Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

On Secure Routing in Low-Power and Lossy Networks: The Case of RPL

Master-Thesis

Heiner Perrey

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Heiner Perrey

On Secure Routing in Low-Power and Lossy Networks: The Case of RPL

Masterarbeit eingereicht im Rahmen der Masterprüfung

im Studiengang Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Thomas C. Schmidt
Zweitgutachter: Prof. Dr. rer. nat. Gunter Klemke

Eingereicht am: August 20, 2013

Heiner Perrey

Thema der Masterarbeit

On Secure Routing in Low-Power and Lossy Networks: The Case of RPL

Stichworte

RPL, sicheres Routing, Sensornetzwerke, Hash-Ketten, Topologie Authentifizierung

Zusammenfassung

Diese Masterarbeit beschäftigt sich mit der Sicherheit des IPv6-basierten Routing Protokoll für „low-power and lossy networks“ RPL. Die durchgeführte Sicherheitsanalyse zeigt, dass ein Angreifer ohne Zugang zu den kryptographischen Schlüsseln weitestgehend abgewehrt wird. Es werden Topologie-Attacken aufgezeigt, welche erst durch den Besitz eben dieser Schlüssel möglich werden und eine ernste Bedrohung für die Sicherheit von RPL darstellen. Im Zuge dessen werden zwei Verfahren zur Authentifizierung der Topologie vorgestellt: VeRA und TRAIL. VeRA nutzt Hash-Ketten, um die Topologie zu sichern. TRAIL hingegen setzt auf das Hin- und Zurücksenden einer Nachricht, die Knoten für Knoten die Topologie verifiziert. Mit Hilfe der konkreten Umsetzung von TRAIL werden erste Studien zur Zeitverzögerung, die durch dieses Verifizierungsverfahren entsteht, durchgeführt.

Title of the Master-Thesis

On Secure Routing in Low-Power and Lossy Networks: The Case of RPL

Keywords

RPL, routing security, sensor networks, hash chaining, topology authentication

Abstract

This master-thesis engages in the security aspect of the newly introduced IPv6-based routing protocol for low-power and lossy networks RPL. In an analysis of the security of RPL, it is shown that RPL protects against most attacks launched by an adversary that does not have access to the security keys. The major threat lies within an attacker who has compromised these keys and who is consequently able to launch topology attacks. Two topology authentication schemes are presented and evaluated which deal with the prevention of such attacks: VeRA and TRAIL. The VeRA approach employs hash chains for verification of the topology of RPL. TRAIL relies on the transmission of a round-trip message that verifies the topology hop-by-hop. It is shown that these schemes can detect and isolate an insider attacker. A practical evaluation of TRAIL gives a first impression of the delays that result from the verification procedure.

Contents

List of Tables

List of Figures

1 Introduction

It's a small world. This is particularly true when looking at the progress in the world of sensor networks. Devices, ever-decreasing in size, price and capabilities, can form large-scale networks that sense and collect environmental information and are increasingly able to make this information available world-wide. For this purpose, researchers have been focusing on attaching these devices to existing networks that use the widely deployed Internet Protocol (IP). This interconnection enables devices to easily communicate using the existing and well-established infrastructure from virtually anywhere in the world. The introduction of IPv6 [1] and its giant address space makes it possible to supply billions of small devices with an unique identifier and thus allows these devices to function in an *Internet of Things*.

However, this world is also dangerous. Especially when constrained devices communicate publicly and wireless, security becomes an important and challenging task. Various deployment scenarios bear the risk of adversaries intruding from within or outside the network and damaging the network operability, driven by profit, political affection or simply just for fun. Countering these threats involves the research of lightweight security schemes, as not only the complexity of applications, protocol stacks and communication patterns have to be adapted to the upcoming constraints, but also the security models. Well-known and widely accepted cryptographic protocols may exceed the limits of most low-power devices. At the same time security becomes more important than ever when these devices monitor or control critical systems such as in smart grids or for medical purposes.

The infrastructure that is required for connecting arbitrary devices with one another needs a routing protocol to efficiently send the data to their destination. Hereby a standardized solution allows a reliable interoperability of these devices in different routing domains. The *Routing over Low-Power and Lossy Networks* (ROLL) working group of the IETF[1] concerned itself with the requirements for such a routing protocol [2]. Their survey concluded that the routing protocols they examined, such as AODV [3] or OLSR [4] do not satisfy the requirements they defined as suitable for LLNs. One of the major milestones of their work led to the standardization of the *IPv6 Routing Protocol for Low-power and Lossy Networks* or RPL [5]. RPL

[1] IETF: Internet Engineering Task Force

has primarily been designed for constrained devices and for data collection at a dedicated sink. The IPv6 compliance allows devices to attach to existing networks and makes RPL a candidate for the standard routing protocol for networks of smallest devices or the very *things*.

This thesis is concerned with the important aspect of security in RPL. RPL prepares for various threats and provides a security model to defend against potential attacks. Although security mechanisms may be implemented at the link layer, RPL is not tied to an underlying protocol and thus the security model is defined in case of absence of link layer security. So far the security of RPL has not been analyzed thoroughly in the literature. Hence, the main contribution of this work is threefold. First, a comprehensive security analysis evaluates the security model of RPL. The goal is to show strengths and weaknesses of this model and to highlight specific attacks. Hereby the main focus lies on topology attacks that greatly influence the routing operations. Secondly, two approaches are presented that mitigate such topology attacks and account for the constraint environment in which RPL typically functions. The existing approach VeRA [6, 7] uses hash chains to protect against topology attacks. VeRA shows vulnerabilities for which countermeasures are proposed [8]. The newly introduced TRAIL [9] follows the same goal and requires a certain communication overhead. Both approaches are then analyzed and compared in the course of this work. Thirdly, the feasibility of TRAIL is practically evaluated in a proof-of-concept implementation.

For a thorough evaluation of the RPL security and for a basic understanding of the topology authentication schemes, a fundamental background is required. Therefore, the remainder of this work is structured as follows. The necessary background to which this work refers to is described Chapter 2. A security analysis of the existing security model of RPL as well as attacks and countermeasures is given in Chapter 3. The VeRA approach as well as attacks on VeRA and novel countermeasures are presented in Chapter 4. The new topology authentication scheme TRAIL is detailed and analyzed in Chapter 5 in combination with a practical evaluation in Chapter 6.

2 Background

This chapter provides the required background for a basic understanding of the applied concepts and gives an introduction of the examined routing protocol RPL.

2.1 Basic Terms of Information Security

Information security comprehends the protection of information according to specific security objectives. The implementation of these objectives is achieved, for instance, by cryptographic protocols.

2.1.1 Security Objectives

Security objectives define elements or properties of a system that deserve protection. Typically, security objectives are classified by the property that they protect. Common objectives include authenticity, integrity, confidentiality, availability and non-repudiation which are introduced in the following.

- **Authenticity**: Authenticity denotes the trustworthiness and genuineness of the origin of a message [10, 11]. Hence, authenticity establishes trust between routing peers and allows the receiver to verify if a sender is actually which it claims to be. Typically, authenticity is achieved by message authentication schemes using a secret key. Hereby, the secret knowledge of a key creates a trust relationship, because only an owner of the secret key is able to create an authenticated message.
- **Integrity**: Message integrity means that a message arrives at its destination without exhibiting any deliberate changes during transfer [10, 11]. Integrity checks are applied to detect such changes. Generally, message authentication schemes also involve integrity checks, so that both security objectives are provided by an authentication scheme.
- **Confidentiality**: Confidentiality denotes the concealment of sensitive information which is only meant for authorized access [10, 11]. Encryption schemes are used to prevent sensitive information from being revealed to an outsider without permission.

- **Availability**: The availability of services is important for a system to perform its operations when required [10]. Hence, the access to information or resources must be provided within a defined time period. Redundancy is a common method of providing availability of a given service.

- **Non-Repudiation**: Non-repudiation ensures that the creator of a message cannot deny being the message originator at later time [10, 11]. Digital signatures may provide non-repudiation where the signer owns the secret key and the corresponding verification is publicly known.

2.1.2 Cryptographic Tools

Security objectives can be implemented by applying cryptographic tools such as encryption or message authentication schemes. This section gives an overview of the cryptographic tools that are used in the work.

One-way hash functions A *one-way hash function* maps a (large) input set of elements with variable length into a (small) co-domain of elements with fixed length [11]. The one-way property allows to efficiently compute the hash value of an element, while the reverse calculation from a hash value to the original element is computationally hard. Hash functions play an important role in cryptography like for authentication schemes and integrity checks.

Hash functions can also be applied recursively to their output to create a hash chain. A random number x is used as seed for a hash function $h(\cdot)$. The output of $h(x)$ is hashed again, so that the second output is denoted by $h(h(x))$ or $h^2(x)$. If this is done i-times, the end of the hash chain results in $h^i(x)$. If any element of the chain is known, it is possible to compute any further element up to the end of the chain by performing the required number of hash operations. However, it is infeasible to create any prior element due to the one-way property of the hash function. Such a hash chain is useful to prove the ownership of secret information without revealing the secret.

An example application for password authentication is given by Lamport [12]: A client and a server communicate over an insecure channel. The client creates a hash chain using its password as seed $h^i(password)$ and securely provides the server with the end of the hash chain. Each time the client wishes to authenticate itself to the server, it sends the $i-1$th hash chain element to the server. The server hashes the element one more time and checks if $h(h^{i-1}(password)) = h^i(password)$. If both hash chain elements match, the server accepts the authentication. Next time the client sends an authentication, it sends the $i-2$nd element,

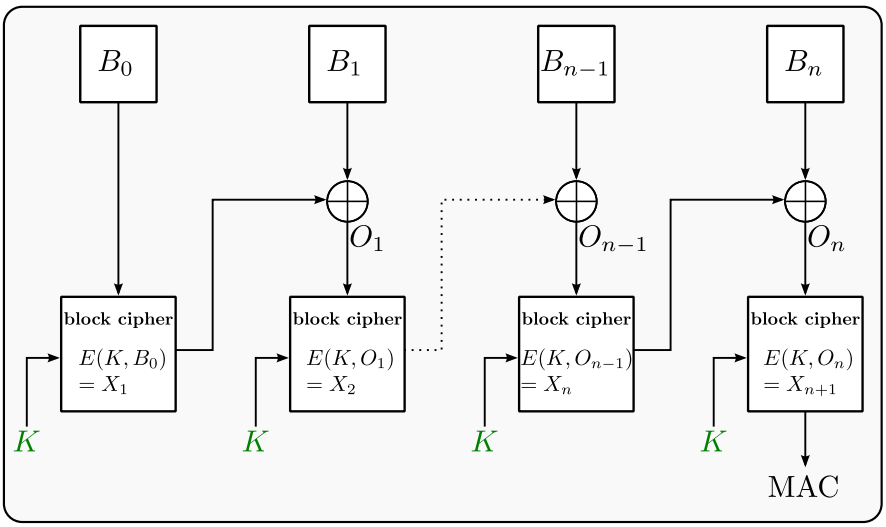


Figure 2.1: Creation of a CBC-MAC in CCM – The first block B_0 is encrypted by a block cipher function E using secret key K. The resulting X_1 is XORed with the second block, B_1. The resulting output O_1 in turn is encrypted, and so on. The last (truncated) block denotes the MAC.

so that the server performs two more hash operations and so on. An adversary that replays a recorded hash element is detected since the element has already been used.

Message authentication codes Hash functions are also applied within other cryptographic constructs such as the creation of a *message authentication code* (MAC). A MAC is used to authenticate the originator of a message and to check its integrity. MACs are based on a shared secret key and are thus created by symmetric protocols. The sender uses the secret key to create a MAC of a message and transmits both messages and MAC to the receiver. The receiver also creates the MAC of the message and compares its own computation to the MAC it received. If both match, the receiver has verified that the message has been created by a key holder and that it has not been modified.

The CBC-MAC (cipher block chaining MAC) [13] and HMAC (keyed-hash MAC) [14] are examples to create such MACs. HMAC appends a shared secret to the message and hashes the concatenation with a cryptographic hash function. Hence, only a secret holder creates a valid MAC. To create a CBC-MAC as shown in Figure 2.1, the message is encrypted in the cipher block chaining mode of operation. The message is split into n blocks of equal size. An initial block contains control flags, a nonce and the length of the message to support variable size messages. This block is encrypted by a block cipher, like AES, applying a secret key. The

resulting cipher is XORed with the second block which denotes the first block of the actual message. The XOR conjunction is encrypted and then XORed with the next block, and so on. The last block denotes the MAC which may be truncated to the desired length.

Symmetric authenticated encryption The CBC-MAC is applied within the CCM mode of operation [13] which denotes a *symmetric authenticated encryption* scheme and is further considered in this work. CCM stands for *Counter with CBC-MAC* which uses a block cipher both for encryption in counter mode and the creation of a CBC-MAC. The CBC-MAC provides authenticity and integrity, while the encryption ensures the confidentiality of the message content.

For encryption the plaintext is divided into blocks of equal size. To each plaintext block a unique key stream is applied by creating the XOR conjunction. To create such distinct key streams, a CCM nonce consisting of a random part and a counter is encrypted using a secret key. For each plaintext the counter can simply be incremented to provide a unique key stream. Authentication is achieved by the creation of a CBC-MAC of the plaintext message using the same secret key. To protect against collision attacks, the MAC is encrypted together with the plaintext message. Furthermore, CCM provides *additional authenticated data* for the authentication of unencrypted information such as routing headers. For this purpose, an additional MAC of this unencrypted information is created but not encrypted.

CCM requires each nonce to be used only once with a given encryption key, so that key streams are not reused [13]. If used twice or more, an attacker simply reveals the XOR-conjunction of two plain texts P_1 and P_2 by combining two cipher texts that were created with the same key stream K_i:

$$(K_i \oplus P_1) \oplus (K_i \oplus P_2) = (P_1 \oplus P_2)$$

He may be able to extract both P_1 and P_2 and thus receives the applied key stream:

$$(K_i \oplus P_1) \oplus P_1 = K_i = enc_{K_s}(\eta_i)$$

where K_i is the secret key stream, K_s is the secret key and η_i is the applied nonce. Each further message using this nonce-key pair is easily decrypted.

Public-key cryptography In contrast to symmetric approaches in which each party holds the same key, a *public-key scheme* is based on asymmetric keys. Hereby, different keys for en- and decryption are used rather than a single secret key [11]. The receiver therefore creates a

key pair. One key is private and kept secret by the owner. The other is made publicly available and thus called public key. The public key is used for encryption and the private key for decryption, so that anyone can encrypt a message, but only the holder of the private key is able to decrypt the message.

The asymmetric scheme relevant for this work is RSA [15] which can be used for encryption and for the creation of *digital signatures* for authentication. To create a digital signature using RSA, each sender signs its messages before transmission. Signing with RSA is done by computing the hash value of the entire message and signing it by encryption with the private key. Hence, the cipher denotes the signature which is appended to the message. A node that receives a signed message first verifies the signature. This is done by computing the hash value of the entire message and decrypting the signature using the public key of the sender thus revealing the hash value. If both hash values match, the message has not been altered but was created by the owner of the private key.

2.1.3 Key Management

Cryptographic tools are used to implement the security objectives. Besides hash functions, all of the presented tools require security keys. To provide and refresh key materials either a manual or automated key management is deployed [16]. Manual key management requires re-keying by hand upon expiration of security keys. The introduction of an automated key management scheme allows dynamic key exchange and re-keying which is required by a large-scale network that employs cryptography [16]. According to Trappe and Washington [17], the following key management approaches are distinguished:

- pre-shared key
- key agreement
- key distribution

A *pre-shared key* is known to all participants prior to communication. Each member of the network is configured out-of-band with a key. In the simplest case, the entire network shares the same symmetric key. Such a group-wise shared key has the drawback of a compromised key breaching the security for the entire group. To mitigate this impact, pair-wise keys may be applied. Each member thus shares a unique key with any other member that it wishes to securely communicate with. The advantage of a pair-wise key management scheme is that a single comprised key does not affect the security of remaining members of the network.

In contrast to a pre-shared key, a *key agreement protocol* like the Diffie-Hellman protocol [18] establishes a symmetric key between two communicating partners. Diffie-Hellman does not require any information to be shared amongst both participants prior to the key agreement. The secret key is produced as result of their communication and only known to the participants [17]. However, such an approach is not suited for group communication, as a key is established only between two members.

In a *key distribution* scheme the security keys are distributed by a key server. This key distribution server is equipped with all secret keys which are distributed to clients when required. To secure the communication a pre-shared key or asymmetric cryptography can be used.

2.2 Bloom Filters

A Bloom filter [19] is a probabilistic data structure for memory-efficient storage of elements. Bloom filters only store a representation and not the element itself, thus allowing a quick execution of membership queries. A query either returns a definite *no* or a *maybe*. Hence, falsely receiving a negative reply is not possible. Each query, however, bears the risk of a false positive which is considered when parameterizing the filter.

A Bloom filter is defined as a bit-vector v of size m with all bits initialized to zero. By using k independent hash functions $h_1(\cdot) \dots h_k(\cdot)$ with a range of $\{0 \dots m\}$, each element of a set $A = \{a_1 \dots a_n\}$ is mapped to k bits in v. To add an element a_i to the filter, the element is hashed by each function, so that $x_j = h_j(a_i)$, with $1 \leq j \leq k$. Each x_j denotes one of k hash values of a_i and is mapped to an index in v which is set to 1. Each bit is set only once. Removing an element by setting its bits to zero is not permitted as it possibly removes other elements as well. The size of each of the input element is reduced to a maximum of k bits. Due to the overlapping of bits of different elements, a query may return a false positive result. An element a_i that is not contained in v may be represented as member if all k bits of a_i coincide with bits of other elements.

To test whether an element a_i is a member of the filter, the element is hashed in the same manner as done for the insertion into the filter. This results in the hash values $x_1 \dots x_k$, where each value represents an index of v. If any index is still set to zero, the element is definitely not contained in the filter. Else, if all bits have been set to one the element *may* be in the filter. By calculating k according to the formula $k = \frac{m}{n} * ln(2)$, f is minimized with respect to a given m and n, where n is the number of elements and m the size of the filter.

The union and intersection operation is supported by Bloom filters as well [20]. Two Bloom filters that have the same size and use the same number of hash functions can be united by using the bitwise OR operation. Hereby each set bit of either filter results in a set bit in the united filter. Alternatively, the intersection of both sets is created by performing bitwise AND operations on both original filters. Only bits that are set in both filters are also set in the resulting filter.

Plain Bloom filters are optimized with respect to the size of the filter m, the number of elements n, the number of hash functions k, and the false positive rate f. For instance, for a given m and n the number of hash functions is optimized to minimize the false positive rate. Bloom filters also allow various optimizations of which two approaches are presented:

- compressed Bloom filter
- scalable Bloom filter

Mitzenmacher [21] reconsiders the optimization problem of Bloom filters. He aims at optimizing a *compressed Bloom filter* that minimizes the bits to be transferred rather than enhancing the required storage space of the uncompressed filter in memory. Since the optimization of plain Bloom filters results in an uniform distribution of bits [21], a compression does not gain any space benefits. He shows that choosing m larger and k smaller results in a smaller compressed Bloom filter that sustains or even lowers the false positive rate. He thus achieves an optimized compressed Bloom filter that is about 30 % smaller than the plain Bloom filter with the same false positive rate.

Compressed or plain Bloom filters require that the number of elements is known prior to using the filter. To achieve more flexibility, Almeida et al. [22] propose *scalable Bloom filters*. In their approach, a Bloom filter is first chosen for a small set of elements with a certain false positive rate. When half of the bits are set, the Bloom filter is full and a new one is added that has a tighter false positive rate. A member query is performed by checking both filters sequentially. The second filter requires a tighter false positive rate as all single filters are independently queried. Take, for example, a scalable Bloom filter with a desired false positive rate of 1 %. The filter consists of a single plain Bloom filter with false positive rate $f_0 = 0.01$. If a new filter were added with a false positive rate of $f_1 = 0.01$, the probability of *not* receiving a false positive when checking for an element in both filters is $P = (1 - f_0) * (1 - f_1) = 0.9801$. The probability of receiving a false positive is therefore $1 - P = 0.0199$ and thus approximately 2 %. Choosing all Bloom filters with a tighter false positive rate lets the overall rate converge to the desired value.

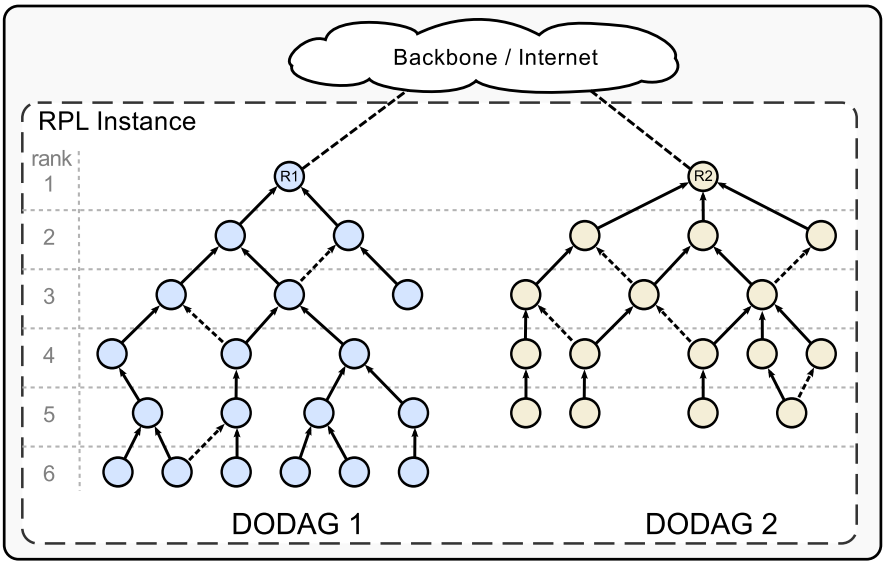


Figure 2.2: Architecture of an RPL Routing Domain – The figure shows a Directed Acyclic Graph that consists of two Destination Oriented DAGs inside an RPL instance. Both DODAGs are connected through a backbone link by their roots, $R1$ and $R2$. The position of each node is indicated by its rank.

2.3 Design Concept of RPL

This section introduces the key features of the *IPv6 Routing Protocol for Low-power and Lossy Networks* (RPL) [5]. This includes the topology creation and maintenance, communication patterns as well as the security properties that are relevant for this survey.

The main goal in the design of RPL is to provide a generic standard for various applications in LLNs. These applications include urban [23] and industrial [24] environments and home- or building automation [25, 26] which differ in terms of traffic flow pattern, security and message delivery requirements. RPL defines the general functionality of topology creation and maintenance and provides specialized and optional features in external specifications to remain applicable across many variable scenarios. An *Objective Function* (OF) defines the exact procedure by which a node selects its position in the network and allows different applications to optimize the topology formation to its specific requirements [5, p. 17].

2.3.1 Architecture of RPL

An RPL domain is constructed as a tree-like topology that consists of various nodes. The hierarchy is represented by a *Directed Acyclic Graph* (DAG). All edges of the DAG are directed

in a way, so that no cycles exist. It fragments into one or more *Destination Oriented DAGs* (DODAG) [5, p. 13], where each DODAG is rooted at a dedicated sink node that serves as data collection point [5, p. 10]. These root nodes provide connectivity between the nodes of the DODAGs through a common backbone link. Each root offers a collection point for data aggregation and functions as an *LLN Border Router* (LBR) to provide connectivity outside the RPL routing domain. Each node in the DODAG functions as a *router* or a *host*. Hosts generate but do not forward RPL traffic. A router both generates and forwards RPL traffic [5, p. 13] and has one or more child nodes. Figure 2.2 illustrates an exemplary RPL instance consisting of a DAG with two DODAGs.

The initial topology is created proactively and allows upward traffic to be forwarded along the DODAG. The root maintains configuration parameters which are distributed downwards to all nodes. Each member of the DODAG periodically transmits these parameters to its neighbors to provide updated routing information for members or to allow new nodes to join the network. A new node processes this information, selects a member as parent and calculates a *rank* which denotes its relative position in the DODAG toward other nodes and the root. The rank of a node is based on the rank of the parent and increases monotonically starting at the root node which has the lowest rank. Each node subsequently calculates a rank that is higher than each of its parents. A parent denotes a potential next-hop and upward path toward the root. The best suited parent according to chosen metrics and constraints is elected *preferred parent* which is the default next hop on an upward path [5, p. 67].

Downward routes allow the root to communicate with nodes within the network and provide connectivity between arbitrary nodes. The root knows a path to all downward destinations and thus forwards the traffic directly. In-network nodes send messages upwards first, until they reach a *common ancestor* of the source and destination which is able to redirect the traffic downwards. Messages travel downwards henceforth until they reach their destination.

Loops in the topology are detected and removed reactively during forwarding operations. A local repair mechanism increases the frequency in which control messages are sent to remove inconsistent routing states. If local repairs cannot sufficiently sustain consistency of the topology, a global repair is initiated by increasing the *DODAG version number*. The version number of a DODAG is an abstract representation of the current topology setting which may evolve due to rank changes or local repairs. If local repairs cannot sustain a consistent topology, an increased version number is propagated by the root and initiates a reconstruction of the entire topology. During such a global repair nodes can rearrange their position in the DODAG. Only the root node is permitted to increase the version number as it affects all nodes in the topology [5, p. 69].

2.3.2 Topology Creation and Maintenance

RPL defines mechanisms for the formation of the topology and repair mechanisms to recover from failure of nodes and routing loops. For this purpose, routers exchange an RPL specific type of ICMPv6[1] messages which contain the required routing information. Up- and downward routes along the DODAG are created by sending messages in the opposite direction. Upward paths toward the root are thus created by sending *DODAG Information Objects* (DIOs) downwards from the root toward the leaves. Downward routes are established by sending *Downward Advertisements Objects* (DAOs) from leaves toward the root.

