Hochschule für Angewandte Wissenschaften Hamburg
*Hamburg University of Applied Sciences*

# Towards a Secure RPL

- Ground Truth -

Martin Landsmann

AW2 - Report

*Fakultät Technik und Informatik*
*Studiendepartment Informatik*

*Faculty of Engineering and Computer Science*
*Department of Computer Science*

Martin Landsmann

# AW2 - Report

Eingereicht am: July 28, 2013

**Martin Landsmann**

**Thema der Arbeit**
AW2 - Report

**Stichworte**
LLN, RPL, 6LowPAN, Routing, Internet of Things, IPv6, Sensorknoten, IT-Sicherheit

**Kurzzusammenfassung**
The *IPv6 Routing Protocol for Low-Power and Lossy Networks* (RPL) has recently been introduced by the IETF as the new routing standard for the *Internet of Things* (IoT). It is organized in a hierarchical graph, optimised for routing towards a root node. A node in such a *Destination Oriented Directed Acyclic Graph* (DODAG) advertises its distance to the root node, its rank, to all adjacent nodes. Joining a DODAG, a node selects parents and announces a higher rank producing a fully connected DODAG with unambiguous hierarchical relations. Although RPL defines basic security modes, it remains vulnerable to topological attacks, which facilitate blackholing, interception, and resource exhaustion.

**Martin Landsmann**

**Title of the paper**
AW2 - Report

**Keywords**
LLN, RPL, 6LowPAN, Routing, Internet of Things, IPv6, Sensor-nodes, IT-Security

**Abstract**

The *IPv6 Routing Protocol for Low-Power and Lossy Networks* (RPL) has recently been introduced by the IETF as the new routing standard for the *Internet of Things* (IoT). It is organized in a hierarchical graph, optimised for routing towards a root node. A node in such a *Destination Oriented Directed Acyclic Graph* (DODAG) advertises its distance to the root node, its rank, to all adjacent nodes. Joining a DODAG, a node selects parents and announces a higher rank producing a fully connected DODAG with unambiguous hierarchical relations. Although RPL defines basic security modes, it remains vulnerable to topological attacks, which facilitate blackholing, interception, and resource exhaustion.

# Contents

# 1 Introduction

This report introduces and discusses security approaches for the routing protocol for low power and lossy networks (RPL)[1]. The first part gives a brief introduction about the operation components of RPL, required as knowledge foundation in this report, and introduces their related vulnerabilities.
The main part of this report focus on related work on securing RPL. In this part three works that addressed the security issues of RPL are discussed.
Finally a summary on the results of the introduced works is given and potentials and challenges of coming work.

**History** The ROLL working group of the Internet engineering task force (IETF) made a survey in mid 2009 on existing routing protocols for low power and lossy networks [2]. They identified the requirements for such a routing protocol and had a deeper look into existing standardised protocols. As a result of the survey, it has been shown that none of the examined protocols satisfied the specified requirements.
This was the basis for a specification of a new routing protocol for low power and lossy networks that meets all the collected requirements.
The IETF started to elaborate the new protocol in the same year which finally resulted in the RFC6550, defining RPL in early 2012.

## 1.1 RPL basics

RPL is built on top of the IPv6 [3] variant for constrained network devices (6loWPAN) [4]. It is a hierarchical converge cast protocol, with the main traffic pattern from the nodes towards the sink or root node. The topology is constructed proactive while fail and error handling is performed reactively.

**Node joining** When a node joins the RPL topology it first listens to advertisements of adjacent nodes offering routes towards the root. These hello or information messages are distributed periodically from every node to the adjacency.

Each advertisement contains the information about the distance to the root, the rank.
A joining node chooses a beneficial node as parent for routing, i.e. a node with a possibly low distance to the root.
It increments the parent node's rank by one, and takes it as own distance to the root. In turn it advertises this information periodically.
Eventually a hierarchical topology constructs in this proactive manner. This procedure forms a routing graph without the need that a node has to care of the child and further distant nodes.

