL protocol implementation on the RIOT platform [18]—an integration of TRAIL in RPL ICMP control messages has little effect on performance and is left to future protocol engineering work. We deployed TRAIL on the DES Mesh Testbed of FU Berlin [19] and performed the comparative experiments described below. The focus of our evaluation lies on the overhead cost and the temporal performance of TRAIL in comparison to unmodified RPL routing. The critical cost metrics for wireless sensor nodes are over the air transmission, e.g., the number of messages sent, as well as message sizes.

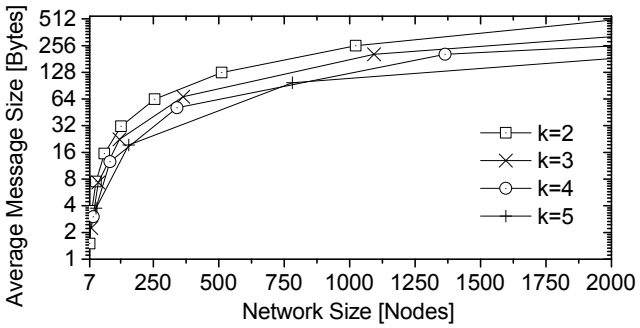


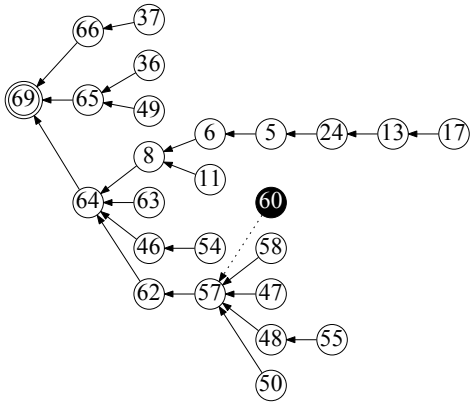
Figure 4. TRAIL MESSAGE SIZES: Average message size distribution for varying fanout degrees k as functions of the network size.

Table 1. Message overhead for different network sizes: (k= number of children, h=height of the tree)

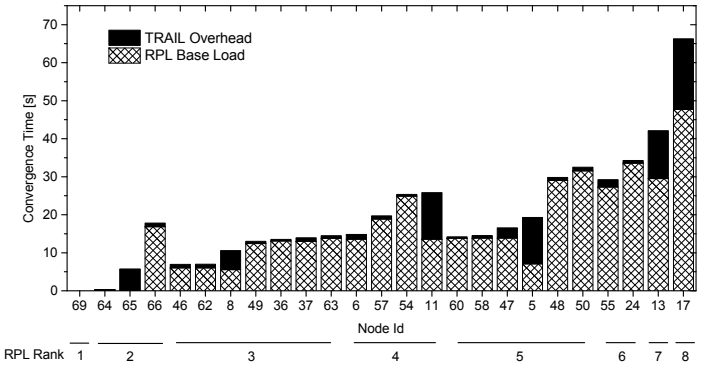
Network Configuration			Message Overhead [Bytes]		
k	h	# Nodes	# Msg. per node	Average Size	Max. Size
2	3	15	2	3.5	10.5
	4	31	2	7.5	22.5
	5	63	2	15.5	46.5
4	3	85	2	12.6	63
	4	341	2	51	255
	5	1365	2	204.6	1023

First we analyze the message characteristics of TRAIL as a function of the network size. The critical resource consumption of TRAIL is given by the sizes of the attestation messages. As nodes need to accumulate nonce values of their parent nodes, the attestation array grows with increasing network sizes. While messages are tiny at the leaf nodes, the array gets larger towards the root node. Fig. 4 visualizes the average message sizes for different fanout degrees k of the inner nodes as functions of the total network size. For simplicity, we assume balanced k-ary trees, but results are not strongly dependent on tree shapes. It is clearly visible that small message sizes compliant to 6LowPAN MTUs constrain network dimensions by about ≈ 250 nodes. The characteristic performance aspects of TRAIL for different network sizes and tree configurations are summarized in Table 1, from which we can extract the extra traffic imposed by TRAIL: Two messages per node at the given size distribution.

Our second evaluation targets at the temporal performance of route convergence. We deployed TRAIL on 25 MSBA2 nodes distributed in the sensor network testbed and compared with an identical pure RPL installation. RPL/TRAIL arranged a DODAG with the highest rank of eight as visualized in Fig. 5(a). Choosing an attacker (node 60) to announce a root rank led to a break up of the pure RPL network. Only seven



(a) EXPERIMENTAL DODAG WITH ATTACKER



(b) ROUTING CONVERGENCE TIMES IN RANK ORDER

Figure 5. Per node performance in joining the DODAG: Pure RPL and the TRAIL overhead as observed in the testbed.

nodes remained in the initial, upright network, while 17 nodes reconnected to the bogus tree. TRAIL discovers and isolates the attacker immediately with its (bogus) rank announcement. As a consequence, TRAIL rearranged a connected tree of all honest neighbors, excluding the bogus node.

Routing convergence times for tree construction were measured node-wise during the experiments. Comparisons between the pure RPL and the overhead induced by TRAIL are plotted in Fig. 5(b). Naturally, the wireless ad-hoc regime produces large variations that are visible for both, RPL and TRAIL. Nevertheless, the additional times needed to join a DODAG while performing the security extensions of TRAIL remain below 20 % in most cases. Occasional authentication messages exceed this limit due to message loss and retransmissions. However, such performance fluctuations are characteristic for all mesh routing operations including RPL.