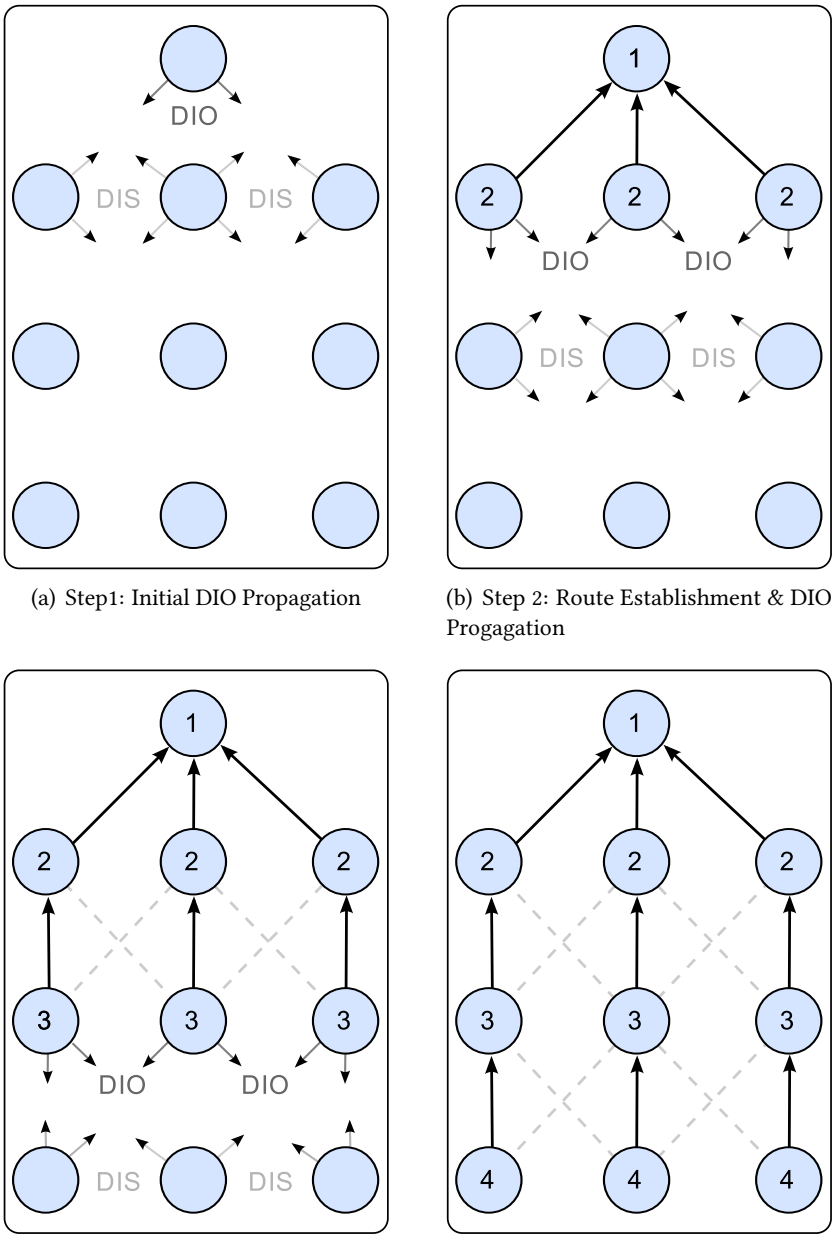
DODAG Formation A DODAG consists of a root node and an arbitrary number of nodes within the network. The exchange of control messages allows nodes to discover and maintain other neighbors and to establish connectivity to the root [5, p. 67]. The root initiates the construction of the topology by distributing configuration parameters enclosed in DIOs [5, p. 66]. To join the DODAG, a node either waits until it receives a DIO from its immediate neighbors or requests a DIO from a neighbor by sending a *DODAG Information Solicitation* (DIS) message [5, p. 115]. The DODAG successively grows as the information is propagated downwards which allows new and distant nodes to join the network. The message exchange during the topology creation is illustrated in Figure 2.3.

In the process of joining, each node subsequently selects *candidate neighbors* and *parents* [5, p. 67]. The candidate neighbors are one-hop neighbors reachable over link-local multicast. A node selects a subset of neighbors as parents which are located higher in the topology and indicate a next hop for upward traffic. Finally, each node elects a single parent to be its preferred parent which denotes the default next-hop for upward traffic [5, p. 67] (see Fig. 2.3(b) – 2.3(d)). All members of a DODAG select at least one parent by which they are attached to the root [5, p. 73] and which allows them to send upward traffic [5, p. 66]. Multiple parents offer optional routes that enable load balancing or alternative paths in case of failure [5, pp. 19, 67].

Once a node has selected a set of successors, it determines its position in the DODAG. For this purpose, the node computes its rank by adding a *step* to the rank of its preferred parent [5, pp. 20 f., 107, 108]. The size of this step is defined by the implementation and the objective function.[2] The minimum increase is denoted by the parameter *MinHopRankIncrease*. Various metrics [29] that are carried in DIOs influence the calculation of the rank which allows determination of the optimal path in terms of metrics, such as number of hops or expected

[1] ICMPv6 is the *Internet Control Message Protocol for IPv6* [27].

[2] The necessary objective functions a node requires to operate in a specific LLN are included in the implementation [5, p. 8].



(a) Step1: Initial DIO Propagation

(b) Step 2: Route Establishment & DIO Progagation

(c) Step 3: Route Establishment & DIO Progagation

(d) Resulting DODAG

Figure 2.3: Creation of Upward Routes in RPL – The rank of a node is denoted by the number in each circle. Default upward paths between two nodes are depicted by black arrows. Grey striped lines indicate optional paths. (a) shows the multicast of DIS and the initial DIO propagation. In Figures (b) and (c) nodes join the DODAG, establish routes and propagate DIO messages. (d) shows the resulting DODAG. (figures adapted from Bauer [28])

MOP	Description	Multicast
0	no downward routes	–
1	non-storing mode	–
2	storing mode	–
3	storing mode	✓

Table 2.1: RPL Mode of Operation (MOP) Encoding [5, p. 40]

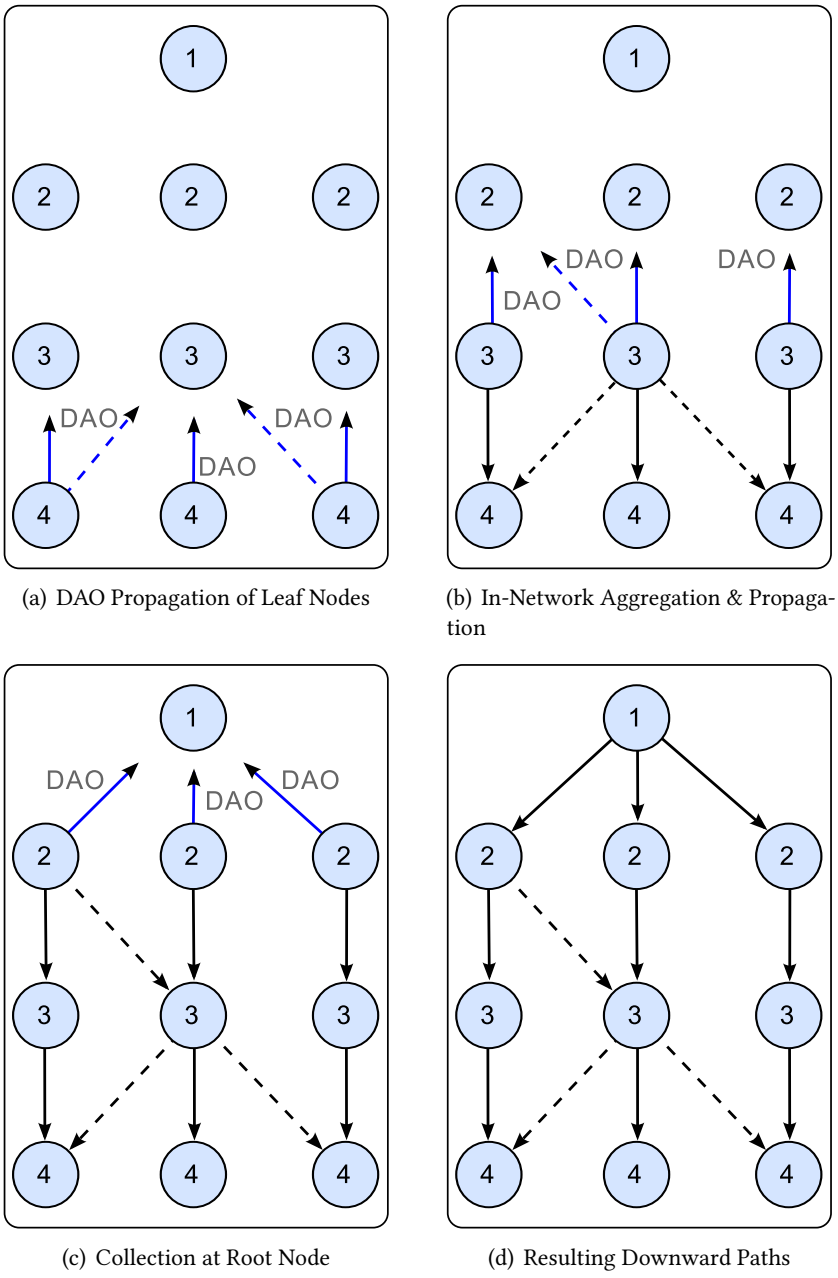
transmission count (ETX). This way the rank represents the relative distance of a node toward the root and defines all parent-child relations in the DODAG [5, p. 20]. Finally, the node advertises its rank by transmitting DIO messages to its neighbors. This allows others nodes to join or update their own routing tables.

A node schedules its DIO transmissions by a "polite gossiping" protocol called *trickle algorithm* [30]. "Polite gossiping" means that a node periodically sends information, but remains quiet if other nodes have recently sent the same information [31]. Hence, trickle distinguishes two basic incidents: reception of consistent messages (same information),[3] and reception of inconsistent messages (new information). In the first case, trickle determines the number of consistent DIO messages received within a certain time interval. A DIO is transmitted if the number of consistent messages that have been received does not exceed a defined threshold. Once the time interval expires, its size is doubled and the interval is started again. This is repeated until a maximum interval size is reached, so that DIOs are sent less frequently as long as only consistent messages are received. However, upon reception of an inconsistent message, the interval size is reset to its original size. In other words, the transmission delay of DIOs grows exponentially until an inconsistent message resets the interval and DIOs are thus sent with higher frequency.

Creation of Downward Routes Downward routes are established for communication of the root with nodes within the network and are provided by the node itself. The node therefore sends DAO messages upwards that contain its IP address or prefix and that allow for the reachability of that node. The support of downward routes is optional and supported in either *storing* or *non-storing* mode of operation as shown in Table 2.1. The mode of operation defines the basic paradigm of how the downward routing states are maintained.

In non-storing mode, the root maintains source routes toward all downward destinations [5, p. 78], so that routers do not store a downward routing state. To advertise a downward destination, a node includes its own address as well as the address of its parent in a DAO. The

[3] An RPL node considers a DIO consistent if it does not change its upward routing state [5, p. 74].



(a) DAO Propagation of Leaf Nodes

(b) In-Network Aggregation & Propagation

(c) Collection at Root Node

(d) Resulting Downward Paths

Figure 2.4: Creation of Downward Routes in RPL – DAO routes are created upon upward paths which are not shown for clarity reasons. Lines in the upward direction denote the sending of DAO messages, downward lines are established paths. Dashed lines indicate an optional route that enables multiple downward paths. (a) shows the initial DAO propagation by leaf nodes. In Figure (b) and (c) nodes receive, aggregate and forward DAO messages from their sub-DODAG. The resulting downward topology is illustrated in (d).

address of the node indicates the downward destination, and the parent's address denotes the path toward it. A node may add several parents to establish multiple downward paths for load balancing or route resilience. Assigning preference among these parents allows for the root to find the optimal downward route to each destination. The DAO is sent directly to the root and forwarded by other nodes. A node that receives such a DAO aggregates the downward routing states by including its own destination and path tuple in the message [5, p. 85]. The root node receives all DAOs and recursively creates source routes from these tuples [5, pp. 81, 85, 88]. If a DAO parent of a node changes, it provides an updated DAO to the root. The root is able to update all affected source routes without acquiring DAOs from all nodes on the path [5, p. 85]. Non-storing mode thus has the advantage of requiring less storage and computation effort by each node. However, a direct node-to-node communication requires a message to travel to the root first which then sends it on a downward route which may lead to longer point-to-point paths.

In storing mode, each router maintains the next-hop information for downward destinations and autonomously forwards downward traffic [5, p. 78]. For this purpose, a node receives DAO messages from its sub-DODAG and updates its downward routing table according to the received routing information. The node aggregates the information and sends a DAO containing its entire downward routing state to one or multiple parents. Each of these parents provides connectivity to all downward destinations that the node advertises [5, p. 86]. As in non-storing mode, assigning a preference among these parents determines the optimal route. In contrast to non-storing mode, the parent processes the information directly, so that the downward routing state propagates hop-by-hop to the root [5, pp. 84, 86]. The advantage of storing mode is the creation of shorter downward paths. A message may not have to travel all the way to the root before being forwarded downwards to higher ranks.

Figure 2.4 illustrates the creation of downward routes for storing as well as for non-storing mode. Note that in non-storing mode it is not necessary to send a DAO to multiple parents to create multiple downward routes as this information can be aggregated in a single message.

DAOs are triggered by the *DAO Trigger Sequence Number* (DTSN) that is contained in DIO messages. By sending an increased DTSN, a node requests the transmission of DAOs from its immediate children. In storing mode, a node increases its DTSN to maintain only its own routing table [5, p. 83]. In non-storing mode, the DTSN is used by the root to demand a global DAO update, in case a global repair (incremented version number) is not required [5, p. 84].

Loop Avoidance Strategies Strictly following the rank properties during the DODAG creation provides a loop-free topology [5, p. 68]. However, due to nodes dynamically joining

Field	Description
$'O' - flag$	traffic direction (flag set: downward, else upward)
$'R' - flag$	indicates rank-error, if set
$'F' - flag$	indicates forwarding-error, if set
$RPLInstanceID$	ID of RPL instance
$SenderRank$	rank of the sender

Table 2.2: RPL Control Information in Datagrams [5, p. 101]

and leaving the network or nodes moving in the topology, routing states become inconsistent. Hence, members of the DODAG produce updated control traffic to prevent outdated neighbor relations from disrupting the routing and forwarding operations.

Once a node is attached to the DODAG and has advertised its rank, a loop avoidance strategy of RPL restricts further rank changes. In general, moving closer to the root by decreasing the rank does not hold the risk of creating a loop [5, p. 72]. A rank decrease is allowed as long as the rank remains greater than the rank of all parents. A node that decreases its rank must therefore remove all parents with lower or equal rank [5, p. 71].

Moving further away from the root by increasing the rank on the other hand may result in a routing loop. A loop is caused, for instance, by a node that selects its former child as new parent after increasing its rank. To avoid these loops, a node advertises rank changes in updated DIO messages. The reception of such an update causes a resetting of the trickle timer as the new information is inconsistent with the node's current routing state. A local repair is thus triggered, resolving the inconsistent routing state.

A node that cannot maintain any valid parents detaches from the DODAG by advertising a rank of *infinity* to all neighbors [5, p. 72]. A child node that receives such a poisoning message from a parent removes that neighbor from its routing table. As an alternative to poisoning, a node may become the root of a *floating* DODAG, for instance, to sustain interconnectivity during a repair [5, p. 18]. Nodes that have no alternative routes join the floating DODAG which allows a communication with other nodes until a connection to the root is re-established.

A node may also remove a downward route if it does not further provide a forwarding path to a target. In storing mode, a node notifies the parent by sending a *No-Path DAO* which is a DAO with a lifetime of zero. A No-Path DAO indicates the loss of reachability toward a target and allows the parent node to delete the child as next hop for downward traffic. A non-storing node that removes a parent notifies the root by sending an updated DAO.

Reactive Loop Detection The loop avoidance strategies help to maintain a consistent topology. However, a certain message loss rate due to poor link quality or signal interference is expected in every LLN. A proactive approach for maintaining the topology creates a larger overhead for links that are not used at the time. RPL therefore provides a reactive detection and recovery mechanism to cope with potential loops by appending RPL control information to datagrams, such as rank and traffic direction [5, pp. 18, 101]. This *rank-based data-path validation* allows RPL to detect inconsistencies reactively rather than resorting to the frequent proactive exchange of control traffic beyond the initial creation of the topology. The control information is processed and updated at every hop on the path toward the destination.[4] A list of the required data-path validation information is given in Table 2.2.

An inconsistency is detected by using three flags ('O', 'R', 'F') and the rank of the sender. The rank of the sender is set every time before forwarding a message by the corresponding sender which is contained in each datagram. This validation process covers two basic scenarios:

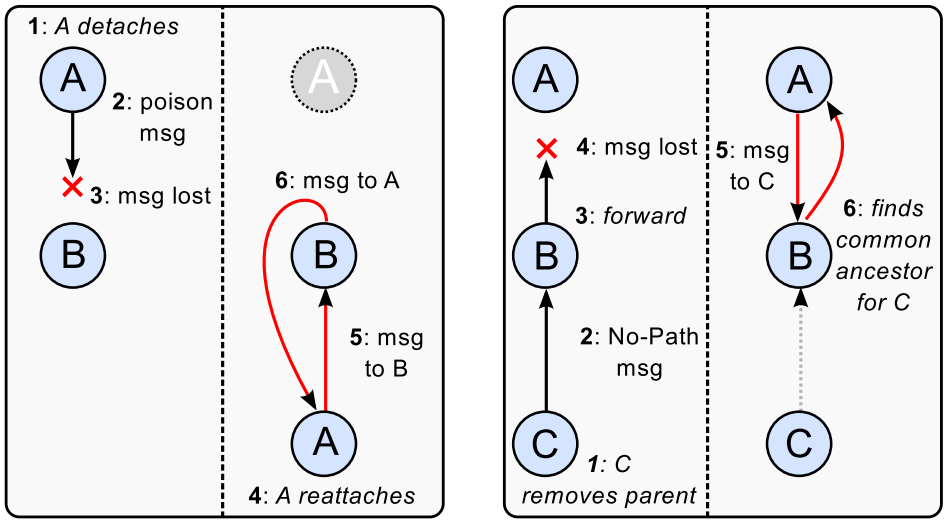
1. The sender's rank and the direction of the traffic ('O'-flag) do not match. The recipient of a downward packet must have a lower rank than its predecessor, indicated by the *SenderRank* field. In turn, the sender's rank of a packet traveling upwards must be higher.

2. A forwarding error occurs ('F'-flag set) which indicates that a node does not maintain a route to the destination, e.g. due to an obsolete or not yet updated routing state.

The loss of two sub-types of messages result in this kind of loop: a DIO message that poisons a route and a No-Path DAO that removes a downward route [5, p. 26 f.]. The following example scenarios illustrate both incidents.

- **Example scenario 1**: Figure 2.5(a) visualizes an upward loop. Node A detaches from the DODAG and poisons its routes. Its child node B does not receive the message and continues to use the detached node A as preferred parent. Next, node A rejoins the DODAG and elects B as preferred parent in turn. Each time either one forwards data traffic, the message oscillates between A and B, thus creating a loop [5, p. 26]. The inconsistency lies within the perspective of both nodes: both act as a child of the other, hence the traffic going upwards travels downwards from the other node's point of view. If A receives a message from B, with a smaller rank and the O-flag set, indicating upward traffic, A will detect the inconsistency.

[4] Details on where the information is placed are not specified by RPL. However, an exemplary external specification by Hui et al. [32] proposes to enclose the control information in the IPv6 Hop-by-Hop Options Header.



(a) Upward Loop (b) Downward Loop

Figure 2.5: Formation of Routing Loops in RPL – Figure a) shows a loop due to a lost poison message. An upward message is falsely sent downwards. In b) the loss of a No-Path DAO results in a downward loop. A downward message is falsely sent upwards.

- **Example scenario 2**: Assume three nodes A, B, and C as illustrated in Figure 2.5(b): Node A is a DAO parent for B and B is a DAO parent for C. Node C invalidates the downward path by sending a *No-Path* DAO message to B. Node B forwards the update to A. However, A does not receive the message and assumes a valid downward route to C over B. Node A sends a message to C. However, B forwards the message back to A, attempting to find a common ancestor. The message oscillates between nodes A and B [5, p. 27]. This inconsistency is addressed by the rule that a message only changes its direction once at the common ancestor [5, p. 104]. Hence, if a child receives a downward packet but does not provide the downward route to that destination, an inconsistency is detected.

Reactive Recovery As depicted in *Example scenario 1*, a message traveling in the opposite direction with respect to the data-path validation indicates an inconsistent routing state. A node that detects this inconsistency sets the rank error-flag in the packet information and forwards the message according to RPL. A single error along a path is not considered critical, therefore the recovery mechanism is only triggered upon the reception of a packet with the error-flag already set [5, p. 103]. In this case, the node discards the packet and resets its trickle timer, thus sending DIO messages in order to locally repair the routing state. The risk of such

a loop is reduced by buffering the current DODAG version number when detaching from the DODAG [5, p. 69]. A node compares the old and new version number when joining a new DODAG version to ensure that it is not re-attaching to its former sub-DODAG [5, p. 69 f.].

The recovery for downward inconsistencies as in *Example scenario 2* is distinguishable by storing and non-storing mode. A node that is unable to forward a downward message in storing mode sets the forwarding error-flag in the RPL packet information and returns the packet to its predecessor. The sender updates the downward routing table by removing the entry that caused the error. The node then unsets the error-flag and attempts to send the packet on a different route [5, p. 104]. This process is repeated until either a valid route is found or no alternative route remains. If no alternative exists, the node may request a DAO update from its children. In non-storing mode, each node picks the next-hop from the source routing header. If a message cannot be forwarded, the node discards the message and sends an ICMP error back to the root [5, p. 103] which may trigger a DAO update from the entire DODAG. The risk of this loop is reduced by acknowledging No-Path DAO messages [5, p. 27].

The above recovery mechanisms describe a local repair. If an intolerable degree of inconsistencies exist or if the local repair mechanisms cannot establish a consistent topology, the root increases the version number to initiate a global repair [5, p.17]. This global repair rebuilds the entire topology and creates a new DODAG. The rank of nodes that join the new version is not related to their rank in the prior version [5, p. 17]. A global repair does not necessarily result in a different topology setting since the same parent-child relations might form.

2.3.3 Communication in RPL

The hierarchical structure of the RPL topology allows traffic to flow in different directions along the DODAG. RPL hereby provides basic traffic flow patterns as well as support for multicast. The following gives an insight into these traffic flow patterns as well as multicast in RPL.

Traffic Flow Patterns RPL provides the infrastructure for the following traffic flow patterns:

- Multipoint-to-Point
- Point-to-Multipoint
- Point-to-Point

The topology is optimized for P2MP and MP2P traffic [5, p. 13 f.] which are the fundamental communication patterns for many applications like home and building automation [25, 26] and urban environments [23].

Multipoint-to-Point (MP2P) traffic is a mandatory feature of RPL and is naturally supported by the structure of the DODAG. Upward routes provide *convergecast* traffic on which all nodes send messages to their parents, so that the traffic is forwarded along the DODAG until it reaches the root that processes or aggregates the data. The root can also function as an LLN border router to allow forwarding to destinations on other networks.

For example, sensor nodes that monitor the temperature of a room will periodically advertise their readings. The root collects measured data, but does not necessarily analyze or evaluate the data, and sends a merged or pre-processed packet to an application server.

Point-to-Multipoint (P2MP) traffic is an optional feature in RPL and supported by the creation of downward routes. Downward routes allow the root to communicate with devices in its sub-DODAG. Depending on the mode of operation setting, nodes either maintain routing states and are able to make forward decisions autonomously, or they only forward the downward traffic according to the source routing header. Downward routes also allow the support of multicast operations. P2MP traffic is required especially in home automation [25] and urban- and industrial environments [23, 24].

Point-to-Point (P2P) communication between two arbitrary nodes of the DODAG is provided by up- and downward routes. P2P communication is often required in applications like building automation [26] and urban environments [23]. In RPL a P2P message destined to an immediate neighbor can be delivered to that node directly [5, p. 77]. Otherwise the message is forwarded upwards toward the root until it reaches a common ancestor that redirects the message downwards. In storing mode, the common ancestor can be any of the nodes on the upward path. In non-storing mode, it would always be the root, with the exception in case that the destination is located on the path from the source to the root node [5, p. 19]. P2P traffic is constrained by the properties of the tree-like topology: It is quite efficient to forward messages up- or downwards, forwarding a messages sideways among the same ranks, however, involves routing arcs [33].[5]

Multicast in RPL The traffic infrastructure can also be applied in support of multicast traffic. RPL specifies the basic support for multicast operations in terms of group registration and forwarding multicast traffic. The details on multicast in LLNs, the creation of a multicast based DODAG, and specific RPL multicast operations are not defined [5, p. 104 f.].[6] The support for multicast is only provided by storing mode and broadcasted by the mode of operation setting in every DIO [5, p. 104].

[5] Goyal et al. [34] propose optimization for non-storing mode by reactively creating P2P routes. They increase the performance of P2P by spanning a temporal DODAG sideways.

[6] Hui and Kelsey [35] define a framework for multicast forwarding in LLN to specify these features.

The registration to a multicast group is equivalent to the advertisement of a downward destination. A node registers to a group by sending a DAO message to one or multiple DAO parents. The target address contains the required multicast group address that the node subscribes to [5, p. 104]. A node either sends the DAO message to multiple parents or the preferred parent only. Hence, each node has to make a trade-off of redundancy and reliability. Using a single route for the reception of multicast traffic holds the risk of message loss. Receiving multicast traffic on multiple routes on the other hand may result in redundant messages which would have to be discarded [5, p. 105].