**Routing** Forwarding of traffic upwards to the root causes consistency checks on every node that routes packets. Due to the topology construction, every node can determinate if a message is forwarded to the right direction, which must be always upwards towards the root node. One exception are special marked messages allowing routing in opposite direction. These types are not considered in this report. So, if a message has been forwarded to a node it has to been sent from a node with greater rank than the receiving node.
If the rank reduction is violated when forwarding, a node determinate a path inconsistency. This simple, yet effective check, prevents or at least detects appeared topology loops. If such inconsistency occurs, the involved nodes try to resolve it locally.
This immediately performed reactive local repair mechanism between the involved nodes rearranges the rank relations and paths of them.

### 1.1.1 Vulnerabilities

**Global repair** Tying on the last paragraph, if the number of inconsistencies exceeds the possibility to resolve them locally, or if the topology is degenerated due to local repairs, the root node can designate a global reconstruction of the topology. A new topology identifier, i.e. the increased "topology" version number, is then distributed by the root node. Upon reception it causes every node to drop all routes and rejoin the topology, which recreates a new graph. This global repair mechanism, that affect all nodes, is obviously vulnerable and needs to be protected against deliberate misuse.

**Rank exploitation** The next vulnerability originates in the fact that nodes try to have a parent with possibly low rank. So if a node advertises a deliberate low rank it can pull a major amount of routes and traffic. This can lead to a situation where traffic can not be forwarded towards the root. It can happen if a node takes such a deliberate low rank, that it only has incoming and no outgoing routes. Or if it just drops the received messages not forwarding them.

# 2 VeRA - Version Number and Rank Authentication

The authors of VeRA [5] elaborated and discussed approaches to provide protection against malicious version number updates, generating global repairs, and deliberate rank announcements. Both properties have been protected in a way that each node can verify their correctness.

## 2.1 Version number protection approach

Their method to protect the version number and their updates uses a reverse hash chain approach. Each element of this chain represents a verifiable secret, that is linked with the numeric version number. The hash chain is constructed as follows, the root chooses a random number $r$ , and computes a hash of it. The resulting hash is again hashed and then again and so forth, until we hashed $r$ , $n$ times. Finally we have a hash chain of $n$ elements, that is:

$$V_i = h^{n+1-i}(r)$$

The reverse manner is indicated by the $V_i$. In the construction phase of the hash chain the $i$ is decreased from $n$ down to 0. This results in $V_0$, that represents the $n$ times hashed $r$.

## 2.2 Rank protection approach

The rank protection hash chain consists initially only of the first hash element. The root chooses a random number $x_i$ for each new version $V_i$ and hashes the $x_i$ once. The corresponding rank to this hash is 0, representing the root rank.
All further ranks are computed from this root rank hash element. For every rank this root rank element is hashed the number of times of the rank distance to the root rank $j$, that is:

$$R_{i,j} = h^{j+1}(x_i)$$

$R_{i,j}$ and the numeric rank $j$ are distributed by each node in their periodic route advertisements as rank information.

The VeRA approach provides a reverse hash chain for each version number, and a procedure to link a numeric rank to a corresponding hash chain element. For the protection purposes, the following message is disseminated through the topology during the built up phase:

$$\langle V_0, MAC_{V_1}(R_{1,l}), \{V_0, MAC_{V_1}(R_{1,l})\}_{sign}\rangle$$

This bootstrap message consists of two parts. On the left side we have the initiating information required by a node to perform the verification procedures, and a signature over them on the right to secure authenticity. The components of the bootstrap message are from left to right:

- $V_0$, the $n$ times hashed random number $r$ representing the initial version hash chain element
- a Message Authentication Code (MAC) [6] taken from $R_{1,l}$ protected with the version hash element of the next version $V_1$
- $R_{i,l}$, the last rank hash chain element where $l$ is the number or the last or highest possible rank known to all participants in advance
- the signature $\{...\}_{sign}$ over the above elements verifies the root as originator of the bootstrap message

Each node verifies the signature and keeps the containing components $V_0$ and the $MAC_{V_1}(R_{1,l})$ from the bootstrap message.

### 2.2.1 Version protection verification

At each successive version update after the bootstrap, the root distributes this topology reconstruction message when enforcing a global repair:

$$\langle V_i, MAC_{V_{i+1}}(R_{i+1,l})\rangle$$

With the verified information from the bootstrap message, a node can check if the version update is originated from the root. Hashing the new version hash element $V_i$, $i$ times must result in $V_0$, which verifies the originator of the global repair as root.