5 Conclusions & Outlook

This work focuses on routing security of RPL, a recent routing protocol for the emerging Internet of Things. Intrinsically, RPL is vulnerable to topology attacks and the current state of the art leaves relevant security issues unresolved.

The contribution of this paper is TRAIL. TRAIL defines a test procedure to inquire on the actual path properties of the routing system. This generic approach is built on first-hand principles and – different from previous work – requires almost no cryptography. Its main cryptographic workload is carried out by the root node, which acts as a (stronger) gateway in typical RPL deployments. TRAIL is designed to minimize network message exchanges and node resource consumption. Our evaluations revealed that the transmissions of bits required by TRAIL remain feasible for typical challenged environments, and that a testbed of typical shape can well operate TRAIL with limited additional effort. Future directions of this work are twofold. First, we will optimize our algorithms further to reduce dependency on network sizes. Second, we intend to apply the TRAIL approach proposed for RPL to other routing protocols.

6 References

[1] T. Winter, P. Thubert, A. Brandt, J. Hui, R. Kelsey, P. Levis, K. Pister, R. Struik, J. Vasseur, and R. Alexander, "RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks," IETF, RFC 6550, March 2012.

[2] A. Mayzaud, A. Sehgal, R. Badonnel, I. Chrisment, and J. Schönwälder, "A Study of RPL DODAG Version Attacks," in *Monitoring and Securing Virtualized Networks and Services*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2014, vol. 8508, pp. 92–104.

[3] A. Dvir, T. Holczer, and L. Buttyan, "VeRA - Version Number and Rank Authentication in RPL," in *IEEE 8th International Conference on Mobile Adhoc and Sensor Systems (MASS)*, Oct. 2011, pp. 709–714.

[4] T. Tsao, R. Alexander, M. Dohler, V. Daza, A. Lozano, and M. Richardson, "A Security Threat Analysis for the Routing Protocol for Low-Power and Lossy Networks (RPLs)," IETF, RFC 7416, January 2015.

[5] O. Gaddour and A. Koubâa, "RPL in a nutshell: A survey," *Elsevier Computer Networks*, vol. 56, no. 14, pp. 3163–3178, Sep. 2012.

[6] T. Markmann, T. C. Schmidt, and M. Wählisch, "Federated End-to-End Authentication for the Constrained Internet of Things using IBC and ECC," in *Proc. of ACM SIGCOMM, Poster*. 2015, pp. 603–604.

[7] J. R. Douceur, "The Sybil Attack," in *Revised Papers from the First Intern. WS on P2P Systems*. London: Springer, 2002, pp. 251–260.

[8] A. Mayzaud, R. Badonnel, and I. Chrisment, "A Taxonomy of Attacks in RPL-based Internet of Things," *International Journal of Network Security*, vol. 18, no. 3, pp. 459–473, May 2016.

[9] A. Le, J. Loo, Y. Luo, and A. Lasebae, "Specification-based IDS for securing RPL from topology attacks," in *IFIP WD*, Oct. 2011, pp. 1–3.

[10] K. Weekly and K. Pister, "Evaluating Sinkhole Defense Techniques in RPL Networks," in *Network Protocols (ICNP)*, Nov. 2012, pp. 1–6.

[11] L. Wallgren, S. Raza, and T. Voigt, "Routing Attacks and Countermeasures in the RPL-based Internet of Things," *Int. J. of Distributed Sensor Networks*, vol. 2013, no. 794326, June 2013.

[12] A. Mayzaud, A. Sehgal, R. Badonnel, I. Chrisment, and J. Schönwälder, "Mitigation of Topological Inconsistency Attacks in RPL-based Low-power Lossy Networks," *International Journal of Network Management*, vol. 25, no. 5, pp. 320–339, 2015.

[13] M. Landsmann, H. Perrey, O. Ugus, M. Wählisch, and T. C. Schmidt, "Topology Authentication in RPL," in *Proc. of the 32nd IEEE INFOCOM. Poster*. Piscataway, NJ, USA: IEEE Press, Apr. 2013.

[14] H. Perrey, M. Landsmann, O. Ugus, M. Wählisch, and T. C. Schmidt, "Topology Authentication in RPL," Open Archive: arXiv.org, Technical Report arXiv:1312.0984, December 2013, 2015.

[15] B. H. Bloom, "Space/Time Trade-offs in Hash Coding with Allowable Errors," *Commun. ACM*, vol. 13, no. 7, pp. 422–426, July 1970.

[16] P. S. Almeida, C. Baquero, N. Preguiça, and D. Hutchison, "Scalable Bloom Filters," *Inf. Process. Lett.*, 101 (6), pp. 255–261, March 2007.

[17] M. Mitzenmacher, "Compressed Bloom Filters," *IEEE/ACM Trans. Netw.*, vol. 10, no. 5, pp. 604–612, Oct. 2002.

[18] E. Baccelli, O. Hahm, M. Günes, M. Wählisch, and T. C. Schmidt, "RIOT OS: Towards an OS for the Internet of Things," in *Proc. of the 32nd IEEE INFOCOM. Poster*. IEEE Press, 2013.

[19] M. Günes, B. Blywis, F. Juraschek, "Concept and design of the hybrid distributed embedded systems testbed," FU Berlin, TR-B-08-10, 2008.