A node that receives multicast traffic from its parents copies the message to all children which are registered to that group as well, except the predecessor of that message. Multicast messages originating from the sub-DODAG are forwarded to the preferred parent or alternative parents, respectively. The root forwards the traffic to external sources, if required. In this manner the root functions as an automatic rendezvous point for the RPL network and an outside domain [5, p. 105].

2.3.4 Security of RPL

RPL provides basic security features to deploy secure control messages by which the communication channel is secured. The security is optional and intended to provide protection in the absence of link layer security. Other than *unsecured mode* where RPL relies on link layer security, RPL security operates in either of two defined security modes that protect the network against an intruder [5, p. 90]:

- preinstalled mode
- authenticated mode

In *preinstalled mode* all nodes are priorly equipped with a global or pair-wise shared key. Using the pre-shared key, each node is enabled to send secured control messages and thus proves its authenticity with this key during the process of joining [5, p. 90]. Preinstalled mode provides message confidentiality, integrity, and authentication [5, p. 32 ff.].

Authenticated mode also relies on a pre-shared key. Using this key, a node is only permitted to function as host and thus not allowed to forward messages on behalf of other nodes. To fully join as a router, the nodes first obtain a second key from a key authority using the pre-shared key. The key authority determines whether the node is permitted to function as a router [5, p. 91]. Authenticated mode is provided for future use, as details on the key authority and key exchange are not specified by RPL [5, p. 92].

RPL specifies the required cryptographic mechanisms to provide authenticity, integrity and confidentiality. These mechanisms are

- authenticated encryption
- digital signatures

RPL employs CCM (Counter with CBC-MAC) [13] as *authenticated encryption* mode, with AES-128 as underlying block cipher. CCM provides a CBC-MAC (Cipher Block Chaining Message Authentication Code) for message integrity and authentication as well as encryption in counter mode for confidentiality. The most important requirement of CCM to ensure security is that the CCM nonces that are required for the generation of unique key streams do not repeat for a given key (see Sec. 2.1.2).

In RPL such a CCM nonce is denoted by a 32 bit incremental counter which is sequentially increased for each secured transmission. This counter is used when encrypting messages in CCM mode and for *message replay protection*. The counter can alternatively store a timestamp in which case *delay protection* is provided as well. To protect incoming as well as outgoing packets, a node maintains two individual counters for each particular destination address [5, p. 94]: an outgoing counter for the encryption of traffic and an incoming counter for the decryption and replay/delay protection of messages [5, p. 96]. The applied outgoing counter value is transmitted in each secure control message to allow decryption for the receiving node.

RPL provides a synchronization by exchanging *Consistency Check* (CC) messages to ensure non-repeating counters and to compensate for counter state loss upon failure of a device. Figure 2.6 shows the CC message exchange after the failure of a node. The synchronization is initiated by a CC request message which contains a nonce and the expected outgoing counter of the destination. A node replies with a CC response message by which it transfers its actual counter values. The requesting node sets the incoming counter to the value received within the response. Its outgoing counter is set to an increment of the counter value that has been applied to secure the response message. Both messages carry the same nonce to prevent a replay attack.

A synchronization can also be initiated when a node receives a control message that is secured by an initial counter state that is inconsistent with respect to its own states. In this case, the receiver issues a CC response to deliver the values it currently maintains for the sending node. For example, a node that has failed and now restarts, begins its CCM counter at zero and sends secured messages. A receiving node maintains the actual counter states for that node and thus transmits its counter states in a CC response [5, p. 47]. This example

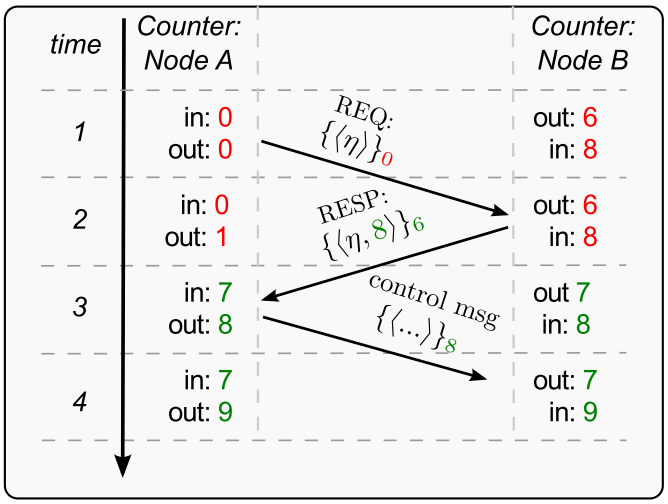


Figure 2.6: Synchronization of CCM Counter – Node A sends a CC request $\{\langle\eta\rangle\}_0$ to B. The request contains a nonce η and is protected by A's initial counter state (0). B replies with a CC response $\{\langle\eta, 8\rangle\}_6$ containing its incoming counter (8) and echoing η. The response is secured by B's outgoing counter (6). Node A resets both counter states, so that further communication uses synchronized counter states.

is analogue to Figure 2.6, however not directly initiated by a CC request, but implicitly by a control message with inconsistent counter state.

An *RSA digital signature scheme* [36] allows the creation of signatures of 2048 or 3072 bit [5, p. 99] as an alternative to authentication by CBC-MACs. The signature is created by signing the SHA-256 (Secure Hash Algorithm, 256 bit) [37] hash value of the message rather than the entire message [5] and provides authentication and integrity. When using digital signatures, encryption can be applied using AES-128 in CCM mode as well. In contrast to the CCM specification, no further CBC-MAC is created when a digital signature is applied because the signature provides sufficient authenticity [5, p. 99].

3 Security Analysis of RPL

In this chapter the security of the IPv6 Routing Protocol for Low-power and Lossy Networks is analyzed. The architectural components of an RPL network comprise of a root node and an arbitrary number of additional routers and hosts as depicted in Figure 3.1. The root is responsible for the creation of the topology and global maintenance, serves as a data collection point and supplies special functionality such as border routing into IP-based networks. Routers send and maintain routing information to provide connectivity between LLN devices. Hosts carry out the actual tasks, for example sensing environmental conditions and sending the measured data to the root.

All messages that are exchanged between devices are transmitted over-the-air. This opens the door for an outsider exploiting potential weaknesses of the network, such as unauthorized modifications of messages or gaining access to sensitive information by monitoring message exchange. Moreover, the accessibility of deployed devices, especially in public places makes it easier for an attacker to compromise a device, if not properly protected. This enables him to read and substitute routing information or obtain security credentials.

RPL specifies several security mechanisms to mitigate these threat scenarios. For a precise security evaluation, it is necessary to first describe potential threats and the attackers. Hence, this chapter begins with the definition of the threat and attacker model.

3.1 Threat and Attacker Model

The threat categories relevant for this work are related to those in the analysis by Tsao et al. [38], and threats therefore fall into either one of the categories authenticity, integrity, availability or confidentiality. Non-repudiation is not considered, because it is not relevant in the communication between devices in the context of the routing protocol [38].

Before categorizing all threats, an overview of the used terminology is given.

- **Control plane**: provides the support that is required for the establishment and maintenance of routing paths. Routers therefore exchange control plane messages which contain the relevant routing information to find the quested path for data packets.

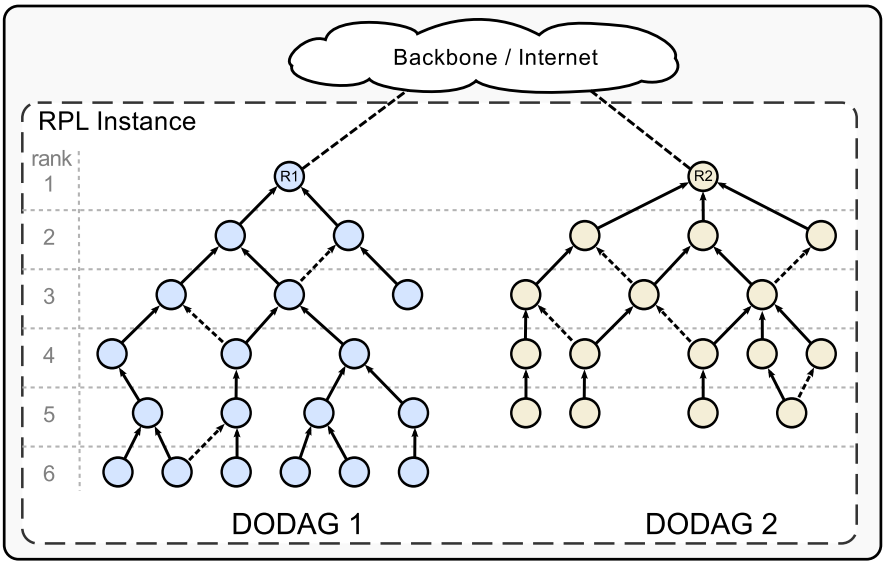


Figure 3.1: Architecture and Components of an RPL Routing Domain – DODAG 1 and DODAG 2 with root nodes $R1$ and $R2$ and many routers and hosts, respectively (empty circles). Each router or host may select multiple parents (dashed lines). Both DODAGs are connected by a common backbone link.

- **Data plane**: provides the support for data forwarding. A router that receives a packet looks up the corresponding entry in its routing table which has been created by the control plane and thus decides the next hop for a given data packet.
- **Deception** [10]
  - *by masquerade:* an attacker impersonates an authorized node to deceive another authorized node. The goal of the attacker is to obtain the privileges of the impersonated entity to execute functions or gain access to resources that require authorization. Examples for an impersonation are replay attacks or identity spoofing.
  - *by falsification:* subcategorizes into *substitution* and *insertion*. Substitution denotes any illegal modification (alter or replace) to valid data to deceive an authorized node. Insertion is the illegal introduction of false data to deceive an authorized node.
- **Disruption** [10]

  - *by corruption*: subcategorizes into *tampering* and *malicious hardware*[1]. Tampering denotes any illegal modifications to the routing logic, data or control information that alter the functions of the routing protocol. Malicious hardware is any introduced or maliciously used hardware to modify routing behavior.
  - *by obstruction*: subcategorized into *overload* and *interference*. Overload is any form of exhaustion of nodes or network. Interference denotes the blocking of communication on the control or data plane.

- **Disclosure** [10]
  - *by interception*: denotes the unauthorized access to sensitive information by spying on the message exchange (eavesdropping).
  - *by inference*: denotes the unauthorized access to sensitive information by interpreting or inferring the characteristics of the communication (traffic analysis).

3.1.1 Identification of Threats

A threat is defined as an incident that potentially causes harm. A vulnerability is a weakness of the system that allows for threats to occur [39]. Attacks aim at damaging or compromising the *assets* of a system by exploiting vulnerabilities and are thus defined as the realization of a threat. Hence, when considering an attack, a threat can be described as *the potential for an attack by exploiting a vulnerability*.

Assets in an LLN are routing information, resources, processes and nodes [38]. Routing information is exchanged over-the-air and partially stored by every node. Resources include CPU cycles, memory, energy capacities and communication bandwidth. Processes provide services such as route creation and maintenance required by the network. Nodes provide interconnectivity and form the basis of the LLN.

From these assets it can be deduced that the possibility of an adversary to gain access to an asset is either given by exploiting the wireless communication channel or by accessing a device physically or remotely. Furthermore, RPL has been designed for interoperability of smallest devices where hosts have the ability to connect to the other IP-based networks. This paradigm creates the threat of attacks launched from the Internet. However, it is assumed that border routers are able to use complex and praxis-proven cryptographic protocols, such as digital signatures combined with an infrastructure for digital certificates to communicate with hosts or routers in other routing domains. This work therefore focuses on threats and attacks

[1] Re-defined from RFC-4949, which defines malicious logic as hardware, firmware and software.

Classification	Threat
Authenticity / Integrity	Deception of an authorized node by *masquerade*
	Deception by *substitutions* in the control / data plane
	Deception by *insertions* in the control / data plane
Availability	Disruption of control / data plane by *corruption*
	Disruption of control / data plane by *interference*
	Disruption of control / data plane by *overload*
Confidentiality	Disclosure of sensitive information by *interception*
	Disclosure of sensitive information by *inference*

Table 3.1: Summary of Threats

launched from inside an RPL routing domain. Hereby it is assumed that border routers, due to their essential role in RPL, are tamper resistant and thus not compromised by an attacker.

The relevant threat categories are summarized in Table 3.1 and described in the following.

Threats to integrity and authenticity The communication channel can be exploited by the substitution of message content or the insertion of forged messages. This can, for example, be achieved by intercepting messages and making illegal modifications before transmitting them to the actual destination or by accessing a node directly. An attacker that has access to a node may substitute its routing state or reprogram the device with a malicious code, so that the node sends false routing information. He thus has the ability to influence or control route creation and maintenance. He further impersonates other authorized devices to gain their privileges. This enables him to access and misuse protected resources or routing information, or to corrupt processes.

Threats to availability A routing protocol requires services for topology/route creation, maintenance and forwarding. Routers or nodes exchange routing information and provide connectivity, so that all messages reach their destination. Nodes require a minimum of resources to function correctly and to provide those services. An exhaustion of these resources leads to a decreased availability of routing and forwarding. Threats on the availability are thus the disruption of the control or data plane. An attacker achieves this by obstruction or corruption of message exchanges.

Threats to confidentiality The routing information is subject to unauthorized disclosure. An adversary gains unauthorized access to the routing information by intercepting the communication channel or by accessing a node directly. He may further analyze the communication

patterns and infer helpful information. He thus gains knowledge of routing states which help him to detect a lucrative access point for further attacks.

3.1.2 Attacker Model

The attacker is provided with certain capabilities and resources to exploit potential vulnerabilities of the RPL network which thus denotes the actual risk of a threat. RPL implements cryptographic countermeasures which require security keys. For this reason, this work distinguishes two classes of attackers, as defined by Karlof and Wagner [40]: an insider that has obtained a set of security credentials, and an outsider attacker with no security keys.

Outsider attacker The outsider attacker tries to intrude the network from outside the security perimeter and thus denotes an unauthorized entity. This means that he has no access to any cryptographic keys and is restricted by the security measures of RPL. He thus intends to exploit the wireless communication channel.

The attacker is assumed to be an experienced hacker. Hence, he is assumed to possess standard tools including a general purpose computing device such as a modern laptop which gives him much higher resources than remaining nodes in the network. Furthermore, air-sniffing tools like Wireshark to capture wireless communication and tools to substitute the content of messages and insert new messages into the network are assumed to be be available to this class of attackers.

The outsider attacker is not limited to a single device. Multiple laptops enable the attacker to establish out-of-band communication between attacking devices. Furthermore, he may install malicious hardware, like directed or more sensitive antennas, to influence the signal propagation of the in-band communication and to increase his transmission and reception range.

Insider attacker All characteristics of the outsider attacker apply to the insider attacker as well. Additionally, the insider attacker is assumed to have obtained a set of security keys. This might be done by an insider (traitor) actually selling or providing cryptographic keys, or by compromising one or more devices of the network. For instance, a memory readout enables him to extract routing information and security keys which he copies to an additional laptop-class device. Hence, he is able to act as an authorized entity with the same privileges of the compromised node. It is further assumed that these privileges allow the insider attacker to function as a router in the network.

3.2 Security Objectives

Security solutions for RPL must mitigate the potential threats. As summarized in Table 3.1, each threat in this model belongs to one of the following three classes: authenticity and integrity, availability or confidentiality.

In the following, the available (cryptographic) tools and protocols that might be used to protect threats against authenticity and integrity, availability and confidentiality are briefly introduced. Note that the physical theft of a node cannot be directly countered on the network layer [38]. Therefore, the countermeasures, for example using tamper resistance hardware, are out of the scope of this thesis. Furthermore, the integrity protection is typically considered in combination with authenticity in the form of (message) authentication. Hence, the tools available for protecting the authenticity and integrity are described in the same section.

Authenticity and integrity Message authentication is required in LLNs to ensure that the source is who it claims to be and thus to create a bond of trust between communicating partners. The RPL security model must therefore prevent an adversary from masquerading as an authorized node and from inserting false messages into the communication channel. Integrity checks are required to detect the substitution of message content. This prevents an attacker from influencing the routing behavior by illegally modifying routing information during transfer.

Availability Mechanisms that provide availability consider the usability or reliability of services and resources when they are required. Thus, the interference or disruption of the communication channel and the exhaustion of the resources of nodes are threats to availability. RPL must therefore provide a sufficient amount of message redundancy to ensure the availability even in the event of node failure and partial disruption of the network.

Implicit threats to availability are evident whenever the violation of integrity or authenticity leads into the unavailability of a service. This is the case, for instance, if traffic is redirected by corruption of routing information. Hence, the implementation of authentication schemes and integrity checks increase the availability as well.

Confidentiality When sensitive data is transmitted over-the-air, it is important to prevent that information from unauthorized disclosure by an attacker. In general, it is hard to detect whether an unauthorized node has read the message during transit. Confidentiality provides protection so that only authorized entities gain knowledge of the message content, and it is typically provided by encryption. Further, an attacker may analyze traffic flows and read

unencrypted routing information as a preparation for another attack. To make it more difficult for an attacker, for instance, to spot lucrative access points, the routing information may be encrypted. However, the mere knowledge of routing information does not directly influence the routing behavior [38]. Hence, these threats are not the primary concern of this survey.

3.3 Security Evaluation of RPL

This section evaluates the RPL compliance to these security objectives in consideration of an outsider as well as an insider attacker. The security mechanisms used in RPL are assumed to operate with a preinstalled key. The keys required for encryption and authentication are shared among root node, routers and hosts by using a key management mechanism. Due to its central role for security models based on cryptographic keys, the key management of RPL is discussed as prerequisite to other security mechanisms.

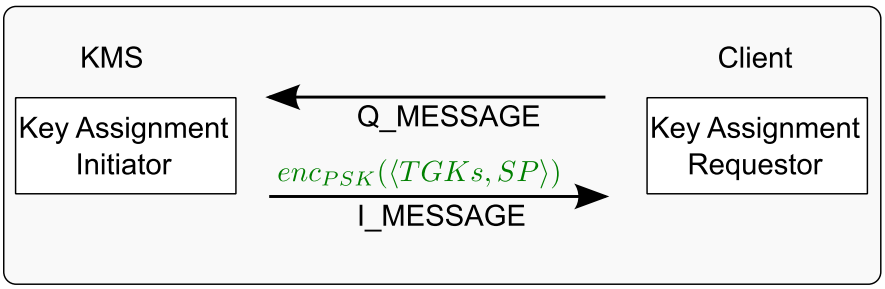
3.3.1 Key Management

Cryptographic protocols require a security key, so that it is important to equip each node with all necessary keys. There are two basic paradigms for key management: manual and automated key management [16]. RPL specifies a manual key management in which keys are preinstalled by some out-of-band mechanism. The specification of an automated key management is left to external documents.

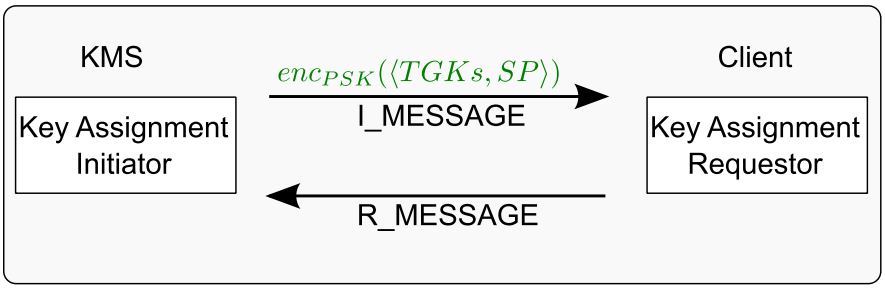
The manual out-of-the-box key management of RPL provides symmetric group keys or pair-wise keys. A group key, however, is a security breach already in case a single key is stolen, as the entire group shares the same key. While pair-wise keys mitigate this impact, they may result in scalability issues, if the number of nodes becomes large, as each node stores $n - 1$ keys to securely communicate with all other nodes.

This minimal specification of a manual key management complies with the overall goal of RPL to provide only basic security features. However, especially large scale networks require an automated key management to prevent the overuse of security keys and to revoke keys in the case of a compromised node [16].

As a possible solution to these problems, Alexander and Tsao [41] propose AMIKEY which is currently at the state of an Internet draft. Their approach denotes a key management for LLNs based on on *Multimedia Internet KEYing* (MIKEY) [42]. Next, AMIKEY (Adapted MIKEY) is briefly introduced and evaluated for use with RPL.



(a) Key Assignment Requested by Client



(b) Key Assignment Initiated by KMS

Figure 3.2: AMIKEY Key Assignment – Simplified view of the key assignment process in AMIKEY. The TEK generation key(s) (TGK) and the security parameter (SP) are encrypted using the pre-shared key (PSK). (a) shows the initiation by the client and (b) the initiation by the key management server (KSM). (Based on [41])

Automated key management by AMIKEY AMIKEY intends to improve the security of RPL by allowing to dynamically exchange master keys or TEK Generation Keys (TGKs). Session keys or Traffic-Encryption Keys (TEK) are derived from the TGK. TEKs allow for frequent key refreshment to prevent the overuse of security keys and allow for re-keying if a session key is compromised. The TGK is exchanged using either a pre-shared key, asymmetric cryptography or optionally Diffie-Hellman key exchange.

AMIKEY is designed to support the authenticated security mode of RPL. In authenticated mode a joining node uses a pre-shared key to obtain a second key from a key authority. Figure 3.2(a) shows a node that requests a key assignment using a pre-shared key (PSK). A client sends a request to a key management server (KMS). The KMS initiates the key assignment by sending an initiator message, protected by the pre-shared key, that contains one or more TGKs and a set of security parameters (SP). The update of a TGK works in the same way, so that a node sends a request message to the KMS and receives a new TGK from the KMS.

Technique	Defends against	Outs.	Ins.
CCM / AES, RSA	unauthorized disclosure of security keys	✓	–
AES in counter mode	unauthorized disclosure of routing information by *interception* of control messages	✓	×
CBC-MAC / RSA signature	deception by *substitution* of information in control messages	✓	×
	deception by *insertion* of forged control messages	✓	×
	deception by *masquerading* as authorized node	✓	×
Counter	deception by *insertion* of outdated control messages (replay)	○	×
Timestamp	disruption by overly delaying control messages	✓	×

(✓) detected / prevented (×) not defended (○) mitigated

Table 3.2: Cryptographic Defense Techniques in RPL – Protection against an insider (Ins.) and outsider (Outs.) attacker.

Re-keying can also be initiated by the KMS as visualized in Figure 3.2(b). The KMS sends the key update to the node which responds to complete the key update and to provide mutual authentication. The communication is secured by a pre-shared key. The key exchange can alternatively be secured using public/private keys or a key exchanged by the Diffie-Hellman protocol. This approach relaxes the scalability issue of pair-wise keys and provides equivalent protection in case of a compromised session key. However, once a PSK is compromised, all communication that has been secured with the PSK is potentially breached. An adversary may decrypt priorly exchanged and recorded TGKs by using a stolen PSK.

A practical issue of AMIKEY is that it has not yet been standardized. Current RPL implementations therefore have to resort to other key management solutions or simply the preinstalled mode of RPL with a manually installed key. The security of this preinstalled mode is analyzed next.

3.3.2 Cryptographic Defenses

RPL defines various cryptographic countermeasures such as authentication schemes, optional encryption as well as replay and delay protection. These cryptographic defenses of RPL cover all types of control messages. However, datagrams which also contain routing information are not secured by RPL. The security protocols described in this section therefore only consider control messages and not datagrams. The security constructs and procedures provided by RPL are summarized in Table 3.2 and discussed in the following.

The cryptographic protocols that protect the *authenticity*, *integrity* and *confidentiality* of control messages of RPL include

- AES-128 in counter mode (CCM[2])
- CBC-MACs (CCM)
- RSA signatures

In RPL, encryption is provided by CCM with *AES-128 in counter mode*. This encryption scheme requires the creation of a unique key stream by using a secret key and a unique CCM nonce for each key stream. In RPL, unique nonces are provided by an incremental counter which is increased for each secured control message and is included in the message. The encryption covers the entire base of the control message, excluding the IPv6 and ICMPv6 header as well as security parameters of RPL which is required for decryption.

CCM provides the authentication of messages by *CBC-MACs* to verify the origin and the integrity of control messages. For this purpose, a MAC is created over the entire unsecured control message. A node that receives such an authenticated message verifies the source and the integrity of the message. If successfully verified, the node can be certain that the content of the message has not been substituted and has been created by the corresponding key owner. The node therefore trusts the data contained in the message and its source. If the verification fails, the message is dropped. If encryption is applied, additional authenticated data provided by CCM is used to authenticate the unencrypted IPv6 and ICMPv6 header and security parameters to allow the verification of these information before decrypting a message [5, p. 97].

An outsider attacker that intends to forge a MAC or to decipher an encrypted message has to break the security of CCM or AES. The security of CCM has been proven by Jonsson [43]. He demonstrates that it is infeasible for an adversary to derive information from a cipher or to forge a valid cipher as well as authenticated data without knowledge of the secret key. Likewise, when properly implemented and applied, no practical attack is known that breaks AES-128 faster than an exhaustive search [44, 45]. An outsider attacker has to find the key by brute-force and search the key space of 2^{128} which is effectively infeasible. A MAC therefore prevents an outsider attacker from falsifying routing information or from masquerading as an authorized node and inserting messages into the network.

In RPL digital *RSA signatures* can be applied as an alternative to CBC-MACs. The authentication and verification process is similar to MACs, only that public/private key pairs are

[2] CCM: Counter with CBC-MAC / CBC-MAC: Cipher block chaining message authentication code

used instead of a single secret key. The sender signs a message with its private key which can be verified by any node that has access to the corresponding public key. Upon success the verifying node trusts that the data has been created by the owner of the private key.

The security of RSA signatures has been proven by Bellare and Rogaway [46]. They show that forging a signature is as hard as solving the RSA problem [47] for which no efficient algorithm is known when using keys of at least 1024 bits [48]. Hence, an outsider attacker can neither forge a RSA signature nor extract the private key from a secured communication and is thus prevented from claiming a false identity or from falsifying routing information.