### 2.2.2 Rank protection verification

The received version update message contains the MAC for the next version update. After the verification of $V_i$, the containing MAC is kept by the node. Then $V_i$ is used as key for the MAC received in the previous version update. Using the MAC with $V_i$ as key, a node can verify the rank $j$ and the corresponding rank hash element $R_{i,j}$ of a parent node.
First the node calculates the distance of the parent's claimed numeric rank and the last or highest possible rank $l$. It hashes the claimed rank hash element of the parent the calculated number of times. Using $V_i$, the node can create a MAC over this element and check if it matches the MAC received in the previous version update, that is:

$$MAC_{V_i}(R_{i,l}) == MAC_{V_i}(h^{l-j}(R_{i,j}))$$

Due to the trust propagation from the bootstrap message the previously received MAC is trustworthy.
After the parent rank has been verified, the child node accepts the parent, and hashes the parent's rank hash element once to take the resulting rank hash as own.

$$R_{i,j+1} = h(R_{i,j})$$

### 2.2.3 Conclusion of the VeRA approach

**Version number update protection** It relies on the signed first reverse hash chain element $V_0$ from the bootstrap message. To break the trust propagation an attacker needs to either guess the next version element $V_{i+1}$ in the current version $V_i$, or try to compute the element which would break the one way manner of the used hash function.

**Rank authentication** The approach provides a node with the knowledge to be able verifying the claimed rank $j$ of a parent node. To break the rank hash chain procedure, an attacker has to obtain an early rank element such as $R_{i,0}$. This would provide the necessary information to compute all successive rank hash elements. To forge an own verifiable rank hash chain $R'_{i,0}$, the attacker needs the knowledge of the next version hash element $V_{i+1}$. With this knowledge, forging a verifiable MAC for the current version is possible.

# 3 Evaluating Sinkhole Defense Techniques in RPL

The authors elaborated and evaluated a sinkhole detection and prevention mechanism [7] It provides protection against malicious packet drop, i.e. a sinkhole attack. They used a VeRA based approach to prevent an attacker from deliberate choosing a low rank and pulling routes for a malicious packet dropping. This mitigated the impact of sinkhole attacks originated from topological constellations. Additionally they elaborated the parent fail-over approach to detect sinkholes.

## 3.1 Sinkhole defense technique - parent fail-over

The root node expects an amount of delivered messages from all nodes in the topology. If the quantity of messages falls below a threshold, the node is recorded into the unheard nodes set (UNS). The authors note that the lower limit of the UNS has to be carefully chosen. A low threshold limit would rise the false detection rate. Contrary a high threshold limit would weaken the protection.
Every node falling under the delivery threshold ends up in the UNS. This list of missing nodes is signed by the root node, and periodically distributed in the topology. The UNS distribution is distinct to the topology traffic. It is performed using the adjacent or neighbour connection of the nodes. Upon reception of an UNS, a node verifies the signature and checks if it is scribed into the list. If the node finds itself listed, it drops the parent node, blacklists it, and chooses a different parent for routing.

## 3.2 Conclusion of this work

The parent fail-over technique successfully detects and prevents sinkole attacks and attacking nodes. The authors evaluated roundly their approach, and have shown that it can significantly improve the delivery performance of the topology.
Additionally they discovered a rank replay vulnerability in VeRA.

# 4 Topology Authentication in RPL (TRAIL)

The authors identify and provide a fix for a new rank vulnerability in VeRA. Additionally an approach to protect VeRA against rank replay attacks is introduced [8, 9]. Finally they present a new approach to protect the routing hierarchy in RPL distinct to VeRA.
This chapter focus on the first two contributions, and briefly introduces the base idea of TRAIL [9], that is a work in progress approach.

**Facts on the vulnerabilities in VeRA** The validity of the last rank hash element $R_{i,l}$ of the current version is protected with a MAC using the current version element $V_i$ as key. A version update message containing $V_i$, also contains the successive version MAC, $MAC_{V_{i+1}}(R_{i+1,l})$. To compute an authentic MAC over a forged rank hash chain the successive version hash element $V_{i+1}$ is required.
The security of the scheme relies completely on this version element as key, which is an absolute sufficient condition in the cryptographic context. The trust from the signed bootstrap message is not propagated to $R_{i,l}$ during successive global repairs. Only the first rank hash chain has a direct correlation to this message.