The signing with a private key allows a distinct identification of the owner of the private key, if combined with a corresponding digital certificate. However, the signature scheme comes with 3 major drawbacks. First it requires additional infrastructure to issue and propagate these certificates and to allow a distinct identification of the key owner. However, as of writing this thesis, such a public-key infrastructure has not been standardized for RPL. Secondly, public and private keys are much larger than symmetric keys: AES keys are provided with 128 bits, whereas RSA signatures in RPL have a key size of 2048 and 3072 bits. Such large keys increase the memory requirements and the transmission costs in a multi-hop LLN [49]. And lastly, the computational requirements and thus energy consumption are much higher compared to symmetric approaches [50, 51]. Using signatures may be useful for an initial authentication of the root, but is infeasible for regular authentications or verifications of in-network routers when considering very constrained devices. Additional signature schemes such as elliptic curve cryptography [52, 53] as well as concepts for certificates can be provided by future external specifications like AMIKEY.

Both approaches, CBC-MAC and RSA signatures, prevent an outsider attacker from the falsification of messages or from masquerading as an authorized node. The insider attacker on the other hand behaves as an authorized node and therefore sends and receives control messages. He may insert false routing information or substitute data within all messages using the acquired security keys. Signatures or pair-wise applied secret keys may hereby reduce the scope of the security keys that can be obtained by an insider attacker [5].

Replay and delay protection In addition to authentication and encryption schemes, RPL also provides *replay and delay* protection by the following constructs:

- sequential counters (replay protection)
- timestamps (delay protection)

In a message replay attack, an adversary retransmits a prior message to an honest node. For the detection of replay attacks, RPL takes advantage of the property of CCM of non-repeating nonces. If a nonce ever repeats, a replay attack is detected. A node therefore maintains two *sequential counters* for each target: an outgoing counter to secure control messages toward another node and a counter for incoming secured control messages of that node. A node that receives a message checks whether the contained counter is an increment of the current incoming counter state for the source. If a counter value repeats, a potential replay attack is detected and the message is dropped [5, p. 96].

This detection techniques can only reliably detect replay attacks, if the counter states of nodes are synchronized. This is achieved by the exchange of consistency check (CC) messages. This process is initiated either upon reception of a CC request or if a secured control message is received that has been secured with an counter state of zero, although the receiving node maintains different states. This may happen when a node reboots and loses its counter states.

Since in both cases the counters are potentially unsynchronized, a replay protection during synchronization is not provided by these counters. For this reason, a CC request message contains an additional nonce that is echoed in the corresponding CC response. When the response is received, the requesting node checks the nonces and only updates its counters upon a positive match. In contrast, a secured control message with the unexpected counter value of zero is not protected against replays. The reason for this is that echoing the zero-counter value in the response is trivially predicted by an attacker and thus does not provide replay protection.

This sequential counter, however, cannot protect against an attacker that prolongs the propagation of control messages. Therefore RPL provides a *timestamp* that detects such delay attacks. In a delay attack, an outsider attacker postpones a control message for a certain period of time. To detect a delay the CCM counter is applied and represented by a timestamp. If the timestamp in a received message indicates that the message delivery took longer than a tolerated delay, the message is dropped. A substitution of the timestamp is detected by the message integrity check. A timestamp also provides replay protection, but comes with the drawback of requiring a loose time synchronization.

3.3.3 Non-Cryptographic Defenses

RPL has certain characteristics within its design that can be applied to increase the availability of services of the network and may also be used to protect against the unauthorized access to routing information. These characteristics are:

Technique	Defends against / mitigates
Control message redundancy	Disruption by *interference* (blocking / dropping of messages)
Multiple paths	Disruption by *interference* (partial blackout of network)
Traffic randomization	Disclosure by *inference* and *interception* of routing information
Bi-directional link validation	Disruption by *interference* (unidirectional links)

Table 3.3: Non-Cryptographic Defense Techniques in RPL – This table shows the inherent design properties that may mitigate attacks on availability and confidentiality. All properties apply to the outsider as well as the insider attacker.

- control message redundancy
- multiple paths
- traffic randomization
- bi-directional link validation

Redundancy is a common method for increasing the availability of a service. Since message loss is one of the characteristics of LLNs, RPL sends *control messages redundantly*. DIO messages, for instance, are scheduled redundantly by the trickle algorithm. If messages are lost or blocked by an attacker, another transmission may be received.

Multiple paths are another form of redundancy by which nodes select multiple parents for resilience against node failure or for load balancing. Multiple paths provide some resilience against denial of service attacks as well. If an attacker disrupts parts of the network, the children of affected nodes may choose an alternative path to sustain the connectivity to the root.

Traffic randomization [38] is a technique by which the next-hop for each packet is chosen at random from one of multiple choices. An attacker that eavesdrops or analyzes traffic flows may thereby only receive a small part of the traffic, so that the impact of these attacks is mitigated. However, it may not be applicable in scenarios where randomized traffic is not desired, for example if routing is optimized for preserving energy resources of certain devices.

Another technique is the *validation of bi-directional links* which is required by RPL. The reason for this is that upward routes are created by sending DIO messages downwards, and downward routes are created by sending DAOs upwards. The validation of bi-directionality

Type	Attack	Outs.	Ins.
Wiretapping	Eavesdropping	✓	×
	Traffic analysis	○	○
Identity misuse	IP spoofing	✓	×
	Sybil attack	✓	×
Crypto-suite attacks	CC-replay attack	×	×
	Cryptographic processing attack	✓	×
Illegal repair	Version number attack	✓	×
	Poisoning attack	✓	×
	Datagram forgery	×	×
Distance fraud	Rank spoofing	✓	×
	HELLO flood attack	✓	○
	Wormhole attack	×	×
Traffic filtering	Selective collision	○	○
	Selective forwarding	✓	○

(✓) detected / prevented (×) not defended (○) mitigated

Table 3.4: Summary of Attacks against RPL – Attacks by an insider (Ins.) and outsider (Outs.) attacker.

can be used to mitigate attacks that create false neighbor relations by creating uni-directional links [40].

These techniques by themselves do not prevent an attack, but may mitigate the impact of attacks launched by outsider as well as insider attackers. Table 3.3 summarizes these characteristics which are further evaluated in the next section.

3.4 Attacks and Countermeasures

The defense techniques of RPL greatly reduce the impact of an outsider attacker, but cannot defend against an insider that has access to the required security keys. Furthermore, datagrams are not covered by the RPL security but play an important role for the maintenance operations. Therefore, datagrams denote a vulnerability which is exploitable by an outsider attacker.

This section presents the potential attacks against RPL in consideration of an insider and outsider attacker and discusses potential countermeasures. The attacks are summarized in Table 3.4 and described in the following.

Wiretapping The communication in RPL takes place over a wireless medium. An attacker that is able to receive the communication of authorized devices may spy on their communica-

tion to obtain sensitive information. This work distinguishes between two attacks that allow this unauthorized disclosure by wiretapping[3]:

- eavesdropping
- traffic analysis

An adversary that *eavesdrops* on the communication channel receives the messages of authorized nodes to gain access to sensitive information. The insider attacker decrypts control messages that are encrypted using the acquired security keys and thus undermines the confidentiality of routing information. Although the outsider attacker is prevented from eavesdropping on the content in control messages, the routing information in datagrams may not be encrypted. In this case the outsider has access to the transmitted information.

Further, the outsider attacker has the ability to analyze traffic flows. In a *traffic analysis*, he monitors traffic flow patterns to obtain and evaluate routing information. The encryption does not conceal header information, so that he examines communication behavior of devices. He implies topology settings and message timings or searches for an attractive attacking point such as a device to compromise. Furthermore, routing may account for energy resources, so that traffic flows reveal information of nodes that have low energy supplies. These devices may be targeted in a denial of service attack. Multiple paths and the randomization of traffic can be applied to mitigate a traffic analysis, if applicable by the specific application. Characteristic traffic flows are thereby obfuscated and reveal less information.

Identity misuse An adversary can also influence the routing behavior by the falsification of messages. When inserting messages with a false identity he obtains unauthorized access to resources or processes and deceives honest nodes or disturbs the network. The following attacks allow an adversary to masquerade as an authorized node:

- IP spoofing
- Sybil attack

In an *IP spoofing* attack the insider attacker uses the IP address of an authorized entity. He spoofs his identity to any other node to gain its privileges. For example, the attacker impersonates the root node and is thus able to propagate the root's rank (see *rank spoofing*).

[3] Although the term *wiretapping* infers a mechanical connection, this work uses the term for wireless links as well according to Shirey [10].

Such an IP spoofing attack could be prevented by using Cryptographically Generated Addresses (CGAs) [54]. A CGA is generated with a public/private key pair, so that only the private key owner can generate messages from this address. CGAs rely on the use of RSA, so that they may be infeasible for use in many LLNs. However, Sarikaya et al. [55] propose a lightweight approach especially for LLNs by using elliptic curve signatures. The use of these CGAs is not yet standardized for RPL, so that IP spoofing is applicable for an insider attacker. As IP spoofing requires the creation of an authenticated message, an outsider attacker is detected by the authentication schemes.

Another misuse of identities is denoted by a *Sybil attack* [56] in which an insider attacker creates multiple fake identities on a single hardware interface. The goal of this attack is to insert the false identities into the routing tables of other nodes. The insider attacker thereby provides seemingly valid routes through multiple artificial parents. A node unknowingly selects these artificial parents and thus a multitude of traffic is directed toward the attacker. Techniques like multiple paths do not mitigate this attack, as all forwarding decisions lead toward the attacker.

The outsider attacker is prevented from both attacks. He may, however, masquerade as an authorized entity by a message replay attack.

Crypto-suite attacks *Crypto-suite attacks* are targeted at the security features of RPL and may be used to implement identity misuse or to exhaust the resources of nodes. Such attacks include:

- CC-replay
- cryptographic processing

Although RPL provides replay protection, an outsider attacker can avoid detection under constrained conditions. In a *CC-replay attack* the outsider attacker exploits a weakness of the CC synchronization process which allows him to replay a control message that is secured by an initial (zero) counter value. Assume two honest nodes which have communicated before and thus maintain non-zero CCM counter states. Providing that the attacker has received a secured control message with a zero-counter state from this communication, he replays this message to the original destination. The receiving node assumes that the source of the message has lost its counter states and thus replies with a CC response. If executed repeatedly the processing of the replayed message and the resulting transmission of a response overloads the victim and strains its energy resources.

Possible countermeasures may involve a threshold to limit the number of initiated synchronizations due to the reception of such a message. This of course bears the risk of an attacker exhausting this threshold to prevent a required counter synchronization upon a device failure. A more complex countermeasure could analyze the progress of security counters during the communication. Hereby, the alternating reception of message that are secured with zero counter states and the correct counter indicates suspicious behavior.

Attacks that are also directed at the security model of RPL are *cryptographic processing attacks* [38]. Hereby an outsider attacker intends to exploit the order in which secured control messages are processed in RPL to force a node to decrypt an invalid message before it is dropped. An outsider attacker substitutes the replay protection counter and replays the forged message. For this purpose, RPL uses additional authenticated data that is provided by CCM. This additional authentication allows the verification of unencrypted information and thus the replay protection counter, so that this attack is detected. The insider attacker on the other hand simply sends valid secured messages and forces an honest node to perform decryptions and verifications to drain energy resources and to disrupt the node.

Illegal repair The initiation of illegal repairs denotes a type of attack that enables an adversary to disrupt larger parts of the network. Hereby the insider and outsider attacker start a global or local repair, respectively. There are several methods that initiate such an illegal repair:

- version number attack
- poisoning attack
- datagram forgery

A version number increase initiates a global repair of the topology and is reserved for the root node. Typically this repair is started when local repairs do not sufficiently resolve the inconsistencies within the topology. In a *version number attack* [6], a global repair is triggered by an insider attacker. He propagates an illegal increased version number to all neighboring nodes. These nodes reset their trickle timer and begin to frequently send DIO messages. The version update is propagated to all nodes in the network, so that the entire topology is re-built by the attacker. When repeatedly executed, this attack disrupts the forwarding operation of the data plane by overloading the communication channel with control plane messages and exhausts the energy resources of all nodes. Countermeasures to this attack are discussed in Chapters 4 and 5.

A variant of the version number attack with local impact is the *poisoning attack*. As presented by Le et al. [57] an insider attacker poisons its upward routes and cause neighboring nodes to update their parents set. As a result, the neighbors reset their trickle timer and send DIOs more frequently for a local repair. This attack is done repeatedly to overload these nodes. Le et al. propose an intrusion detection system that monitors the network and detects such an attack.

The propagation of a falsified version number or increased rank requires an attacker to send authenticated messages, so that only an insider attacker has the ability to launch these attacks. An outsider attacker is limited to the falsification of datagrams which contain routing information as well and which are not protected by the security of RPL.

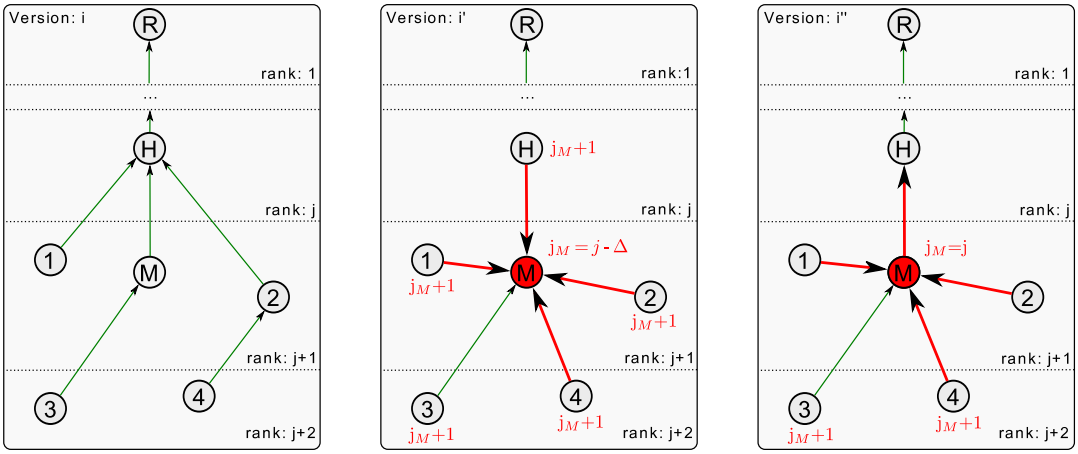
Hui and Vasseur [32] describe an attack in which a *datagram forgery* allows the outsider attacker to create an inconsistency that results in a local repair. Hereby the outsider attacker modifies the routing information of datagrams, so that the traffic direction is inconsistent with respect to the rank within the datagram. He returns the datagram to the predecessor, so that it resets its trickle timer as it wrongfully detects a routing loop. Alternatively, the outsider attacker clears downward routing entries in storing mode, when a datagram is returned to the predecessor with the forwarding error bit set. As this indicates a false downward routing state within the predecessor, it wrongfully deletes the downward route.

To mitigate the impact of attacks on the routing information in datagrams, Hui and Vasseur propose a threshold that restricts the number of times a node maintains its routing tables due to inconsistencies in datagrams. A drawback of this approach is that an attacker may exhaust this threshold to locally disable the repair service.

Distance fraud In a *distance fraud* attack the adversary propagates a false topological or physical distance to other nodes. He attracts traffic by seemingly providing a shorter distance to the root than other nodes in the vicinity or creates non-existing neighbor relations. These attacks include:

- rank spoofing
- HELLO flood
- wormhole

In RPL the topological distance to the root is represented by the rank. In a *rank spoofing attack* [6], an insider attacker propagates an improperly decreased rank to improve its position in the topology. Since the low rank suggests a more profitable path toward the root, neighboring nodes wrongfully select the attacker as parent.



(a) Regular RPL routing hierarchy

(b) Topology after a rank spoofing attack

(c) Topology after a replay attack

Figure 3.3: Rank Spoofing and Replay Attacks – RPL topologies (a) of regular arrangement and (b) after 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. (c) visualizes a replay of the parent rank, only attracting nodes 1, 2, and 4 with intact upstream to H.

In RPL the rank is calculated by incrementing the parent's rank. The minimum increase between the parent's and the own rank is denoted by the parameter *MinHopRankIncrease* (MRI).[4] Hence, any propagated rank lower than the sum of the parent's rank and MRI is considered illegal and therefore rank spoofing. The lowest rank spoofing is therefore a rank replay in which an attacker claims the rank of his parent. The spoofed rank, j_{spf}, is thus defined by

$$j_{spf} = j_P - \Delta \qquad \text{with } \Delta \geq 0, \tag{3.1}$$

where j_P denotes the rank of the parent and the parameter Δ represents the degree of the rank spoofing in rank level or hops.

Figure 3.3(a) shows an exemplary RPL topology created exclusively with honest ranks. Figure 3.3(b) illustrates the forged topology after a rank spoofing attack. Node M propagates a lower rank than all its immediate neighbors. This causes all nodes, even former parent H, to select M as parent and creates a sinkhole within the topology. This sinkhole prevents the attacker from sending traffic to the root, because all surrounding nodes selected him as

[4]For simplicity, this work assumes that each node increases its rank by 1 hop per rank level.

next-hop for upward traffic. To remain connected to the root, the attacker may resort to a rank replay attack.

A rank replay attack is a special case of rank spoofing in which the attacker illegally claims the rank of his parent. This is equivalent to a rank spoofing with $\Delta = 0$ by which the adversary moves one level up in the hierarchy. Figure 3.3(c) shows the effect of a rank replay attack. Three additional nodes choose the attacker as parent. The attacker may use node H for upward traffic, as it does not select him as parent due to his equal rank. However, once the attacker sends upward traffic through a parent node of equal rank, the parent will notice a violation of the monotonic rank order. The message is not dropped until the second inconsistency on the path, but decreases the probability of successful message delivery. The insider attacker circumvents this consistency check by using a correct rank when communicating with his parent and announces the replayed rank to its children. Since RPL does not specify any checks that combine the rank of child nodes with the rank in a datagram, the inconsistency detection fails to detect such a rank replay attack. Topology authentication schemes to defend against rank spoofing and rank replay are thoroughly described in Chapters 4 and 5.

Distance fraud is also possible by the introduction of malicious hardware. The insider attacker may propagate his real or a spoofed rank with an increased transmission strength. In this *HELLO flood attack* [40] the insider attacker sends DIO messages to distant nodes, which are unable to transmit messages over the same distance. Each transmission of these nodes toward the attacker is lost in a *black hole* [40]. A mitigation technique is the validation of bi-directional links by which a node only accepts messages if it also has a connection to the source. However, as shown by Wallgren et al. [58], this validation only mitigates the attack to some degree. Furthermore, bi-directional link validation techniques fail once an attacker uses a very sensitive receiver allowing him to receive traffic of his victims and to confirm bi-directionality [40]. However, since a HELLO flood attack requires the attacker to provide a valid DIO message, the outsider attacker is prevented from this attack.

Another possibility for a distance fraud is a *wormhole attack*. Hereby two attackers communicate through an out-of-band channel. Through this wormhole the attackers may connect honest nodes that are actually far away, and inject a lower rank anywhere in the topology. All traffic is tunneled through the wormhole and passes the attackers. The out-of-band channel hereby creates a virtual upward path that allows the attacker at higher rank level to create a sinkhole and to forward data traffic to his accomplice. Hereby he sustain the connectivity to the root. In principle the wormhole attack can be used in combination with various attacks [58] and is applicable to an outsider as well as insider attacker.

Traffic filtering In traffic filtering attacks an adversary selectively interferes with the communication of other nodes to block certain messages or even entire services. Traffic filtering may be implemented by the following attacks:

- selective collision
- selective forwarding

In a *selective collision attack* [59] the outsider attacker creates targeted physical interference to prevent honest nodes from sending or receiving messages. He predicts the point in time a node transmits a message and sends messages simultaneously to occupy the communication channel or to create collisions. The required timing can be inferred by prior traffic analysis. By blocking DIO messages the attacker impedes the propagation of a version update and – in contrast to the version number attack – prevents affected nodes from performing a (global) repair. As this attack requires precise timing, the attacker might not be able to block all messages. In general, attacks that involve physical jamming are impossible to be prevented on the routing layer. They are best anticipated on the physical layer [38], for instance, by spread spectrum communication [60] in which different channels are used which may not be affected by the attack.

The insider attacker has the ability to *selectively forward* messages [40]. As he functions as authorized entity, he is included in the topology and selectively forwards messages that he receives. He may deny forwarding of data plane messages or refuse to propagate control messages, such as DIOs or DAOs. For instance, by not forwarding DAOs to the root, the attacker prevents his sub-DODAG from creating downward routes.

Selective collision as well as forwarding attacks can be mitigated by traffic randomization, since only part of all traffic may be forwarded or directed toward the attacker [58].

3.5 Discussion

RPL defines the basic security features to provide protection against potential attacks. Many details have been left to external specifications such as the definition of an automated key management. The manual key management provided by RPL meets the minimal requirements for implementing security in an RPL network. Further specifications such as AMIKEY may supplement an automated key management for scenarios that cannot manually provide security keys. Until a standard is released, implementations have to resort either to non-standardized approaches or the preinstalled security mode of RPL.

The preinstalled security mode provides authenticity and integrity as well as confidentiality by a pre-shared key. Confidentiality is provided by encryption. Although the encryption in RPL makes it more difficult to implement some attacks, it cannot prevent attacks on the routing operations that were presented in this work. Authenticity and integrity therefore denote the most important objective. Two approaches are provided for the authenticity and integrity of messages: message authentication codes and digital signatures. While message authentication codes provide authenticity and integrity that is feasible even for restricted devices, the digital signature scheme requires large keys which result in high computational and memory overhead as well as large message sizes. Until additional signature schemes are specified, RPL implementations may apply message authentication codes or resort to careful use of digital signatures, such as an initial authentication of the root node.

These cryptographic protocols prevent most attacks against an outsider attacker that exploits the control message exchange. However, an outsider attacker may launch a message replay attack under constrained circumstances and thus drains the energy resources of the victim node. Other attacks are targeted at the routing information in datagrams by which a local repair is started or the integrity of routing tables is violated.

Since cryptographic protocols are ineffective once an attacker has access to the security keys, an insider attacker may launch various attacks. Although design properties of RPL can mitigate some of these attacks, the insider attacker remains a serious threat to an RPL network. Especially rank spoofing and version number attacks affect large parts or even the entire network. Due to their severe impact, in the further course of this work the focus is directed to these topology attacks and especially to defense techniques against version number and rank spoofing as well as rank replay attacks.

4 Topology Protection in RPL: VeRA

The version number and rank spoofing attacks denote effective attacks that are launched by an insider attacker. The version number attack enables an adversary to start a global repair. If done repeatedly, this type of attack disturbs the network communication and exhausts the resources of nodes. In a rank spoofing attack the adversary redirects traffic toward himself by lowering his rank beyond the allowed limits.

A countermeasure against these topology attacks is proposed by Dvir et al. [6]. They introduce the *Version Number and Rank Authentication* (VeRA) protocol for RPL which aims at detecting version number and rank spoofing attacks by applying one-way hash chains to protect the version number and ranks. While a version number attack can be sufficiently subverted, VeRA remains vulnerable to topology attacks by rank spoofing [8, 9, 61]. The first vulnerability allows an attacker to create a forged rank hash chain and to claim *any* rank in the topology. In a second attack, the adversary replays the rank of his parent and thus moves *one* rank level up in the hierarchy.

In this chapter, two defense techniques [8, 9] are proposed that defend against these vulnerabilities and make VeRA resistant against rank spoofing and rank replay attacks. An encryption chain is introduced that prevents an attacker from creating a forged rank hash chain. A further challenge-response scheme based on the secret knowledge of propagated elements of the rank hash chain detects a rank replay attack. However, it is shown that the isolation of the attacker remains a challenging task, and a first impression to its solution is given.

4.1 Overview of VeRA

The topology authentication scheme VeRA prevents version number attacks and rank spoofing by using one-way hash chains with a fixed length. The principle of VeRA is based on two key properties of hash chains. First, a node that holds a hash element cannot efficiently compute the inverse and thus a prior element of the hash chain. Secondly, it is easy to compute the end

Symbol	Definition
i	Index for Version Hash Chain $(0 \dots n)$
j	Index for Rank Hash Chain $(0 \dots l)$
l	Index of last Rank Hash Element
n	Index of last Version Hash Element
V_i	i-th Element of Version Hash Chain
$R_{i,j}$	j-th Element of Rank Hash Chain for i-th Version
r	Random Seed for Version Hash Chain
x_i	Random Seed for Rank Hash Chain at Version i
$h(\cdot)$	One-Way Hash Function
$enc_k(\cdot)$	Symmetric Encryption of $\cdot$ with Key k
$dec_k(\cdot)$	Symmetric Decryption of $\cdot$ with Key k
c_i	i-th Element of Encryption Chain

Table 4.1: Glossary of Notations

of the chain if the length is known. These properties allow for a node to validate the version number and the rank of its parents as follows.[1]

VeRA generates hash chains to represent all version numbers and the ranks for each version. The hash chains are created by repeatedly using the output of a hash function $h(\cdot)$ as new input for the same hash function, starting with a random seed as visualized in Table 4.2. A reversed hash chain for the version numbers is created where each link of the chain $V_n \dots V_0$ denotes a specific version. In addition, a rank hash chain is created for each version where a rank is implied by the index of each link $R_{i,0} \dots R_{i,l}$. A single rank or version hash element, respectively, is thus denoted by the formula:

$$V_i = h^{n+1-i}(r), \quad R_{i,j} = h^{j+1}(x_i) \tag{4.1}$$

Seed r is chosen once and initializes the version hash chain. Since each version has a separate rank hash chain, the random value x_i is chosen uniquely for each version.

In the bootstrapping phase, the root node creates a message authentication code $\text{MAC}_{V_1}(R_{1,l})$ of the last element of the next version's rank hash chain with the corresponding version hash element as secret key. Furthermore, the root generates a signature $\{V_0, \text{MAC}_{V_1}(R_{1,l})\}_{sign}$ of the MAC and the last element of the version hash chain V_0. The version hash element is later used to verify each version update. The MAC creates a correlation between V_1 and the last

[1] The used abbreviations are summarized in Table 4.1.

$h(r) = V_n$	$\xrightarrow{h(V_n)}$	V_{n-1}	$\dots$	V_1	$\xrightarrow{h(V_1)}$	V_0
x_n		x_{n-1}	$\dots$	x_1		
h(x_n)		h(x_{n-1})		h(x_1)		
$\downarrow$		$\downarrow$		$\downarrow$		
$R_{n,0}$		$R_{n-1,0}$	$\dots$	$R_{1,0}$		
h($R_{n,0}$)		h($R_{n-1,0}$)		h($R_{1,0}$)		
$\downarrow$		$\downarrow$		$\downarrow$		
$\vdots$		$\vdots$	$\dots$	$\vdots$		
h($R_{n,l-1}$)		h($R_{n-1,l-1}$)		h($R_{1,l-1}$)		
$\downarrow$		$\downarrow$		$\downarrow$		
$R_{n,l}$		$R_{n-1,l}$	$\dots$	$R_{1,l}$		

Table 4.2: Creation of the Version Number and Rank Hash Chains in VeRA – The authenticated RPL topology creation begins in version V_1, since version V_0 initializes the protocol.

element of the rank hash chain used in version 1. The signatures reliably authenticate the root as creator of the version number hash chain.

In the initialization phase, the root node disseminates these values to all children:

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

If the signature is successfully verified, the receiving nodes accept and store the parameters V_0 and $\text{MAC}_{V_1}(R_{1,l})$. In each version update the root node disseminates the parameter

$$\langle V_i,\, \text{MAC}_{V_{i+1}}(R_{i+1,l})\rangle \qquad .$$

A node receives these parameters, e.g. in a DIO along with other parameters such as numeric rank and version number. A node thus receives the version hash element V_i to verify the version number by checking if $h^i(V_i) = V_0$ holds. In other words, it hashes the received version hash element i-times which must result in the end of the version hash chain V_0 or the version number is invalid. Furthermore, a node uses V_i as key to verify the rank of its parent. The parent claims an arbitrary rank that is represented by one of the rank hash elements. The propagated rank of the parent is denoted by j, so that the child performs $l - j$ hash operations

on the received rank hash element to obtain the end of the chain. The MAC which is now verifiable by V_i authenticates the end of the rank hash chain:

$$\text{MAC}_{V_i}(R_{i,l}) = \text{MAC}_{V_i}(h^{l-j}(R_{i,j})) \tag{4.2}$$

If both version number and rank validation are successful, the ranks of the parent and version number are verified according to VeRA. A node creates its own rank by performing additional hashing operations on the parent's rank hash element. The number of hashing operations depends on the objective function in use. The node propagates its rank hash element in its DIO transmissions.

4.2 Attacks against VeRA

The VeRA approach successfully prevents a version number attack in which an adversary illegally propagates an increased version number. However, VeRA is subject to two topology attacks that enable an attacker to create a forged rank hash chain or replay the rank of its parent.

4.2.1 Version Delay Attack

In VeRA the correlation between a version hash element V_{i+1} and the last link in the rank hash chain is established by a cryptographic message authentication code $\text{MAC}_{V_{i+1}}(R_{i+1,l})$ with the version hash element used as secret key. In version V_i the MAC for the next version $i+1$ is disseminated without the key V_{i+1} that is required to create the MAC. Due to the one-way property of the hash function, the adversary cannot reconstruct V_{i+1} from V_i with feasible effort and thus cannot create a valid MAC prior to the release of V_{i+1}. The MAC and the version hash element of the first version are signed, so that the first rank hash chain is securely linked to the first version.

However, this correlation can be broken in subsequent version updates. Subsequent MACs are not signed and solely rely on the secrecy of the required key V_{i+1}. An attacker exploits the postponed propagation of the security key in a *version delay attack*. To launch such an attack, the adversary withholds two subsequent version updates. He delays version V_i and its contained $\text{MAC}_{V_{i+1}}(R_{i+1,l})$. The attacker prevents honest nodes from receiving the version update by performing a selective forwarding or collision attack on the DIO messages (see Sec. 3.4).

He waits until he receives the next version update and obtains V_{i+1}. Next, he creates a forged rank hash chain $R'_{i+1,l}$ and computes its $\text{MAC}'_{V_{i+1}}(R'_{i+1,l})$. He releases the previous version V_i in which he replaces the original MAC by the forged MAC$'$. In this version he is still bound to the original rank hash chain because the MAC that authenticates the rank hash chain has been propagated in version $i-1$. Finally, when he releases version $i+1$, he may claim any rank of the chain. This is possible since he holds the entire hash chain $R'_{i+1,l}$ which is verifiable by the forged MAC$'$. To continue the spoofing attack in consecutive versions, he delays each subsequent version update to obtain key V_{i+1}, before releasing V_i.

For a child node it is impossible to detect this attack, because all information is verifiable according to the VeRA protocol. In contrast, nodes that are closer to the root than the attacker, still use the original hash chain and detect the use of an overly lowered rank with respect to the traffic direction. To avoid detection, the attacker uses the true rank to communicate with his parents and the forged rank for its sub-DODAG. As a result, the attacker resides in two versions at the same time: in the real version propagated by the root and in the manipulated version of the attacker. However, a node in his sub-DODAG may receive the version update from an unrelated neighbor, thus migrating to the new version and escaping the attack. The attacker may prevent this by choosing a position in the DODAG where he denotes a bottleneck.

4.2.2 Rank Replay Attack

VeRA is also vulnerable to rank replay attacks. The vulnerability consists in a node revealing its rank hash element to prove its rank. This proof is, however, not bound to a node and thus easily replayed by an attacker.

In VeRA a parent sends all information required for rank authentication to its children: its numeric rank j and the correlating rank hash element $R_{i,j}$. An honest child node selects rank $j+1$ and creates $R_{i,j+1} = h(R_{i,j})$. In its advertisements the honest child propagates a verifiable rank $j+1$. In a rank replay attack, a malicious node does not create $R_{i,j+1}$ and simply forwards the rank hash $R_{i,j}$ of its parent and thus claims rank j instead of $j+1$. Children of the attacker successfully verify the rank according to VeRA. Furthermore, the effect of the rank replay attack propagates to the children of the attacker as they select rank $j+1$ instead of $j+2$ and thus seemingly denote a more attractive next hop to their children in turn.

Rank Hash Element	Applied Key	Cipher
$R_{n,l}$	–	$c_n = R_{n,l}$
$R_{n-1,l}$	$R_{n,l}$	$c_{n-1} = enc_{R_{n,l}}(R_{n-1,l})$
...	...	...
$R_{1,l}$	c_2	$c_1 = enc_{c_2}(R_{1,l})$
$R_{0,l}$	c_1	$c_0 = enc_{c_1}(R_{0,l})$

Table 4.3: Creation of the Rank Encryption Chain

4.3 Defense Techniques

The vulnerabilities of VeRA allow an attacker to spoof any rank by delaying version updates or to claim the rank of his parent in a rank replay attack. The attacker improves his position in the topology and thus redirects traffic toward himself. This section proposes a prevention technique for the version delay attack as well as a detection and mitigation technique against rank replay.

4.3.1 Version Delay Countermeasure

The version number hash element in VeRA is linked to the corresponding rank hash chain to prevent an adversary from advertising a forged rank hash chain in a given version. Section 4.2.1 showed that this correlation dissolves after the first version update. This vulnerability is exploited by an attacker that delays a version update and thus obtains the key to authenticate a forged rank hash chain. The introduction of a *reversed encryption chain* defends against such an attack.

The encryption chain is created as part of the initialization phase of VeRA. The root therefore creates the version number hash chain and all rank hash chains first. Next, the last rank hash element of each chain $R_{n,l} \ldots R_{0,l}$ is encrypted as depicted in Table 4.3. The encryption begins at the last version n, so that $R_{n,l}$ denotes the first key. This key is used to encrypt the next last rank hash element $R_{n-1,l}$. The resulting cipher is used as key to encrypt the next rank hash $R_{n-2,l}$ and so forth. In general, each cipher $c_n \ldots c_0$ is created by the formula:

$$c_i = enc_{c_{i+1}}(R_{i,l}) \tag{4.3}$$

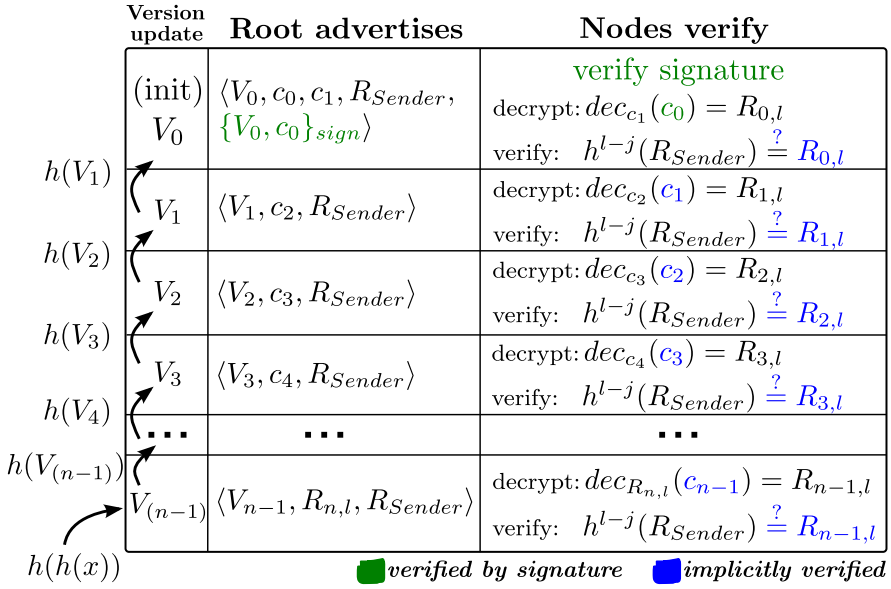


Figure 4.1: Verification of the Rank Encryption Chain

After the initialization phase, the root signs the last link of the encryption chain c_o and the version number hash element of the first version V_0. The root advertises the message

$$\langle V_0,\ c_0,\ c_1,\ R_{Sender},\ \{V_0,\ c_0\}_{sign}\rangle$$

to all children. It contains the version number hash V_0, the last two ciphers of the encryption chain c_0 and c_1 the rank hash element of the sender R_{Sender}, and the signature. Cipher c_1 is included in the initial message as well to allow rank validation and the creation of the topology upon reception of the first message. All further ciphers $i + 1$ are sent in each subsequent version update. The rank verification process is illustrated in Figure 4.1 and works as follows.

A node receives the initial message and verifies the signature. Upon success, it stores all values that it has received. The rank is verified by decrypting c_0 using c_1 as key. The decryption is denoted by the formula:

$$R_{i,l} = dec_{c_{i+1}}(c_i) = dec_{c_{i+1}}(enc_{c_{i+1}}(R_{i,l})) \tag{4.4}$$

The node validates that $R_{0,l} = h^{l-j}(R_{Sender})$ holds. In each subsequent version update, the node obtains V_i and checks the version number by calculating $h^i(V_i) = V_0$. Further it obtains cipher c_{i+1} which is used to decrypt the cipher c_i that it received in the last version. Note that

cipher c_0 is protected by the signature and all subsequent ciphers are implicitly verified since they are deduced from the initially signed cipher.

4.3.2 Rank Replay Countermeasure

In a rank replay attack, the adversary exploits the use of hash chains in VeRA. The attack is based on the fact that a node is required to reveal information to allow the verification of its rank that in turn can be replayed to a third party to claim the rank of that node. This section proposes countermeasures against such a rank replay attack in VeRA. First, a detection technique for a rank replay attack is presented that is based on a challenge-response. The isolation of the attacker is discussed subsequently.

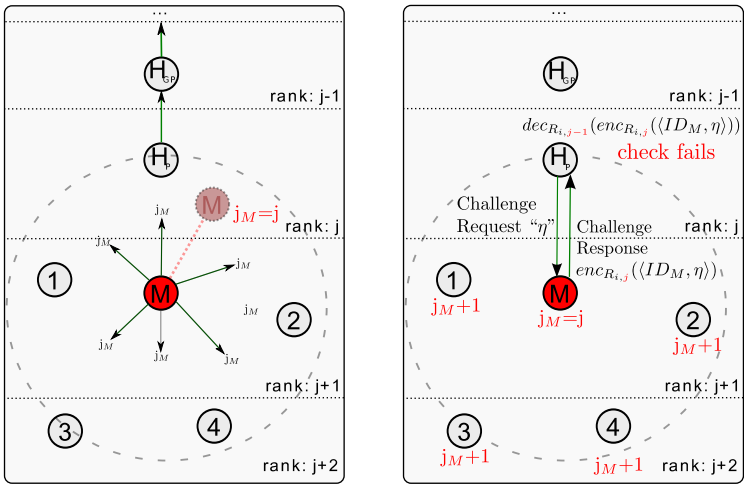
Detection of a rank replay attack To detect a rank replay attack, the proposed scheme is composed of the following techniques:

- challenge-response procedure
- local rank announcement

The *challenge-response* procedure is based on the distribution of the elements of the rank hash chain. In VeRA, an honest node of rank j holds two valid rank hash elements: The hash element of its parent $R_{i,j-1}$ to verify the rank of that parent, and the rank hash it computes for its own advertisements $R_{i,j}$. The one-way property of hash functions prevents any node from obtaining the rank hash of a grandparent $R_{i,j'}$ where $j' < j - 1$. Hence, an adversary of rank $R_{i,j+1}$ that replays rank $R_{i,j}$ of its parent cannot construct the rank hash element of its grandparent $R_{i,j-1}$.

The key concept of the challenge-response is to take advantage of the grandparent's hash element as *secret knowledge* shared between two honest neighbors of equal rank. A node therefore challenges a neighbor with the same rank to encrypt a random number with the grandparent's hash element as secret key. As only an honest parent knows this hash element, it can provide a valid response and thus prove the legitimacy of its rank. The attacker on the other hand only has access to the replayed hash element of the challenger and thus cannot provide such a response.

A detailed illustration of the challenge-response procedure is given in Figure 4.2. Node M replays rank j of its parent H_P which challenges M to prove its rank. H_P therefore sends a random nonce η to M. The random nonce prevents a message replay of a priorly solved challenge. Furthermore, M must add its identifier to the packet to ensure that M has solved



(a) Replay Attack (b) Challenge-Response

Figure 4.2: Rank Replay Defense by Challenge-Response – (a) Attacker M replays the rank announcement via multicast thus falsifying the local topology. (b) Its parent H_P receives the replay as well. The honest node challenges the attacker by sending a random nonce η. The challenge can only be solved if M has a relation with a node of rank $R_{i,j-1}$.

the challenge independently. This prevents M from challenging another neighbor with the nonce provided by H_P.

To pass the challenge, M has to encrypt the message $\langle ID_M, \eta \rangle$ using the rank hash element of grandparent H_{GP} as secret key. This is possible only if M knows a parent of rank $\leq j$ and thus has access to the required secret key $R_{i,j-1}$. In the example in Figure 4.2, M has replayed the lowest rank hash element available and is thus unable to provide a valid response. It may either encrypt using the wrong hash element $R_{i,j}$, forge the response in some other way or not reply at all. In either case, the misbehavior is detected by H_P, so that M fails the challenge.

In case M advertises an honest rank $j+1$ and is challenged by a neighbor of equal rank, it provides a valid response $enc_{R_{i,j}}(\langle ID_M), \eta \rangle$ by using rank hash element $R_{i,j}$ as key. The challenger decrypts the message and checks if nonce η and ID_M are correct. Upon success, the challenge-response procedure is finished.

The challenge-response is initiated upon an inconsistency within data traffic. However, assume an adversary that obfuscates a rank replay attack by using two ranks as described in Section 3.4. He replays the parent's rank to child nodes and uses the true rank for upward

connections. Parent nodes that receive his upward traffic discover a consistent rank. If challenged, the attacker passes as he provides a valid response for his real rank. The *local rank announcement* provides the basic technique to pin an attacker on a single rank and to determine when to challenge a node. Each node is therefore required to advertise its rank transparently to all neighbors which store the rank for later comparison (see Fig. 4.2(a)). The announcement ensures that each node only uses one rank, and thus all one-hop neighbors have a consistent view of each other's rank.

Both local rank announcement and challenge-response work together as follows. Consider Figure 4.2(a) in which node M advertises rank j of his parent H_P which stores this rank for neighbor M. Once the adversary sends data traffic to his parent it must use rank j in the RPL packet information. Otherwise H_P drops the message due to the inconsistent rank with respect to M's prior announcement. H_P examines the packet information and detects that M uses an equal rank of j for upward traffic. H_P thus challenges node M which cannot pass the challenge as it does not hold the required rank hash element $R_{i,j-1}$. If a local repair does not remove this inconsistency and M continues to send upward traffic to H_P with rank j, H_P identifies M as attacker.

Isolation of the attacker Now that the culprit of a rank replay attack can be identified by an honest neighbor, the challenging problem remains how to isolate the attacker. Since a node higher in the topology and thus closer to the root detects the adversary, it can simply ignore the attacker. Nodes in the attackers sub-DODAG on the other hand have to be notified separately as they are still unaware of the attack. The detecting node is thus faced with two challenging problems: first, it has to inform the nodes in the sub-DODAG of the attacker and secondly, it has to show proof that their selected parent has in fact replayed a rank.

The first problem is challenging, because the detecting node has no reliable knowledge of the topology and does not know the identity of the affected nodes. This is because in RPL's non-storing mode, nodes do not keep track of downward targets. In storing mode, downward routing tables may be aggregated and thus may not contain explicit information. Furthermore, the attacker may forge or even block the downward routing information to avoid detection, so that even the root node may not be able to reconstruct the routes passing the attacker and thus lead to the affected nodes. The detecting node may send a broadcast alert to reach the infected area. However, at this point the affected nodes have no evidence to trust the alert message which leads to the second problem of showing proof.

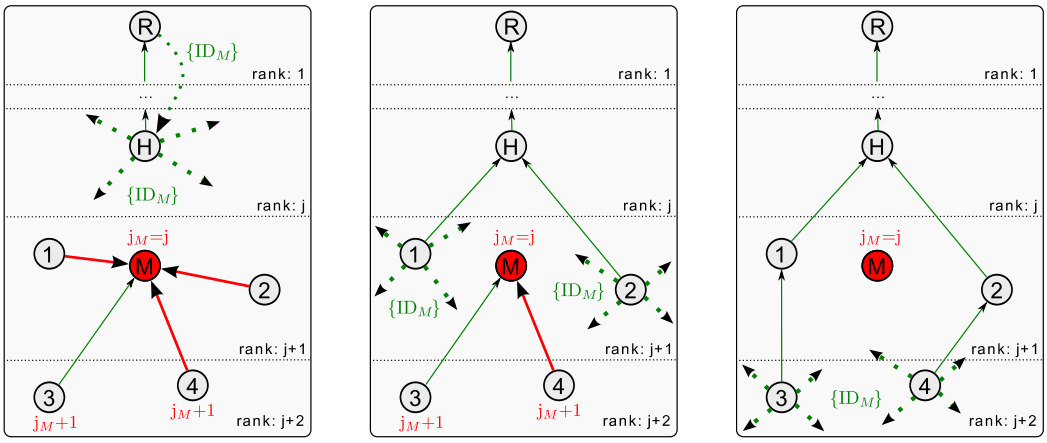
This second problem is difficult to handle, because in consideration of an insider attacker the trust that is established by secret keys is corrupted. Although the detecting node may send

an authenticated alert message, this authentication is meaningless, since an insider attacker sends authenticated messages as well. Relying on such an authentication opens the door for an attacker framing honest nodes. The only reliable source of trust is the root node which is assumed not to be compromised. Hence, the detecting node cannot provide the required proof by itself and must include the root in the validation process.

The detecting node sends an alert message first to the root node which contains the identity and claimed rank of the attacker. From the root's point of view the detecting node may either be honest or may be an attacker trying to frame an honest node. Two approaches have been proposed that deal with the problem of detecting and isolating attacker or defective nodes in RPL. Wallgren et al. [58] send a *heartbeat message* to all nodes within a regular interval. A node that does not reply has no sufficient connectivity to the root and is thus considered defective or malicious. Weekly and Pister [61] propose that the root analyzes the data delivery rate of each node. A node that does not meet a defined rate is considered potentially malicious. This approach takes advantage of the property that LLN routers are typically also hosts that produce data – otherwise a malicious router could not be detected as it would only forward data traffic.

To completely isolate the attacker, both approaches resort to a black- or whitelist that is maintained by the root. A blacklist contains nodes that are unreachable and are therefore not considered for parent selection. In contrast, a whitelist contains all nodes that are reachable. A whitelist is thus filled a priori and starts with a large number of nodes, while the blacklist grows during routing operations. The size of both lists is thus determined by the number of malicious or unreachable nodes. When an attacker prevents its entire sub-DODAG from communicating with the root, all nodes in the sub-DODAG will eventually be added to the list.

However, the notification of a single attacker is only relevant to immediate children of the attacker. An approach that only carries this information to the immediate children limits the overhead to the affected area. Notified children ignore the adversary and either select a different parent or poison their upward routes to indicate the loss of connectivity to the root. In the latter case, due to RPL recovery mechanisms nodes automatically reconnect once alternative paths exist. A possible alternative for isolating the attacker is an alert message that is signed by the root and sent over unicast to the infected area and then propagated in a local flood to the affected nodes. As illustrated in Figure 4.3, the root creates a message with the ID of the unreachable node. The message is signed and sent back to the detecting node. The detecting node adds a small hop-count and broadcasts the message. A node that receives the alert message checks if one of its neighbors matches the ID. Upon a positive match, the node ignores that neighbor for further parent selection and chooses an alternative route or



(a) Alert Propagation (b) Isolation in Progress (c) Successful Isolation

Figure 4.3: Attacker Isolation for Rank Replay Defense – Local flooding of the signed alert message $\{ID_M\}$. In a) node H receives the alert message from the root and begins its dissemination. b) shows nodes 1, 2 isolating M by correctly selecting H as parent. In c) malicious node M is isolated from topology.

temporarily disconnects from the network. The message is further propagated by all receiving nodes. To increase the probability of a successful delivery to all affected nodes, the hop-count is only decreased by nodes that do not maintain a parent with the ID in the alert message. Hereby the message is kept close to the attacker and is dropped if it moves too many hops away.

4.4 Security Evaluation

The proposed defense techniques in this chapter comprise an encryption chain and a challenge-response scheme to provide protection against arbitrary rank spoofing and a rank replay attack in VeRA. This section analyzes the security of these schemes. Prior to the evaluation of these approaches assumptions on the attacker are made.

4.4.1 Attacker Model

The attacker model is adapted from the insider attacker in Section 3.1.2. The presence of one or multiple attackers is assumed that have captured devices of the network and thus function as authorized node. Hereby, the attackers are assumed to be constrained by the capabilities of the captured nodes and thus unable to install malicious hardware like directed antennas to

influence the propagation of their transmissions. The attackers are further unable to use an out-of-band channel or to create multiple identities on a single interface in a Sybil attack.

Two different types of attacker are distinguished: non-collaborating and partly-collaborating attackers. Non-collaborating attackers are distributed in the network and do not communicate or cooperate. partly-collaborating attackers agree upon their actions prior to deployment, however, are unable to communicate after deployment unless within direct communication range.

4.4.2 Reversed Encryption Chain

The encryption chain establishes a correlation between the last rank hash chain elements of each version number. This is achieved by successively encrypting the last rank hash elements with the cipher that results from the last encryption, starting at the last rank hash element of the last version number. The last cipher is signed and propagated in the initial message.

An attacker that forges a rank hash chain with the last element $R'_{0,l}$ for the initial version has to find cipher c'_1, so that $R'_{0,l} = dec_{c'_1}(c_0)$. However, the cipher c_0 is signed, so that finding such a cipher is cryptographically infeasible, if the hash function is secure against pre-image attacks and providing that the symmetric encryption and signature scheme are secure.

The subsequent ciphers are not signed. For the next version update the attacker may forge a rank hash chain $R'_{1,l}$. Since cipher c_1 has been used as key in the verification of the first version, it is implicitly authenticated. The attacker must therefore find a cipher c'_2 so that $R'_{1,l} = dec_{c'_2}(c_1)$ which results in the same effort as in the first version and is thus infeasible.

The same holds for each subsequent version i. An attacker that tries to forge a rank hash chain has to create a cipher c'_{i+1} so that the decryption of c_i correctly results in $R'_{i,l} = dec_{c'_{i+1}}(c_i)$. The attacker is therefore prevented from forging a rank hash chain and from claiming *any* rank.

The protection is provided even when attackers are able to partly collaborate as there is no prior knowledge which the attackers may agree upon and that would allow an attack on the encryption chain.

4.4.3 Challenge-Response Scheme

The challenge-response scheme detects *one or multiple non-collaborating attackers* that use the rank of a parent. Since all children receive the rank hash from their parent for rank verification, an adversary may forward his parent's hash element instead of computing an honest one.

A parent that detects this fraud challenges the attacker to provide the encryption of a nonce. The key for encrypting the nonce is the grandparent's rank hash, which is only known to nodes that have calculated an honest rank. The nonce binds the message to a challenge-response process and detects the replay of a response from a prior challenge-response initiation. When chosen sufficiently large, the probability of a prior challenge-response using the same nonce is negligibly small.

To prevent an adversary from replaying the challenge to a neighbor of same rank, the challenged node also includes its identifier. If the adversary replays the nonce to challenge a different neighbor, the encrypted response will contain the identifier of that neighbor by which an attack is detected.