## 4.1 Attacking VeRA

A malicious node receives a version update message from the root, to switch from $V_{i-1}$ to $V_i$, i.e. $\langle V_i, MAC_{V_{i+1}}(R_{i+1,l})\rangle$.
The malicious node does not distribute the received version update message to its neighbours and children. Possibly it attempts to disturb neighbouring performed distributions to hide the version update message from propagation. Then the node waits until the next version update, receiving the successive information $\langle V_{i+1}, MAC_{V_{i+2}}(R_{i+2,l})\rangle$ from the root withholding its distribution again.
From this moment on, the malicious node exploited all information and keys to perform a delayed rank forgery attack. That is, it first creates a deliberate rank hash chain $R'_{i,l}$. Then it uses the last received version hash element $V_{i+2}$ to create an authentic MAC over the forged rank hash chain, $MAC_{V_{i+2}}(R'_{i,l})$.

It distributes $\langle V_{i+1}, MAC_{V_{i+2}}(R'_{i,l})\rangle$ to cause its child nodes to switch to the next but delayed version $V_{i+1}$. This breaks the security of the rank hash chain and the protective MAC. At the next version update from $V_{i+1}$ to $V_{i+2}$, the malicious node can compute and claim all ranks for its own.
The described procedure can be repeated on every genuine successive version update.

### 4.1.1 Protecting VeRA against rank forgery

The authors propose a nested rank encryption chain to recreate a trust propagation link between the bootstrap message and every rank hash chain. The change to VeRA is that all rank hash chains $R_{0,l}...R_{n,l}$ are computed for all version reverse hash chains $V_0...V_n$ in advance by the root node.
The base idea of the encryption chain is that the last but one version rank hash chain element $R_{n-1,l}$ is encrypted using the last version rank hash chain element $R_{n,l}$ as key. This symmetric encryption results in cypher $c_{n-1}$ which is used as key to encrypt the previous version rank hash chain element $R_{n-2,l}$ to cypher $c_{n-2}$.
All elements of the encryption chain envelop as follows:

$$\begin{aligned} c_{n-1} &= enc_{R_{n.l}}(R_{n-1,l}) \\ c_{n-2} &= enc_{c_{n-1}}(R_{n-2,l}) \\ &\dots \\ c_0 &= enc_{c_1}(R_{0,l}) \end{aligned}$$

It is assumed by the authors, that a strong symmetric cryptography scheme is used for the nested encryption, e.g. the Advanced Encryption Standard (AES) [6] with a sufficient number of bits.
The cypher $c_0$ is distributed in the signed bootstrap message, which is used and establishes a trust propagation anchor.
Every successive version update message contains the corresponding cypher of the successive version rank hash chain $c_{i+1}$. With this cypher, $c_i$ can be decrypted to obtain the corresponding rank hash chain $R_{i,l}$ for the current version update $V_i$. More formally:

$$\begin{aligned} R_{0,l} &= dec_{c_1}(c_0) \\ &\dots \\ R_{i,l} &= dec_{c_{i+1}}(c_i) \end{aligned}$$

### 4.1.2 Rank replay protection

Repeating or replaying a learned message or information is always possible to a node. A learned rank hash element of a parent node can be replayed by a malicious node to claim a better rank. The authors introduce a challenge response procedure to approach this vulnerability.
It forces the attacker to either obey the RPL rank rules and take a higher rank than its parent's, or to be avoided by other nodes. The procedure is based on the parent child rank relationship of RPL, with the main idea that only a proper chosen rank enables a node to solve the challenge.

Assuming two nodes. Node $N_{R_{i,j}}$, and a potentially malicious node $N_M$, which request to route traffic through $N_{R_{i,j}}$. $N_{R_{i,j}}$, wants proof for $N_M$'s rank, and creates a message containing the $ID_M$ of $N_M$, e.g. the link local address, and a nonce $r$. Then it asks a parent node of rank $R_{i,j-1}$ to encrypt the message $\langle ID_M, r \rangle$.
The parent node receives and encrypts it using the rank hash element $R_{i,j-2}$ of its own parent as key, that is the grandparent of $N_{R_{i,j}}$.
The parent node returns the cipher $c = enc_{R_{i,j-2}}(\langle ID_M, r \rangle)$ back to $N_{R_{i,j}}$. $N_M$ is then asked to decrypt the $c$ as proof of its rank. If $N_M$ has a legitimate rank it also has a parent node which is able to aid $N_M$ to decrypt $c$. If $N_M$, solves the challenge, it has proof for its rank.
If the challenge can not be solved by $N_M$, $N_{R_{i,j}}$ does not forward traffic to, and from $N_M$.