A parent initiates the challenge-response, if it detects that a node of same rank forwards upward data traffic toward it. To circumvent detection, an attacker that sends data traffic upwards may use two ranks. He therefore replays the parent's rank to its children and uses its true rank for upward traffic. The rank announcement prevents this twofold use of ranks. All nodes transparently propagate their ranks which are stored by all neighbors nodes. An attacker that propagates an honest rank and uses a forged one in its data traffic is thus detected.

However, a chain of k collaborating attackers that are directly connected may circumvent this detection technique as follows. Assume two malicious nodes M_1 and M_2 within direct communication range of each other. M_1 has an honest parent P of rank j and chooses an honest rank $j + 1$. M_2 is located one step down in the topology and receives the rank hash element for rank j by M_1 and replays this rank instead its honest rank $j + 2$. All upward traffic of M_2 is tunneled to M_1 that uses its honest rank to forward the traffic to P. The challenge-response is not initiated because M_1 uses its honest rank and forwards all traffic from its accomplice M_2. Such an attacker constellation circumvents the challenge-response and allows replay of the rank of the first honest parent on the upward path.

4.5 Discussion

The VeRA approach prevents a version number attack in which an adversary illegally initiates a global repair. A rank spoofing is prevented by a rank hash chain for each version number. However, the protection against rank spoofing is circumvented by an attacker that postpones a version update. He thus obtains the required key in the next update and forges a rank hash chain to claim any possible rank. The proposed encryption chain defends against this attack by creating a correlation between all rank hash chains. With this modification VeRA

is secured against rank hash chain forgery when considering multiple non-collaborating or partly-collaborating attackers.

VeRA is also vulnerable to rank replay attacks. A parent must provide its rank hash element to prove its rank. An attacker simply forwards this hash element to his children and claims the rank of his parent. The proposed challenge-response detects this attack by verifying if a node owns the rank hash element that is required for the creation of the propagated rank hash.

The challenge-response is initiated upon an inconsistency of the priorly announced rank and the rank used for data traffic. An attacker that uses a different rank than advertised or uses the rank of his parent for upward traffic is detected by a parent node. Therefore this approach locates partly-collaborating attackers that are not directly connected. Two collaborating attackers with a direct connection circumvent this rank announcement and are able to launch a rank replay attack.

The rank announcement thus increases the number of attackers required for rank replay attacks from one to two or more. Furthermore, for a successful detection this approach requires an attacker to actually forward data traffic. This requirement could be relaxed by a proactive approach in which all nodes challenge all neighboring nodes. Hereby even a *silent attacker* is detected, but at the cost of an increased communication overhead.

A rank replay attack is detected by nodes of equal rank. To reliably isolate the attacker and to mitigate the impact of the attack, the sub-DODAG of the attacker needs to be informed. The root is therefore included in the validation process. Two approaches have been presented for the isolation: Black- and whitelisting and sending an alert message. Black- and whitelisting results in constant overhead for the entire network, so that an approach is desired that only involves the affected nodes. The proposed local flooding of an alert message denotes such an approach, but cannot guarantee to reliably inform all affected nodes.

In conclusion, the modified VeRA approach prevents version number attacks and arbitrary rank spoofing by multiple attackers that may even partly collaborate. For non-collaborating attackers, this approach can detect a rank replay attack. However, the isolation of the attacker remains a challenging task. A solution to this problem is introduced in the next chapter by TRAIL, in which children test their parents independently, so that an attacker is reliably detected and isolated without straining uninvolved nodes.

5 Topology Protection in RPL: TRAIL

This chapter introduces the rank and version number attestation scheme TRAIL (*Trust Anchor Interconnection Loop*) [9]. TRAIL provides a lightweight and generic approach by which children independently verify the rank of their parents. Hereby TRAIL does not rely on the use of expensive cryptography and only resorts to a single signature per rank validation.

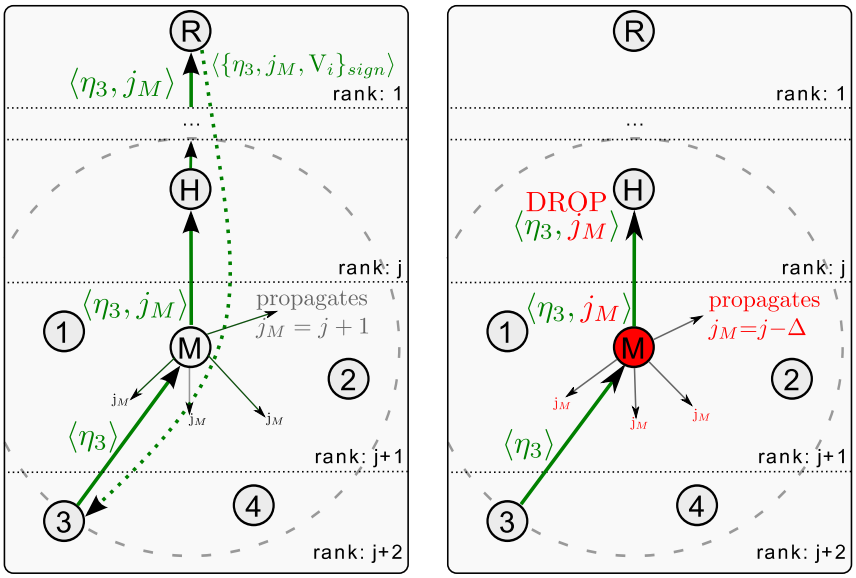
The key idea of TRAIL is to establish the root as *trust anchor*. A node therefore verifies the consistency of ranks on an upward path by sending an *attestation message* toward the root. The message undergoes rank validations on every hop and is dropped upon a violation. If the message successfully travels to the root, the node has a valid upward path based on consistent ranks. A positive attestation is provided by a digital signature of the root.

The rank validation relies on the intrinsic routing behavior of RPL. Since nodes only verify the signature, the major workload is shifted to the root node. As essential part of the LLN, the root is typically equipped with sufficient resources that allow to carry out these more complex tasks.

This chapter introduces the features and characteristics of TRAIL. Since the rank validation requires each node to send a message to the root, an approach is proposed that aggregates all validations and thus decreases the message overhead. An evaluation of TRAIL examines its scalability and security properties. A discussion of TRAIL in comparison to the modified VeRA approach concludes this chapter.

5.1 Concept of TRAIL

TRAIL provides a protection technique against version number attacks and rank spoofing as well as rank replay protection. Rank spoofing is detected by an attestation message that contains validation parameters such as ranks and nonces and is sent toward the root. Each hop performs rank validations to ensure consistent upward progress of the message. The root node takes on responsibility for providing the required trust and acknowledges the arrival by a digital signature. The reception of the signed message provides proof for monotonically increasing ranks on the upward path. By including the version number into the attestation message, nodes verify the propagated version as well.



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

Figure 5.1: Principle of TRAIL Rank Validation – Node 3 sends a nonce, η_3, to M for rank validation. In a) M announces its true rank. The message is signed by the root and thus validates M's rank. In b) M spoofs its rank by $\Delta \geq 0$. Since $H's$ rank $j_H \leq j_M$, the message is dropped.

Rank replay protection is provided by the local rank announcement introduced in Chapter 4. Each node therefore advertises its rank to all neighbors, so that an adversary that replays its parent's rank is detected. In the following both techniques are described in detail.

5.1.1 Rank Spoofing Protection

TRAIL provides rank spoofing protection by an attestation message that is sent toward the root. The reception of the signed response affirms consistent ranks on the upward path. To outline the principle of TRAIL, first the verification of a single path is described.

Key concept – single path validation The basic idea of TRAIL is that each node sends an individual attestation message to the root node. As depicted in Figure 5.1(a), assume that node 3 intends to select M as parent: Before accepting it as parent, node 3 starts the validation process by sending a random nonce η_3 to M. Node M creates an attestation message $\langle \eta_3, j_M \rangle$, containing the nonce and its rank j_M which is sent to the root. On each hop, a node performs

a rank validation by checking whether the rank in the attestation message is monotonically increasing and therefore greater than its own.

M sends the attestation message upwards to its parent H. Before accepting the message, H checks the rank in the message. If H detects a rank smaller or equal to its own rank, it simply drops the message and thus ends the validation process. Since H's rank j is lower than rank $j + 1$ contained in the attestation message, H forwards it to its parent. On every hop the message is subject to the same rank verification until it eventually arrives at the root. The root performs the rank validation one last time and upon success includes the current version number VN_i and a signature of all values $\{\eta_3, j_M, V_i\}_{sign}$. The signed message is sent back to node 3.

On the way back, a receiving node forwards the signed attestation message only if the contained rank is greater than its own to detect an attestation message that has been altered at lower rank levels. Once node 3 receives the signed response, it verifies the signature and matches its nonce η_3, rank j_M and the version number. If all parameters are successfully verified, M has proven a valid upward path with only consistent ranks and a valid version number. Node 3 selects M as parent.

Any node that discovers an inconsistent rank within the attestation message or the rank announcement discards the message as depicted in Figure 5.1(b). A node that receives the signed message and discovers that the rank in the attestation message is not the one claimed by its parent or fails to match its nonce, continues by selecting a different parent or disconnects from the DODAG.

All nodes test their parent and thus recursively check for inconsistent ranks on the entire upward path. However, this single path validation requires all nodes in the network to send at least one attestation message as well as forward all messages from their sub-DODAG. Hence, this approach scales linearly with the number of nodes in the network. To optimize the scalability of this procedure, it is turned into a scalable path validation in which rank validations are aggregated.

Scalable path validation To decrease the number of messages during the validation process, all nonces can be aggregated in one message. This scalable path validation is initiated by the leaf nodes. The validation process includes an attestation message that travels upwards toward the root. To enable each node to match its nonce and thus to verify the rank of its parent, a monotonic rank order within the attestation message is required. This is done by storing all nonces in an array at a specific index. The index represents the rank of the parent. Each node therefore writes all received nonces in the array at the lowest index. It then sends the

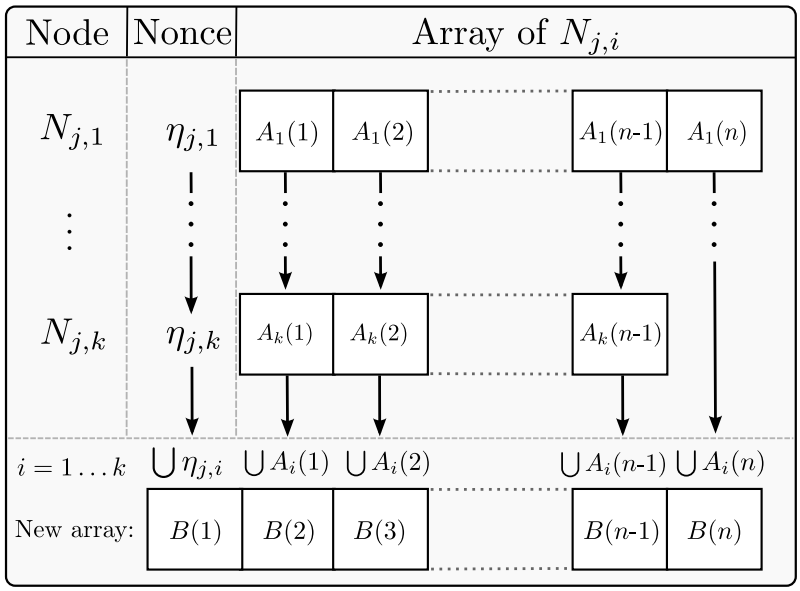
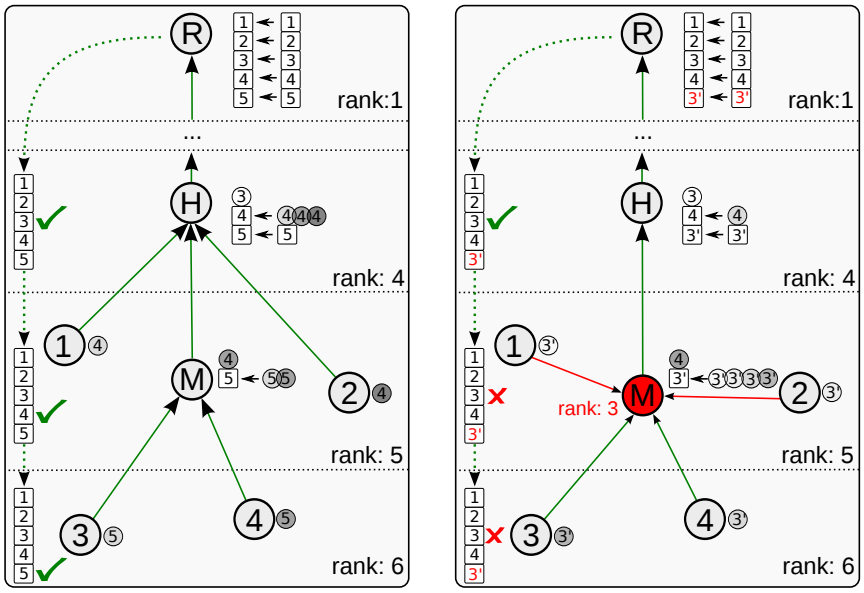


Figure 5.2: Merging of Nonces inside a Node – In a) the received Bloom filter arrays $A_1 \dots A_k$ of nodes $N_{j,1} \dots N_{j,k}$ of rank j are united into the new array B starting at its 2nd index position. $B(1)$ holds the united received nonces $\eta_{j,1} \dots . \eta_{j,k}$ in a newly created filter v.

aggregated nonces as well as an own nonce to its parent, which proceeds in the same way. Hence, when the array arrives at the root, nonces at a specific index of the array verify nodes of same rank.

In detail, each node $N_{l,k}$ that discovers that it has no children, sends a random nonce $\eta_{l,k}$ to its parent. Once the parent has received all nonces $\{\eta_{l,k}\}_k$ of all children, it aggregates them in a single array element. The index of the array element will later denote the rank of the parent. To minimize storage and transmission requirements, the nonces are inserted in a Bloom filter before writing them into the array element. Further, the parent creates a single nonce to verify its own parent. Both, nonce and array, are stored for later comparison and forwarded to the grandparent.

The grandparent receives a single nonce and an array from each immediate child. Since the tree may be unbalanced, the received arrays do not necessarily have the same length. Therefore, the grandparent aligns the received arrays at the lowest index and aggregates the containing Bloom filters, as depicted in Figure 5.2. All Bloom filters at index $A_i(1)$ are aggregated and written into a new array B at index 2. All further indices $A_i(k)$ for $i = 1 \dots k$ are merged into $B(k+1)$. Lastly, an empty Bloom filter is inserted at $B(1)$ in which all single



(a) Successful Validation (b) Failed Validation

Figure 5.3: Principle of Nonce Aggregation in TRAIL – In a) nonces (circle) and arrays (square) are aggregated in the DODAG. The number denotes the index of the array and thus the rank of the parent. Nodes match their signed nonce at the index that denotes the parent's rank. In b) M propagates a false rank of 3, so that nodes 1 and 2 select M as parent. The verification fails, because nodes cannot match their nonces as they move to the wrong index.

nonces are aggregated. The grandparent creates a single nonce and stores it along with the array for later comparison. The grandparent then sends both array and nonce to its parent.

The message travels hop-by-hop toward the root node. Once the attestation message arrives at the root, it includes the version number and a signature of the entire message. The signed attestation message is propagated to all nodes in the network. On reception, a node verifies the version number and the rank of its parent by checking the array at the corresponding index and by matching its nonce in the Bloom filter. Further, it checks that no second array element contains the same nonce and that no nonces have been removed by comparing the signed array with the priorly stored array that it has forwarded. If all of these verifications are successful, the nonces and array elements have not been manipulated or reordered and the ranks on the upward path monotonically decrease toward the root. Figure 5.3 shows a successful rank validation and details how the verification fails when an attacker claims a false rank.

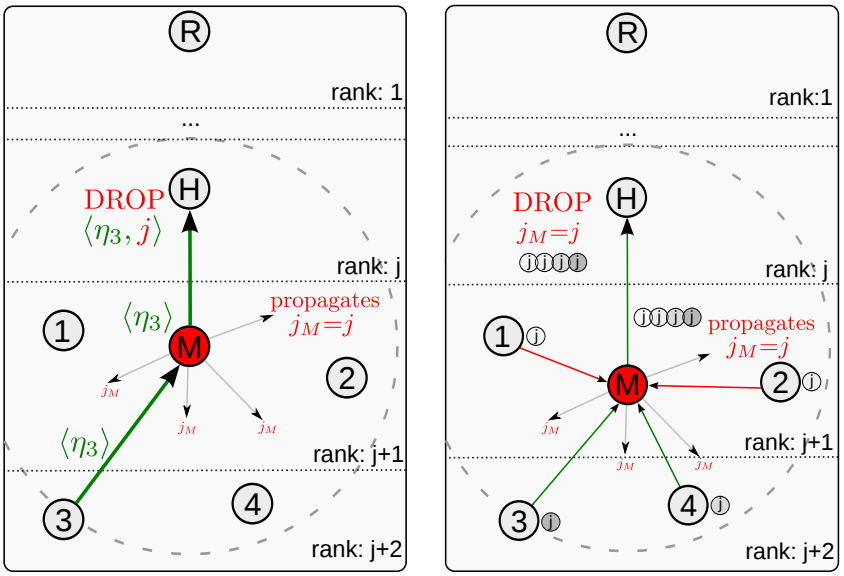
If any of the verifications fail, the node discards the message and chooses an alternative parent if available. If it has no optional parents, it disconnects from the DODAG. In this way an arbitrary rank spoofing is detected by each node independently, so that the attacker can be completely isolated. However, this approach by itself is susceptible to rank replay attacks and requires further protection as depicted in the next section.

5.1.2 Rank Replay Protection

In a rank replay attack, an adversary claims the rank of its parent. In TRAIL the adversary replays the parent's rank simply by replaying the attestation request of its children. Each node receives a nonce from its children. An adversary that claims his parent's rank does not include the received nonces in the attestation message, but replays the nonce to its parent. The parent, unwitting of an attack, authenticates its rank to the attacker. Once the attacker obtains a signed attestation message, he replays this message to its children. Child nodes do not detect that the message originates from the attacker's parent and thus validate the replayed rank of the attacker.

To detect such an attack, TRAIL uses the local rank announcement introduced in Chapter 4. The rank announcement requires all nodes to transparently advertise their ranks to all neighbors. Neighboring nodes store the rank for later comparison and to decide whether to forward an attestation message or not. Figure 5.4(a) shows the principle of this technique to validate a single path: Attacker M has claimed rank j. Child node 3 sends its nonce to attacker M. However, M does not create an attestation message and replays the nonce to its parent H which assumes that it is being tested by M. Before forwarding an attestation message $\langle \eta_3, j \rangle$, H checks whether the priorly advertised rank of M is greater than its own. Since $j_M \leq j$, H drops the nonce. Attacker M is thus unable to provide a signed attestation message containing η_3 and rank j to 3. Hence, node 3 ignores M for parent selection. The same principle applies to the scalable approach as illustrated in Figure 5.4(b). Malicious node M replays the nonces of its children to H. The nonces are dropped by parent H due to the equal rank in M's rank announcement.

The rank replay protection scheme prevents an attestation message from being forwarded once the rank announcement of a child node has revealed a rank violation. As described in Section 4.4.3 and further discussed in the next section, the rank announcement does not protect against directly connected attackers.



(a) Single Path Validation (b) Scalable Path Validation

Figure 5.4: Rank Replay Protection in TRAIL – In a) M claims the rank of H and replays the nonce η_3 of node 3. H drops the replayed nonce because $j_M = j$. In b) children of M send nonces (circles) to validate M's rank j. M replays all nonces to H. H drops the nonces due to the equal rank in M rank announcement.

5.2 Evaluation of TRAIL

This section evaluates the security properties of TRAIL as well as the message sizes as a result of the use of Bloom filters.

5.2.1 Security Evaluation

TRAIL provides two defense techniques: an attestation message against arbitrary rank spoofing and the local rank announcement to protect against rank replay. The attestation message contains nonces and the ranks of tested parents. The nonces are created by each node to ensure freshness of the message. The rank is represented by the index of a particular array element and ensures strict upward progress within the entire path. The rank announcement detects a parent that replays the nonces of its children to verify a replayed rank. Table 5.1 gives an overview of the properties of both techniques.

Attestation Message		Rank Announcement
Nonce	Rank	Rank
scope: within node	scope: entire path	scope: one-hop
binds message to node	validates rank	ensures single rank per node
prevents rank spoofing (≥ 1 attacker)		prevents rank spoofing/replay (1 attacker)
allows rank replay (≥ 1 attacker)		allows rank spoofing/replay (≥ 2 attacker)

Table 5.1: Properties Rank Validation Techniques in TRAIL

In the following the security of both approaches are evaluated. The attacker model is taken from Section 4.4.1 and thus considers one or multiple non-collaborating attackers and one or multiple partly-collaborating attackers.

Rank announcement The rank replay protection of TRAIL is provided by a local rank announcement of all nodes. The rank announcement successfully detects a single attacker or multiple non-collaborating attackers that replay the rank of their parents. However, two partly-collaborating attackers that are directly connected circumvent the rank announcement and are thus able to replay the rank of the first honest parent.

Assume two directly connected malicious nodes M_1 and M_2. M_1 is located closer to the root and has an honest parent P, while M_2 is located directly below M_1. M_1 propagates an honest rank while M_2 replays the rank of honest parent P. Hereby M_1 hides the illegal rank decrease of M_2 by creating a *communication barrier* between P and M_2. The attackers are able to replay the rank of the parent of M_1. Nonetheless, the attestation message prevents an arbitrary rank spoofing as described next.

Attestation message TRAIL requires each node to maintain a valid upward path to the root. The aggregated attestation message provides the monotonically increasing rank order from the root toward the leaves by an array that stores all nonces. Since the array is created hop-by-hop, a rank spoofing is detected when a node cannot match its nonce at the expected array index.

One or multiple non-collaborating attackers may launch the following attacks:

- deny forwarding the attestation data
- rearrange or insert bogus attestation data
- deny sending a nonce

A node receives k nonces and arrays of already merged nonces $A_{1...k}$ from its children. An attacker may *deny forwarding the attestation data.* Hence, he only forwards the array without merging the nonces or does not forward the array at all. If the attacker does not forward the array or does not merge the nonces, the affected nodes are excluded from the validation process, so that they cannot match their nonce. Consequently, they will ignore the attacker who is thus isolated from the DODAG.

The same holds for an adversary that *rearranges the attestation data.* If he inserts the nonces of his children at the wrong index or shifts the array elements, nonces move to a higher rank. Affected nodes thus cannot match their nonce at the expected array position and ignore the adversary. Alternatively, he may *insert bogus attestation data.* Each new array element will, however, move higher indices to higher ranks and prevent affected nodes from matching their nonce correctly. Note that the attacker cannot move or insert nonces at a lower rank, since he cannot influence the array creation after forwarding it.

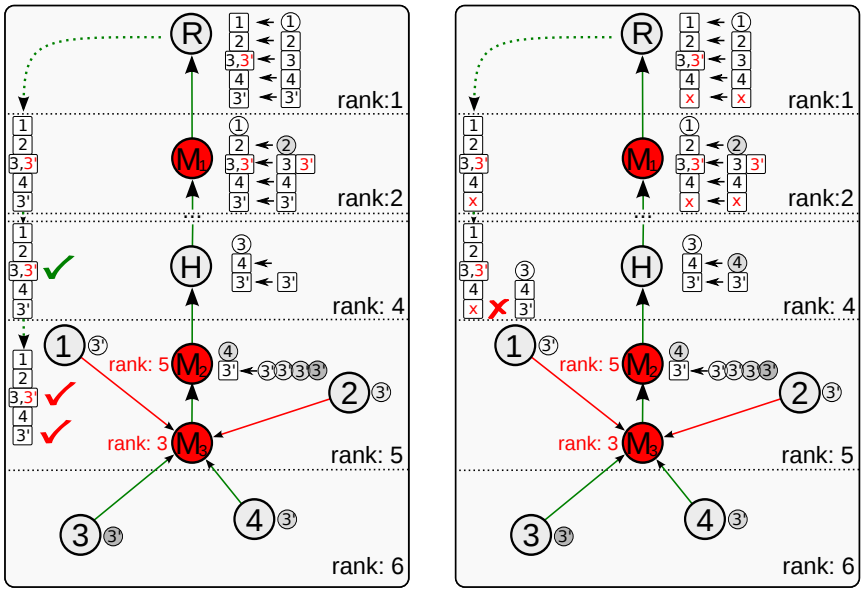
The adversary may *deny sending a nonce.* He merges all k nonces at the correct array position. He only forwards the aggregated array and does not include his own nonce in the attestation message. However, such an attack has no effect since the parent will insert an empty Bloom filter for all single nonces which is either filled with nonces of other nodes or remains empty. The TRAIL validation proceeds without the attacker who cannot verify his parent. Children of the attacker will be able to find their nonce, since they have been correctly merged.

Hence, TRAIL reliably protects against one or multiple non-collaborating adversaries. However, once attackers are able to agree upon their action prior to deployment, they can plan attacks and thus aid each other in circumventing the security. Such partly-collaborating attackers can launch the following attacks:

- rearrange attestation data
- remove attestation data

An attacker that moves the attestation data of child nodes can only move their nonces to a higher rank, since the remainder of the array is created after forwarding. However, three collaborating attackers M_1, M_2 and M_3 on the same upward path can *rearrange attestation data.* As illustrated in Figure 5.5(a), M_3 claims a false rank of 3. Children in its vicinity forward their nonces which are correctly merged by M_3. M_3 tunnels the attestation data to M_2 that forwards it upwards using a consistent rank. Once M_1 receives the attestation message, it *predicts* the real rank of M_3 and merges all nonces at index 5 to index 3.[1] Hereby the array

[1] To predict the correct array position, the attackers estimate the location in advance at which they will attack.