## 4.2 Distinct approach to VeRA - Trust Anchor Interconnection Loop

The approach prevents rank spoofing and provides a path validation upwards from the most distant nodes to the root, the leaf nodes. It uses a round trip message that validates a path towards the root node when it returns with a valid signature.
Nodes collect nonces $n_{c_1}...n_{c_n}$ from their child nodes when providing a route to them. The collected nonces are united in a Bloom filter [10] $B_{Rank\ j}$, which is a space efficient data structure storing the information about the existence of a scribed element.

$$B_{Rank\ j} = [n_{c_1}...n_{c_n}]$$

Each filter is forwarded to the parent node together with an own nonce. Just as before, the parent node unites all received child nonces in a new Bloom filter. Additionally all received Bloom filters are merged aligned at the beginning into a new Bloom filter.

$$\begin{aligned}&[B_{Rank\ j_1}]\\&\dots\\&[B_{Rank\ j_n}]\\=&[B_{Rank\ j}]\end{aligned}$$

The united filters are attached behind the newly created Bloom filter from the received nonces.

$$[B_{Rank\ j-1}][B_{Rank\ j}]\dots[B_{Rank\ n}]$$

Such arrangement constructs an array of Bloom filters positioned at the corresponding rank distance to the root node. In this manner, eventually every Bloom filter contains all nonces from one topological rank.

$$[B_{Rank\ 0}]\dots[B_{Rank\ j-1}][B_{Rank\ j}][B_{Rank\ j+1}]\dots[B_{Rank\ n}]$$

Upon reception, the root node signs this array, and multicasts it back in the topology. Receiving it, a node verifies the signature and checks if the previously scribed nonce is present in the array element of the node's rank. If the filter contains the nonce the path is validated and can be trusted to forward traffic.

## 4.3 Conclusion and Outlook

The version number update is protected successfully by VeRA, propagating the trust of a signature in a reversed hash chain. The rank spoofing vulnerability can be handled with the repaired VeRA approach, using nested encryption chains with direct relation to the signed bootstrap message. The TRAIL approach introduces a distinct way protecting against the rank spoofing, providing a round trip message validating all ranks at once. The sinkhole protection is addressed by the parent fail-over technique which detects and eventually isolates sinkholes, by tracking a message delivery threshold. Either the repaired VeRA approach, or TRAIL prevents sinkholes in a topological context.

The presented approaches are still vulnerable against wormhole attacks. Providing knowledge out of band with collaborating attackers enables them to bypass the securing mechanisms.

# Bibliography

[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] S. D.-H. P. Levis, A. Tavakoli, “Overview of Existing Routing Protocols for Low Power and Lossy Networks,” IETF, Internet-Draft draft-ietf-roll-protocols-survey-07, April 2009. [Online]. Available: http://tools.ietf.org/html/draft-ietf-roll-protocols-survey-07

[3] S. E. Deering and R. M. Hinden, “Internet Protocol, Version 6 (IPv6) Specification,” IETF, RFC 2460, December 1998.

[4] G. Montenegro, N. Kushalnagar, J. Hui, and D. Culler, “Transmission of IPv6 Packets over IEEE 802.15.4 Networks,” IETF, RFC 4944, September 2007.

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

[6] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone, *Handbook of Applied Cryptography*.

[7] K. Weekly and K. Pister, “Evaluating Sinkhole Defense Techniques in RPL Networks,” in *Network Protocols (ICNP), 2012 20th IEEE International Conference on*, Nov. 2012, pp. 1–6.

[8] M. Landsmann, H. Perrey, O. Ugus, M. Wählisch, and T. C. Schmidt, “Topology Authentication in RPL,” in *Proc. of the 32nd IEEE INFOCOM. Poster.* Turin, Italy: IEEE Press, 2013, accepted for publication.

[9] [redacted].

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

*Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.*

Hamburg, July 28, 2013 Martin Landsmann