(a) Detection of Duplicate Nonces (b) Detection of Removed Nonces

Figure 5.5: Nonce Duplicate and Removal Detection – Squares denote array elements, circles denote single nonces. Attacker M_1 copies the nonces from the correct array element to the index of the spoofed rank of M_3. In a) Nodes check for duplicates and detect the modification. In b) M_1 removes the duplicate nonces. Honest node H detects missing nonces and drops the message.

element at the index of M_3's real rank is moved to the index of the spoofed rank. In addition, M_1 merges all array elements according to TRAIL. The forged attestation message is sent to the root which signs it.

Once a child of M_3 receives the signed array, it successfully matches its nonce at rank 3. Since each of the copied nonces is now included twice in the array, a node detects a duplicate of its own nonce. Such a duplicate either denotes a false positive of the Bloom filter or an attack. Since the false positive rate f can be chosen arbitrarily small when configuring the Bloom filter, an attack is detected with the probability of $1 - f$. In Figure 5.5(a) nodes 1 to 4 are leaf nodes, so that M_1 only copies one array element. However, for larger trees M_1 has to copy all following elements as well, so that the sub-DODAG of nodes 1 to 4 match their nonces.

This detection technique, however, is circumvented by *removing attestation data* from the array. M_1 therefore removes the duplicate nonces from the Bloom filter in the appropriate

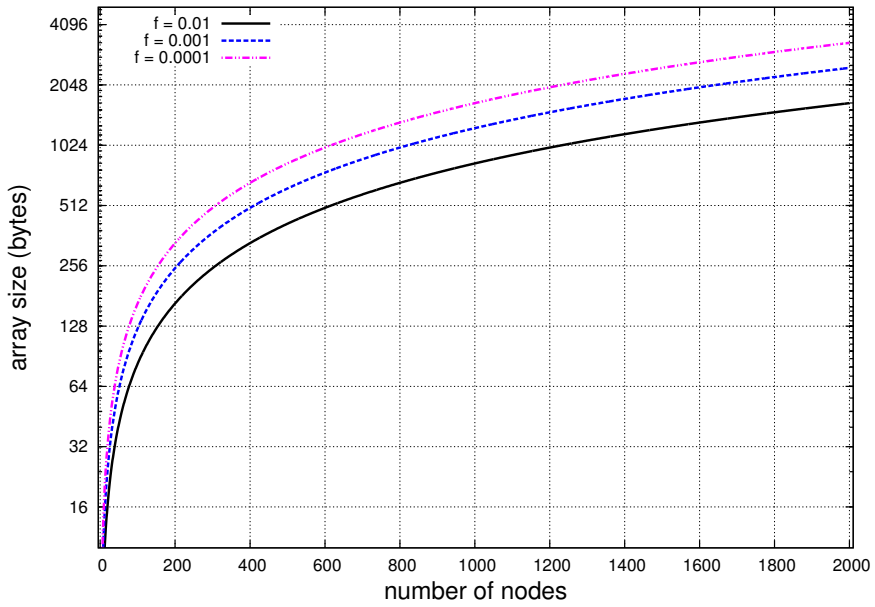
array element by setting all bits to zero, as seen in Figure 5.5(b). However, such a deletion of the nonces is detected by the honest parent H on the upward path. H has priorly merged all nonces, including the array of M_1, and holds the original array which must be a subset of the signed array. By aligning both arrays at the index of its nonce, H detects that bits of the Bloom filter in the signed array have been removed. Node H detects the attack and does not further propagate the array. Furthermore, H may select a different parent, as it knows that a malicious node on the upward path has illegally modified the Bloom filter.

These techniques mitigate multiple partly-collaborating attackers. Nonetheless, they cannot prevent an attacker from obtaining the signed attestation message from a neighbor of an unrelated branch of the DODAG. Honest nodes on such a branch do not drop the forged message, since the removed nonces have not been sent on their upward path. Their original array does not contain the removed nonces. However, M_3 spoofs its rank, so that all neighboring nodes select it as parent. M_3 thereby prunes unrelated branches and decreases the number of nodes from which the other attacking nodes might receive the signed message. M_2 is required to have an additional honest parent which is located on an upward path different from node H. Furthermore, for a successful attack both upward paths must not meet at a common ancestor that is between M_1 and M_2. Such a common ancestor would merge the arrays and detect any removed nonces. Consequently, the probability of the attackers receiving a signed message highly depends on the topological formation.

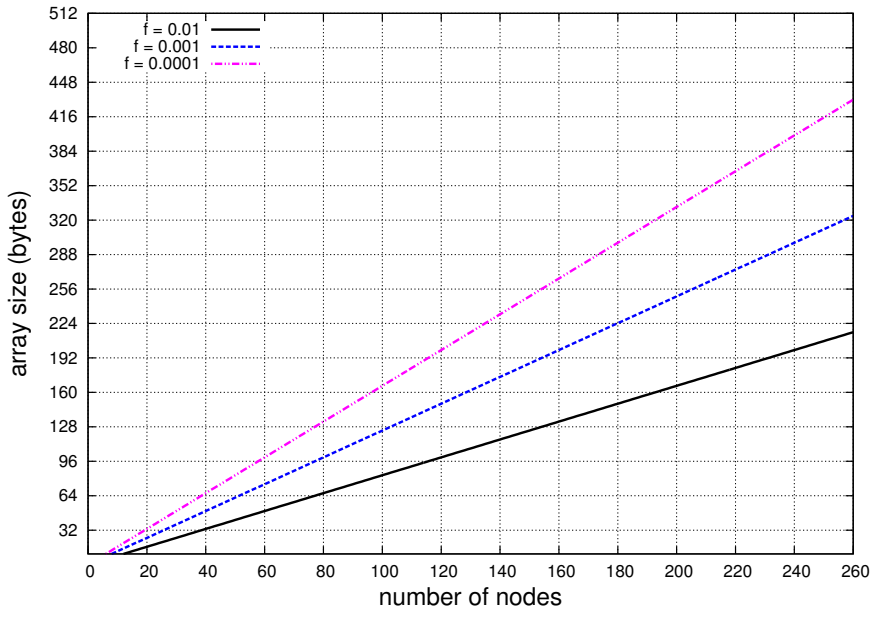
5.2.2 Scalability Evaluation

TRAIL uses Bloom filters to aggregate nonces of all nodes for space efficiency. Each nonce is thus hashed k times with different hash functions, and each hash value is mapped to one bit of the Bloom filter. This technique reduces the size of each nonce to k bits. To minimize the transmission size, a compressed Bloom filter is applied that is optimized for a certain false positive rate. As demonstrated by Mitzenmacher [21], a compressed Bloom filter, in theory, can be compressed to about 70 % of the size of the equivalent uncompressed filter with the same false positive rate. Although not achievable in practice, it denotes the ideal case for the minimum transmission size. Therefore in this work the ideal case is analyzed to identify the constraints of TRAIL.

Figure 5.6 shows the growth of the Bloom filter array under the assumption of optimal compression with respect to different false positive rates. The size of the compressed Bloom filter increases linearly with the number of nodes or number of stored nonces. For a maximum array size of around 100 bytes, TRAIL supports a network of approximately 120 nodes while maintaining a false positive rate of 1 % or around 60 nodes for 0.01 %. It is shown that even



(a) Logarithmic y-Scale



(b) Linear x/y-Scale

Figure 5.6: TRAIL Compressed Array Sizes for Different Error Rates

under ideal conditions the array size grows fairly large, so that TRAIL is applicable for small networks. Especially because the false positive rate denotes a parameter that is critical for the security of TRAIL it should be kept as low as possible.

When applying Bloom filters the size of the filter has to be known in advance. Consequently, nodes that typically have only limited knowledge of the topology, have to configure a Bloom filter that holds all nonces of a corresponding rank. If chosen too small the filter may exceed its capacity before all nonces are aggregated. To allude this risk the uncompressed filter is chosen arbitrarily large. Let's assume a large set of n elements of which only Δ elements are inserted into the (large) uncompressed filter where $\Delta \ll n$. This filter is compressed with a high compression rate, because many bits are still set to zero. With each element that is inserted more bits are set, so that the resulting compressed Bloom filter grows while the size of the uncompressed filter remains the same. Nodes lower in the hierarchy therefore only send small messages, and the message size increases with each hop.

Table 5.2 shows the average and maximum transmission sizes of the Bloom filter array with different configurations of a balanced k-ary tree and different false positive rates f. It can be observed that for a balanced tree of two children per node and a height of 7, the maximum transmission size for 127 nodes results in about 105 bytes and an overall average message size of about 39 bytes.

To analyze the growth of message sizes in TRAIL, one has to consider how the nonces are sent and aggregated in the network. For simplicity it is assumed that leaf nodes include their nonce in a Bloom filter before forwarding, so that they send a Bloom filter containing only one element. Each parent on the next rank level forwards all nonces of its children and its own nonce and so on. However, since the number of nodes toward higher ranks increases for each level, the number of nodes that send small messages is relatively large as opposed to the number of nodes that send large messages. The average upward message size is thus kept relatively small. The completely aggregated message sent by the root has the maximum message size. This message is forwarded by all nodes but the leaves. Leaf nodes only receive this message. The overall average message size is moderate and roughly half of the maximum message size. These values are based on the ideal assumption that a compressed Bloom filter has a size of $z = m * ln(2)$ [21], where m is the size of the equivalent uncompressed filter. Hence, for a compressed Bloom filter with a false positive rate of 1 % each element requires ≈ 6.64 bits.

In practice, the achieved transmission size is constrained by the compression rate and overhead of the applied compression function, computational overhead that results from

Configuration				Message Sizes in Bytes		
f	# Children	h	# Nodes	Avg. upwards	Overall Avg.	Max. Size
0.0100	2	4	15	2.91	6.09	12.46
0.0100	2	6	63	4.30	20.31	52.32
0.0100	2	7	127	5.07	38.54	105.47
0.0100	2	8	255	5.86	74.50	211.77
0.0100	4	4	85	3.09	32.02	70.59
0.0100	4	5	341	3.89	123.59	283.19
0.0100	4	6	1365	4.71	488.52	1133.61
0.0010	2	4	15	4.36	9.14	18.69
0.0010	2	6	63	6.45	30.46	78.48
0.0010	2	8	255	8.79	111.75	317.66
0.0010	4	4	85	4.64	48.03	105.89
0.0010	4	5	341	5.84	185.39	424.79
0.0010	4	6	1365	7.07	732.79	1700.41
0.0001	2	4	15	5.81	12.18	24.91
0.0001	2	6	63	8.60	40.61	104.64
0.0001	2	8	255	11.72	149.00	423.55
0.0001	4	4	85	6.19	64.04	141.18
0.0001	4	5	341	7.78	247.18	566.39
0.0001	4	6	1365	9.42	977.05	2267.22

Table 5.2: TRAIL Average and Maximum Message Sizes – Table shows the average and maximum message overhead in TRAIL for different false positive rates f and a balanced k-ary tree of different heights h.

compression and hashing as well as memory constrains for the size of the uncompressed filter [21].

As an alternative to compressed Bloom filters, scalable filters proposed by Almeida et al. [22] can be used. When applied to TRAIL, the total number of nonces per filter does not have to be known in advance, as the scalable Bloom filter grows dynamically. Lower nodes therefore configure a small filter that only holds a few nonces. On each hop, a node adds a new Bloom filter if necessary that is configured with a tighter false positive rate.

When a node aggregates the Bloom filters it receives from its children, it may have to reorganize these filters before adding them to a single scalable Bloom filter. A parent node may receive Bloom filters with equal false positive rates from its children. If simply added to a single scalable filter, the overall false positive rate will not converge to the desired value. The reason for this is that a scalable Bloom filter comprises of one or more independent

Bloom filters which are sequentially queried when checking for an element. To achieve a false positive rate that converges to a specific value each new Bloom filter must have a tighter false positive rate. Hence, for use of scalable Bloom filters in TRAIL, configuration details still require further research.

5.3 Discussion

TRAIL uses an attestation message and the rank announcement to validate the version number and ranks on the upward paths. Each node sends a nonce to the root which is returned with the digital signature and that provides the required trust for a positive attestation. The rank that is to be verified is represented by the array index at which the nonce is stored. When a node receives the signed attestation message it checks for its nonce at the corresponding array index. If the node cannot match its nonce, it has detected a parent that has illegally lowered its rank. Since the version number is included in the signed message, the node verifies the version number as well. TRAIL provides protection against arbitrary rank spoofing by multiple non-collaborating attackers and two or more directly connected attackers that partly collaborate. Rank replay protection is provided by the rank announcement and only protects against attackers that are not directly connected.

TRAIL also protects against three partly-collaborating attackers and detects the later removal or copying of nonces. However, if the attackers receive the signed attestation message from neighboring nodes of an unrelated branch of the tree, they can deliver a forged and signed message to their children and claim an arbitrary low rank. Although such an attack is highly dependent on the topology formation, it denotes a threat to TRAIL that requires further attention.

The modified VeRA approach follows the same goals by applying hash chains as well as an encryption chain by which nodes successively validate the version number and the rank of their parents. A single digital signature achieves a secured hash chain for version updates and rank validations. Under the assumption that attackers cannot communicate after deployment, the encryption and rank hash chain provide sufficient protection.

Both approaches are susceptible to rank replay attacks by multiple partly-collaborating attackers within direct communication range. Such a constellation circumvents the rank announcement and allows the attackers to replay their parent's rank. In the modified VeRA approach the honest parent does not challenge the attacker, since it only communicates with the one that advertises a consistent rank. In TRAIL the parent node accepts the single nonces that have been replayed by the attacker.

Hence, under the assumption that none of the attackers are directly connected, both the modified VeRA approach and TRAIL reliably detect a rank replay attack. In the modified VeRA approach the detecting parent can simply ignore the attacker. The efficient isolation of the attacker by his sub-DODAG proves to be a difficult task. In TRAIL on the other hand, child nodes verify the consistency of their parent's rank independently and are able to isolate a present attacker. However, while denoting a conceptually sound approach, the resulting message sizes of TRAIL restrict its application domain to small networks, so that future research is required to extend its applicability.

6 Practical Evaluation of TRAIL

The goal of TRAIL is to allow every node to validate the rank of its parent and the version number propagated by the root. TRAIL hereby relies on the exchange of an attestation message which is sent to the root and subject to rank validations on each hop. The root signs the message and sends it back to the validating nodes. The main overhead in TRAIL therefore lies in the transmission of such validation messages. Since each node has to wait for the signed attestation message, the usual operation of the network is postponed.

To evaluate the impact of this delay and to study other characteristics, a proof-of-concept prototype implementation has been developed. Using a small setup of sensor nodes, two experiments have been performed to measure the time it takes to send the attestation message to the root and back to the sender.

This chapter describes the prototype and the experiments in detail and concludes with a discussion of the results.

6.1 Implementation Prerequisites

The goal of the implementation is to obtain representative time measurements that illustrate the delay when deploying TRAIL in an RPL network. The requirements for successful experiments are categorized in requirements for the soft- and hardware and presented in the following.

6.1.1 Software Choices

Since the prototype is implemented to function in an RPL network, it requires an operating system with an RPL implementation. Furthermore, to obtain representative results, the physical and MAC (Media Access Control) layer protocol must be deployable for low-power sensor nodes.

The standard *IPv6 over Low-Power Wireless Personal Area Networks* [62] (6LoWPAN), describes how IPv6 packets are sent and received over a low-power link, using the IEEE 802.15.4 [63] technology. 6LoWPAN is the state of the art transmission protocol for RPL, so that the operating system for this work must provide a 6LoWPAN stack.

CPU	Model:	ARM7 TDMI
	Word size:	32 bit
	Frequency:	up to 72 MHz
Memory	RAM:	98 KB
	Flash ROM:	512 KB
Transceiver	Model:	Texas Instruments/Chipcon CC1100
	Frequency:	863 – 870 MHz
	receive/transmit FIFO:	64 byte

Table 6.1: Technical Details of MSB-A2 Sensor Board [67, 68]

Two operating systems were considered for use in this work: Contiki OS [64] and RIOT OS [65, 66]. Contiki is an operating system for sensor platforms and also provides simulation tools. RIOT is a newly introduced operating system that is currently under development by an open community. Both operating systems comprise an RPL and 6LoWPAN stack and are open source, so that they were both candidates for this work. However, RIOT has been chosen as operating system, since it works out-of-the-box with the hardware platform provided by the *Freie Universität Berlin* (FU Berlin) and because of a close cooperation and first hand support from the developers of RIOT OS.

The development of RIOT directly evolved from the operating system μkleos. As to developing the TRAIL prototype, not all required functionality had been imported from μkleos to RIOT, so that in this work RIOT's predecessor μkleos is used. The source code of μkleos is written in the programming language C. The TRAIL prototype is therefore also written in C and adapted to already available functionality of the RPL implementation of μkleos.

6.1.2 Hardware Choices

The prototype is tested on the ScatterWeb MSB-A2 [67] hardware platform developed by the FU Berlin. The MSB-A2 can be powered by USB port or an external power supply. The hardware constraints of these boards are summarized in Table 6.1.

The transceiver of the MSB-A2 boards has a send/receive buffer of 64 bytes [68]. The transceiver functions in the 868 MHz ISM band [67] and is thus compatible with the IEEE 802.15.4 standard [63] and 6LoWPAN. Additionally, the boards are equipped with removable antennas.

μkleos has been developed to run on the MSB-A2 boards, so that the compatibility of μkleos and the boards is sufficiently high. The MSB-A2 currently operates in the DES-Testbed

(Distributed Embedded Systems)[1] at the FU Berlin. It is thereby part of a multi-hop network for long-term studies of various research projects.

6.2 Design of the Prototype

The prototype implements the basic features for single path validation in which all nodes send an attestation message to the root and forward messages on behalf of other nodes. The goal of the prototype is to analyze and evaluate the overhead in delays in a one-hop and two-hop scenario respectively. To allow this evaluation, the TRAIL prototype must be able to send and receive an attestation message and collect the required data for analysis. To describe the design, first a brief overview of the relevant functions of the RPL implementation in μkleos is given into which TRAIL is integrated.

6.2.1 Integration in μkleos

The RPL implementation of μkleos comprises two parts that are relevant for the TRAIL prototype: the main RPL process and the trickle timer. The main RPL process sends, receives and processes RPL control messages that it has received. It determines the type of the message such as DIO, DIS, DAO or DAO-ACK and calls the processing function which performs the RPL operations. The send function for control messages creates an ICMPv6 packet for each message type and passes it down to the link layer. The main functions for TRAIL are included accordingly. A send and receive function is created for the TRAIL attestation message to allow the required functionality.

The second part is the trickle timer which schedules the sending of DIO messages and contains the timer thread for sending DAO acknowledgements. This timer is used by the TRAIL prototype to allow the periodical sending of an attestation message for automated data collection.

6.2.2 Functional Overview

The implementation of the TRAIL prototype follows the structure of μkleos for sending RPL control messages. Hence, a send and process function is included in the RPL process. To function correctly a new send and receive buffer and a structure for the attestation message is defined. The attestation message comprises the parameters required for validation, including a nonce, a rank, the version number as well as parameters such as RPL instance ID and a

[1] `www.des-testbed.net`

sequence number of the attestation message. For returning the message to the sender and to allow the verification of ranks, the source address and a signature are required.

The send function simply copies the parameters into the send buffer and creates an ICMPv6 message. The process-function performs the actual TRAIL logic as depicted in Figure 6.1. On the upward paths the tested node adds its rank and forwards the message upwards. Each node on the upward paths checks the rank and only forwards the message the root node, if the rank is greater than the own rank. The root includes a signature and sends the message back to the source. On the downward path each node checks the source address within the message. If a node detects that it is the originator of the message, if verifies the signature. The message is accepted if the signature is valid, and dropped otherwise. Note that the in case of the prototype the computation of the signature is not considered. Hence, an array of 32 bit integers that sums up to 384 bits is sent to simulate a concise signature such as an elliptic curve signature.

To allow the root to send the response back to the source and to stay independent of the creation of downward routes during the experiments, each node maintains its downward routing table when receiving the attestation message on the upward path. This function does the same as a DAO message, but allows to disable the sending of DAO messages and saves required memory when disabling the DAO timer thread.

6.3 Measurements and Results

The prototype is tested in two experiments that measure the transmission time of the attestation message. The rank validation in TRAIL requires a certain communication overhead: A node sends an attestation message to the root and completes the validation when it receives a version of the message that has been signed by the root. The results of the experiments give an impression of the practical implications of TRAIL in terms of delays due to rank validation. As an exemplary reference value, the time to create the topology in μkleos without TRAIL is used. The following gives an overview of the results of the experiments. Prior to the evaluation of the actual test runs, the setup is described.

6.3.1 Preliminary Outline

Experimental setup The setup consists of MSB-A2 boards that are powered by the USB port of the host system. All boards are flashed with the same version of μkleos that includes the TRAIL prototype implementation.

The boards are connected to the same host system and each device is around half a meter apart from the next device. For the two-hop experiment, the root and the source node are

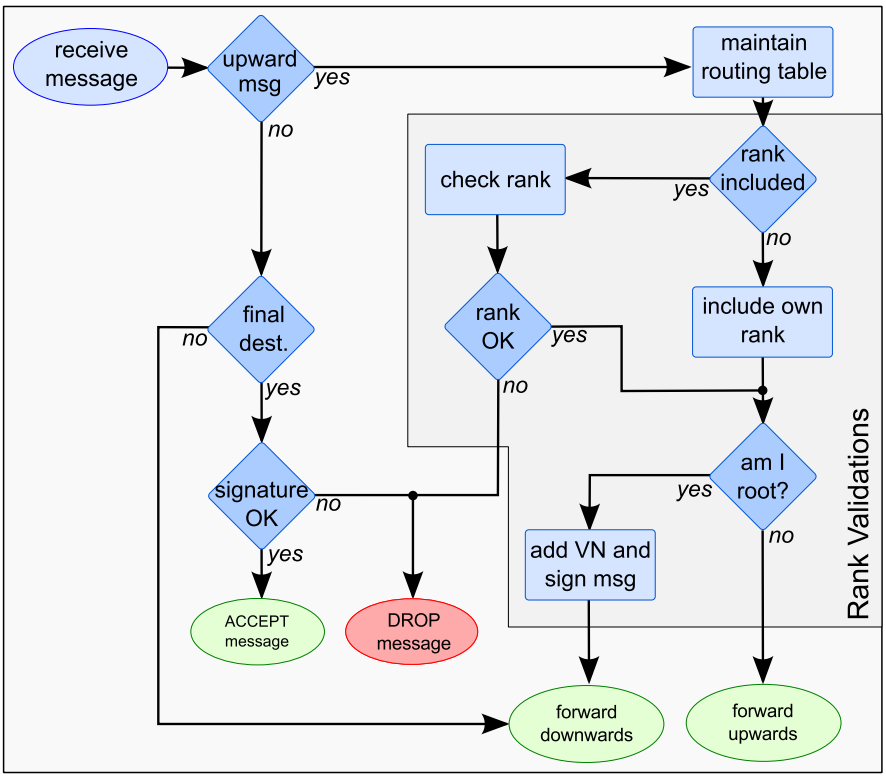


Figure 6.1: Processing of Attestation Message in TRAIL Prototype

connected by the intermediate node. To prevent a parent-child relation between root and source due to the short physical distance, the antennas of both root and source are removed. To further reduce the transmission range, obstacles are placed between source and root. The intermediate node uses the antenna to provide sufficient connectivity with the root and the source node, respectively.

The scheduler for DAO messages has been disabled to prevent memory exhaustion of the boards. The nodes maintain their downward routing table using the information in the attestation message. This is required so that messages can travel back to the source. Furthermore, the parent lifetime has been increased to the maximum value to prevent upward routes from expiring during the experiments if DIO messages are not received frequently enough.

The prototype can be configured so that the source node periodically sends attestation messages. For the timing measurements, the nonce in the attestation message is used as timestamp which is set by the sending node. The timer in μkleos starts to count the microseconds

after the booting. The boards are started sequentially, and the clocks are not automatically synchronized. The delay that results from the sequential boot-up is subtracted from the measurement.

Expected and reference values The goal of the conducted experiments is to determine the time delay that results from deploying TRAIL. In each experiment a total number of 300 messages are transmitted. Each message is sent from a source node to the root and has a payload size of 34 bytes, including ICMPv6 and IPv6 headers. The response from the root contains additional 48 bytes for the signature-dummy leading to a total payload of 82 bytes. Larger message size of the response will result in a higher transmission time. It is expected that the transmission times will increase around a factor of 2 when the hop count is increased by one.

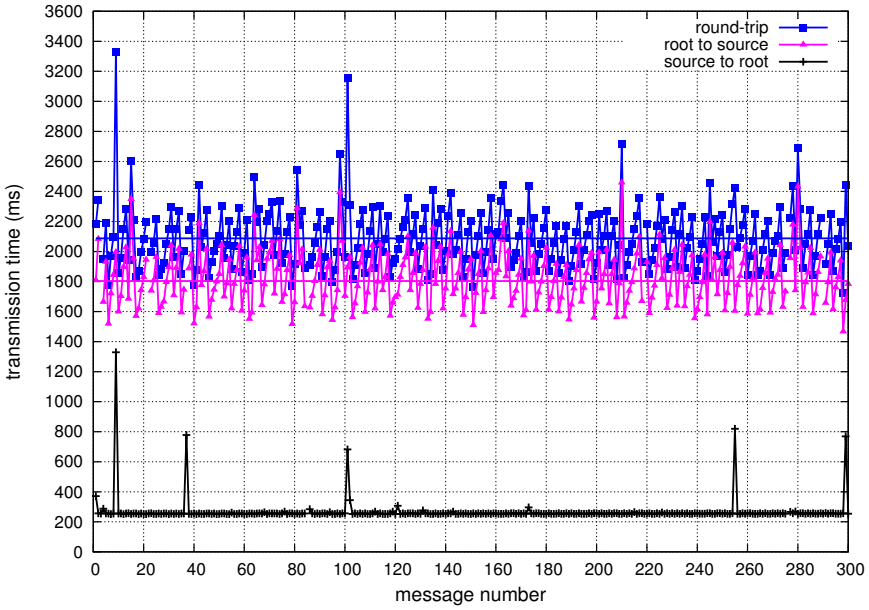
As a reference value, an estimate of the network creation time without TRAIL in μkleos is taken. The trickle timer schedules the sending of DIO messages and thus determines the minimum delay for the creation of the topology. The DIO has a payload size of 44 bytes and its transmission time has been found to be around 1 second.[2] Therefore it takes the first node at least 1 second to select the root node as parent without considering message loss and assuming that devices are started at the same time. The trickle parameters are equal in all nodes, so that the DIO is propagated with a delay of around 1 second per hop.

6.3.2 Experiment 1: Single Hop

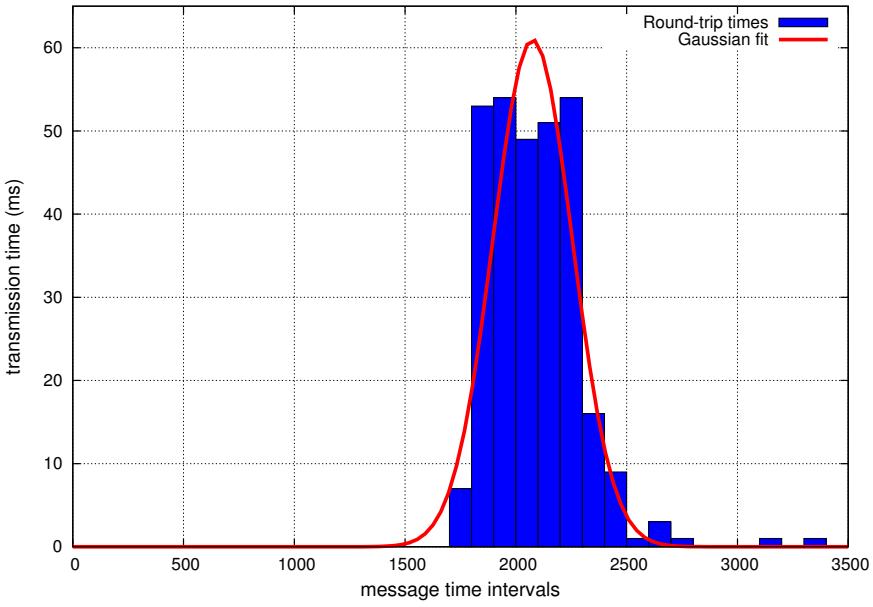
The first experiment uses two nodes: a root and one immediate child node that is the source of the attestation message. The child sends the periodic attestation message every 5 seconds to allow the root node to receive the response before sending the next message. Figure 6.2(a) shows the absolute delays in milliseconds. The crosses (+) indicate the time it took the request from the source node to reach the root. The triangles show the transmission time of the responds containing the signature-dummy from the root to the source. The squares denote the round-trip time of the message and depict the sum of each corresponding cross and triangle.[3]

It can be seen that the time for a message from the source to the root (crosses) is more than factor 5 of the messages back (triangles). The message to the root takes an average of 262 ms with a standard deviation of $\sigma = 9$ ms. The response takes 1804 ms with a σ of 177 ms. The shorter time for the message to the root can be explained by the smaller message size of 34 bytes of the upward message. Furthermore, the 82 byte response is fragmented into 4 packets

[2] Based on exemplary time measurements of a single node sending DIO messages.
[3] Note that a slide deviation is caused by the manual time synchronization of the boards.



(a) Absolute Delays



(b) Histogram: round-trip time

Figure 6.2: TRAIL Round-trip Time over One Hop

which results in additional overhead. The fragmentation is a result of the buffer size of the transceiver of the MSB-A2 board.

The round trip message takes an average of 2088 ms with $\sigma = 176$ ms. An overall of 9 messages where lost, so that the message loss rate for this experiment is 3 %. Message loss in Figure 6.2(a) is represented by a disruption in the line as seen at message number 3 in the line of triangles and squares. Figure 6.2(b) shows the frequency distribution of the round-trip times. It can be seen that the distribution is approximately a Gaussian-shape around the mean value of 2088 ms.

Characteristic peaks in Figure 6.2(a), for instance at message numbers 9 or 37, are due to the periodic sending and receiving of DIO or link layer messages. The noticeable sawtooth pattern within the upper two curves (triangles and squares) suggests a characteristic behavior of the 6LoWPAN stack of μkleos. This may be the result of buffering or possibly a periodic event. Although denoting an opportunity for optimizing the transmission time, a detailed evaluation of this pattern is not within scope of this work.

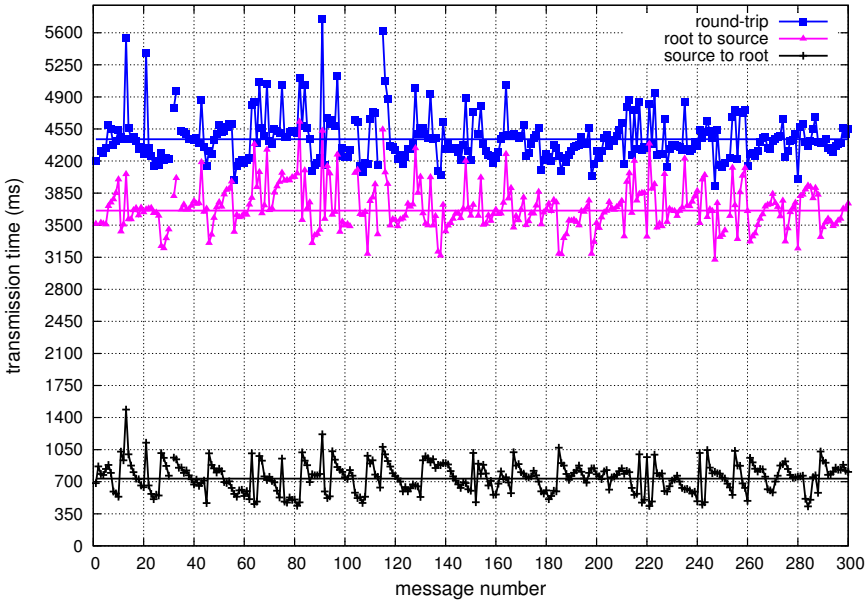
Further delays are caused by beaconing messages. The nodes send regular beaconing messages by which the signal strength of other nodes is determined and updated and thus occupy the channel. By Carrier Sense Multiple Access (CSMA), a node senses whether the channel is occupied before transmitting and delays sending a message until the channel is free. Furthermore, MAC frames in μkleos are acknowledged which results in additional delays.[4]

6.3.3 Experiment 2: Two Hops

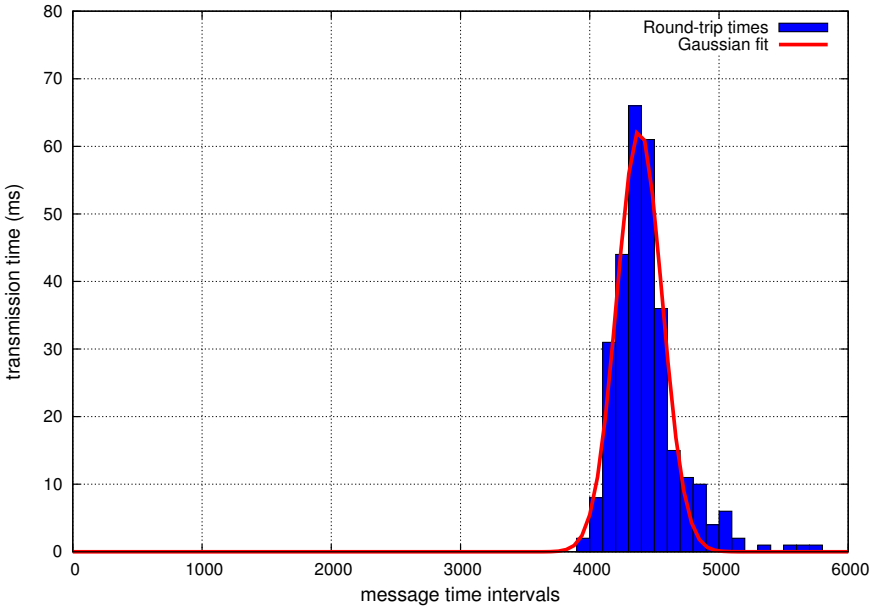
The second experiment uses three nodes: a root, an intermediate node and a source node. The source periodically sends an attestation message every 8 seconds. The interval is chosen larger than the interval in the first experiment since the message travels two hops and thus takes more time to return to the source. The absolute transmission times are illustrated in Figure 6.3(a). The message to the root now takes an average of 734 ms with $\sigma = 160$ ms. The downward path of the message including the signature-dummy takes 3659 ms with $\sigma = 203$ ms. The average round-trip time over two hops is 4438 ms with $\sigma = 174$ ms. As expected, this is a factor of two when comparing to the one-hop round-trip time. In this experiment only 8 messages were lost, so that message the loss rate is 2.6 %.

In Figure 6.3(a) a characteristic sawtooth pattern is visible as well. However, it shows more irregularities than in Figure 6.2(a), so that the distribution seen in Figure 6.3(b) is closer to a normal (Gaussian) distribution.

[4] As mentioned by a developer of RIOT OS / μkleos, 2013-08-08.



(a) Absolute Delays



(b) Histogram: round-trip time

Figure 6.3: TRAIL Round-trip Time over Two Hops

6.4 Discussion

This chapter gave a first impression of the delays for rank validations when applying TRAIL. In two experiments the delays of the attestation message have been measured. It was shown that the delay can be estimated to 2 seconds per hop for each rank validation when deploying the single path rank validation in μkleos. Considering the topology creation in μkleos the joining process of a node is delayed by around 3 seconds. However, it is expected that this outcome has some optimization potential due to μkleos specific behavior on lower layers. Therefore an implementation of TRAIL in the latest version of RIOT OS could already show shorter delays. Alternatively, different studies that involve other hardware platforms with larger buffers can also provide lower delays due to less fragmentation for the applied packet sizes.

Consequently, this evaluation shows that TRAIL is well suitable for static networks where rank changes occur infrequently and that tolerate this delay. The prototype demonstrates that TRAIL can be implemented with relatively low effort and is easily included into an existing RPL implementation. The delay in the range of seconds per hop make it applicable in scenarios that are not highly dependent on low delays during bootstrapping and in networks that do not comprise of a great number of rank levels.

7 Conclusions & Outlook

This work concerned with the aspect of secure routing in low-power and lossy networks. The focus was drawn to the case RPL, a newly standardized routing protocol. RPL is primarily designed for convergecast in which a sink node collects the data from many sensor nodes. While RPL is aiming at becoming the standard solution for various deployment scenarios such as urban, industrial and home environments, each of these scenarios has special security requirements. RPL therefore provides a basic security framework to protect against potential threats and attacks, and requires external specifications for additional features such as an automated key management.

RPL employs cryptographic protocols that protect against most attacks launched by an outsider attacker. However, once an attacker has access to cryptographic keys he may launch several topology attacks, such as rank spoofing and version number attacks. Rank spoofing allows the adversary to create a sinkhole by propagating a false rank. In a version number attack he initiates a global repair and thus disturbs the network. Two approaches that anticipate these attacks were analyzed: VeRA and TRAIL.

VeRA authenticates ranks and the version number by use of one-way hash chains. While VeRA hereby protects against version number attacks, it shows two vulnerabilities that enable an attacker to circumvent the applied rank verifications. These vulnerabilities allow an attacker to decrease his rank to an arbitrary value or replay the rank of his parent. To counter these attacks, two defense techniques have been proposed: An encryption chain that prevents rank spoofing and a challenge-response procedure to detect rank replay attacks.

TRAIL inverts the validation process, so that children independently verify the rank of their parents by sending a validation message to the root. The root functions as trust anchor and thus acknowledges the rank validations of child nodes. TRAIL detects rank spoofing and rank replay attacks and requires each node to send only two messages. A disadvantage of TRAIL is the message size that increases with the number of nodes.

A practical evaluation for the operating system μkleos gave a first impression on the delays that result from the rank validation of single nodes in TRAIL. The delays for rank validation

postpone the parent selection process for about 2 seconds per hop, and show that TRAIL is well applicable in static networks where rank changes occur infrequently.

Future research will have to focus on reducing the message size of TRAIL to make it applicable on a larger scale. Furthermore, the work on TRAIL should be continued with the aim of extending the concept to other routing protocols. Regarding the security of TRAIL and VeRA, further attention is required to defend multiple collaborating attackers with respect to rank replay attacks.

It would be worthwhile to extend the TRAIL prototype with the required features for the scalable rank verification with Bloom filters to investigate its feasibility in real life scenarios. The implementation should be integrated into RIOT OS and thus take advantage of new features and recent developments. Such an integration is very promising in terms of optimization possibilities. Further evaluations may also consider the energy consumption of TRAIL and analyze the overall delay for large-scale networks. The DES-Testbed at the FU Berlin may provide a good opportunity to perform these studies.

Bibliography

[1] Stephen E. Deering and Robert M. Hinden. Internet Protocol, Version 6 (IPv6) Specification. RFC 2460, IETF, December 1998.

[2] Arsalan Tavakoli, Stephen Dawson-Haggerty, and P Levis. Overview of Existing Routing Protocols for Low Power and Lossy Networks. Internet-Draft – work in progress 07, IETF, April 2009.

[3] C. Perkins, E. Belding-Royer, and S. Das. Ad hoc On-Demand Distance Vector (AODV) Routing. RFC 3561, IETF, July 2003.

[4] T. Clausen and P. Jacquet. Optimized Link State Routing Protocol (OLSR). RFC 3626, IETF, October 2003.

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

[6] 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)*, pages 709–714, Oct. 2011.

[7] Amit Dvir, Laszlo Dora, Levente Buttyan, and Tamas Holczer. Version Number and Rank Authentication. Internet-Draft – work in progress 01, IETF, January 2012.

[8] Martin Landsmann, Heiner Perrey, Osman Ugus, Matthias Wählisch, and Thomas C. Schmidt. Topology Authentication in RPL. In *Proc. of the 32nd IEEE INFOCOM. Poster*, Piscataway, NJ, USA, April 2013. IEEE Press.

[9] Heiner Perrey, Martin Landsmann, Osman Ugus, Matthias Wählisch, and Thomas C. Schmidt. TRAIL: Topology Authentication in RPL. 2013. Submitted to 33rd IEEE INFOCOM.

[10] R. Shirey. Internet Security Glossary, Version 2. RFC 4949, IETF, August 2007.

[11] Bruce Schneier. *Applied Cryptography*. Wiley & Sons, Hoboken, NJ., 2nd edition, 1995.

[12] Leslie Lamport. Password Authentication with Insecure Communication. *Communications of the ACM*, 24(11):770–772, November 1981.

[13] D. Whiting, R. Housley, and N. Ferguson. Counter with CBC-MAC (CCM). RFC 3610, IETF, September 2003.

[14] Hugo Krawczyk, Mihir Bellare, and Ran Canetti. HMAC: Keyed-Hashing for Message Authentication. RFC 2104, IETF, February 1997.

[15] R. L. Rivest, A. Shamir, and L. Adleman. A Method for Obtaining Digital Signatures and Public-key Cryptosystems. *Commun. ACM*, 21(2):120–126, Feb. 1978.

[16] S. Bellovin and R. Housley. Guidelines for Cryptographic Key Management. RFC 4107, IETF, June 2005.

[17] Wade Trappe and Lawrence C. Washington. *Introduction to Cryptography with Coding Theory*. Pearson Education, Inc., 1st edition, 2002.

[18] W. Diffie and M. Hellman. New Directions in Cryptography. *IEEE Trans. Inf. Theor.*, 22(6):644–654, Nov. 1976.

[19] Burton H. Bloom. Space/Time Trade-offs in Hash Coding with Allowable Errors. *Commun. ACM*, 13(7):422–426, July 1970.

[20] Andrei Broder and Michael Mitzenmacher. Network Applications of Bloom filters: A Survey. *Internet Mathematics*, 1(4):485–509, 2004.

[21] Michael Mitzenmacher. Compressed Bloom Filters. *IEEE/ACM Trans. Netw.*, 10(5):604–612, October 2002.

[22] Paulo Sérgio Almeida, Carlos Baquero, Nuno Preguiça, and David Hutchison. Scalable Bloom Filters. *Inf. Process. Lett.*, 101(6):255–261, March 2007.

[23] M. Dohler, T. Watteyne, T. Winter, and D. Barthel. Routing Requirements for Urban Low-Power and Lossy Networks. RFC 5548, IETF, May 2009.

[24] K. Pister, P. Thubert, S. Dwars, and T. Phinney. Industrial Routing Requirements in Low-Power and Lossy Networks. RFC 5673, IETF, October 2009.

[25] A. Brandt, J. Buron, and G. Porcu. Home Automation Routing Requirements in Low-Power and Lossy Networks. RFC 5826, IETF, April 2010.

[26] J. Martocci, P. De Mil, N. Riou, and W. Vermeylen. Building Automation Routing Requirements in Low-Power and Lossy Networks. RFC 5867, IETF, June 2010.

[27] A. Conta, S. Deering, and M. Gupta. Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification. RFC 4443, IETF, March 2006.

[28] Veronika Bauer. Routing in Wireless Sensor Networks: An Experimental Evaluation of RPL. Masterthesis, École Polytechnique, Paris, France, December 2010.

[29] JP. Vasseur, M. Kim, K. Pister, N. Dejean, and D. Barthel. Routing Metrics Used for Path Calculation in Low-Power and Lossy Networks. RFC 6551, IETF, March 2012.

[30] P. Levis, T. Clausen, J. Hui, O. Gnawali, and J. Ko. The Trickle Algorithm. RFC 6206, IETF, March 2011.

[31] Philip Levis, Neil Patel, David Culler, and Scott Shenker. Trickle: A Self-Regulating Algorithm for Code Propagation and Maintenance in Wireless Sensor Networks. In *Proc. of the 1st Symposium on Networked Systems Design and Implementation*, pages 15–28, San Francisco, USA, March 2004.

[32] J. Hui and JP. Vasseur. The Routing Protocol for Low-Power and Lossy Networks (RPL) Option for Carrying RPL Information in Data-Plane Datagrams. RFC 6553, IETF, March 2012.

[33] Mukul Goyal, Jerry Martocci, Yusuf Bashir, Emmanuel Baccelli, and Ted Humpal. A Performance Analysis of P2P Routing along a DAG in LLNs. Internet-Draft – work in progress 00, IETF, March 2010.

[34] M. Goyal, E. Baccelli, M. Philipp, A. Brandt, and J. Martocci. Reactive Discovery of Point-to-Point Routes in Low-Power and Lossy Networks. RFC 6997, IETF, August 2013.

[35] Jonathan Hui and Richard Kelsey. Multicast Protocol for Low power and Lossy Networks (MPL). Internet-Draft – work in progress 04, IETF, February 2013.

[36] J. Jonsson and B. Kaliski. Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1. RFC 3447, IETF, February 2003.

[37] D. Eastlake and T. Hansen. US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF). RFC 6234, IETF, May 2011.

[38] Tzeta Tsao, Roger Alexander, Mischa Dohler, Vanesa Daza, and Angel Lozano. A Security Threat Analysis for Routing over Low-Power and Lossy Networks. Internet-Draft – work in progress 03, IETF, June 2013.

[39] Osman Ugus. *Secure and Reliable Remote Programming in Wireless Sensor Networks*. PhD thesis, FernUniversität Hagen, 2012.

[40] Chris Karlof and David Wagner. Secure Routing in Wireless Sensor Networks: Attacks and Countermeasures. *Elsevier Ad Hoc Networks*, 1(2–3):293–315, 2003.

[41] Roger Alexander and Tzeta Tsao. Adapted Multimedia Internet KEYing (AMIKEY): An extension of Multimedia Internet KEYing (MIKEY) Methods for Generic LLN Environments. Internet-Draft – work in progress 04, IETF, September 2012.

[42] J. Arkko, E. Carrara, F. Lindholm, M. Naslund, and K. Norrman. MIKEY: Multimedia Internet KEYing. RFC 3830, IETF, August 2004.

[43] Jakob Jonsson. On the Security of CTR + CBC-MAC. In *Selected Areas in Cryptography*, pages 76–93. Springer, 2002.

[44] Alex Biryukov, Orr Dunkelman, Nathan Keller, Dmitry Khovratovich, and Adi Shamir. Key Recovery Attacks of Practical Complexity on AES Variants With Up To 10 Rounds. Cryptology ePrint Archive, Report 2009/374, 2009. `http://eprint.iacr.org/`.

[45] Andrey Bogdanov, Dmitry Khovratovich, and Christian Rechberger. Biclique Cryptanalysis of the Full AES. In *Proc. of 17th ASIACRYPT*, pages 344–371, December 2011.

[46] Mihir Bellare and Phillip Rogaway. The Exact Security of Digital Signatures–How to Sign with RSA and Rabin. In *Proceedings of the 15th EUROCRYPT*, EUROCRYPT'96, pages 399–416, Berlin, Heidelberg, 1996. Springer-Verlag.

[47] Ronald L. Rivest and Burton S. Kaliski Jr. RSA Problem. In *Encyclopedia of Cryptography and Security*. Springer-Verlag, 2005.

[48] Thorsten Kleinjung, Kazumaro Aoki, Jens Franke, Arjen K. Lenstra, Emmanuel Thomé, Joppe W. Bos, Pierrick Gaudry, Alexander Kruppa, Peter L. Montgomery, Dag Arne Osvik, Herman Te Riele, Andrey Timofeev, and Paul Zimmermann. Factorization of a 768-bit

RSA modulus. In *Proceedings of the 30th annual conference on Advances in cryptology*, CRYPTO'10, pages 333–350, Berlin, Heidelberg, 2010. Springer-Verlag.

[49] Krzysztof Piotrowski, Peter Langendoerfer, and Steffen Peter. How Public Key Cryptography Influences Wireless Sensor Node Lifetime. In *Proceedings of the Fourth ACM Workshop on Security of Ad Hoc and Sensor Networks*, SASN '06, pages 169–176, Alexandria, Virginia, USA, 2006. ACM.

[50] A.S. Wander, N. Gura, H. Eberle, V. Gupta, and S.C. Shantz. Energy Analysis of Public-Key Cryptography for Wireless Sensor Networks. In *Third IEEE International Conference on Pervasive Computing and Communications*, pages 324–328, March 2005.

[51] N.R. Potlapally, S. Ravi, A. Raghunathan, and N.K. Jha. Analyzing the Energy Consumption of Security Protocols. In *Proceedings of the International Symposium on Low Power Electronics and Design*, pages 30–35, 2003.

[52] Victor S Miller. Use of Elliptic Curves in Cryptography. In *Lecture Notes in Computer Sciences – Advances in Cryptology*, pages 417–426, New York, NY, USA, 1986. Springer.

[53] Neal Koblitz. Elliptic Curve Cryptosystems. *Mathematics of Computation*, 48(177):203–209, January 1987.

[54] T. Aura. Cryptographically Generated Addresses (CGA). RFC 3972, IETF, March 2005.

[55] Behcet Sarikaya, Frank Xia, and Greg Zaverucha. Lightweight Secure Neighbor Discovery for Low-power and Lossy Networks. Internet-Draft – work in progress 03, IETF, April 2012.

[56] John R. Douceur. The Sybil Attack. In *Revised Papers from the First International Workshop on Peer-to-Peer Systems*, pages 251–260, London, UK, 2002. Springer-Verlag.

[57] A. Le, J. Loo, Yuan Luo, and A. Lasebae. Specification-based IDS for securing RPL from topology attacks. In *IFIP Wireless Days (WD)*, pages 1–3, October 2011.

[58] Linus Wallgren, Shahid Raza, and Thiemo Voigt. Routing Attacks and Countermeasures in the RPL-based Internet of Things. *International Journal of Distributed Sensor Networks*, 2013. to be published – `downloads.hindawi.com/journals/ijdsn/aip/794326.pdf`.

[59] Wenyuan Xu, Ke Ma, Wade Trappe, and Yanyong Zhang. Jamming Sensor Networks: Attack and Defense Strategies. *Network, IEEE*, 20(3):41–47, may-june 2006.

[60] Anthony D. Wood and John A. Stankovic. Denial of Service in Sensor Networks. *Computer*, 35(10):54–62, October 2002.

[61] Kevin Weekly and Kristofer Pister. Evaluating Sinkhole Defense Techniques in RPL Networks. In *Network Protocols (ICNP), 2012 20th IEEE International Conference on*, pages 1–6, November 2012.

[62] N. Kushalnagar, G. Montenegro, and C. Schumacher. IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs): Overview, Assumptions, Problem Statement, and Goals. RFC 4919, IETF, August 2007.

[63] IEEE Computer Society. IEEE Std. 802.15.4™-2003 – Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (LR-WPANs), Oct. 2003.

[64] A. Dunkels, B. Gronvall, and T. Voigt. Contiki - A Lightweight and Flexible Operating System for Tiny Networked Sensors. In *29th IEEE Local Computer Networks*, pages 455–462, Tampa, FL, USA, November 2004.

[65] Oliver Hahm, Emmanuel Baccelli, Mesut Günes, Matthias Wählisch, and Thomas C. Schmidt. RIOT OS: Towards an OS for the Internet of Things. In *Proc. of the 32nd IEEE INFOCOM. Poster*, Piscataway, NJ, USA, 2013. IEEE Press.

[66] Emmanuel Baccelli, Oliver Hahm, Matthias Wählisch, Mesut Günes, and Thomas C. Schmidt. RIOT: One OS to Rule Them All in the IoT. Research Report RR–8176, INRIA, Dec. 2012.

[67] M. Baar, H. Will, B. Blywis, T. Hillebrandt, A. Liers, G. Wittenburg, and J. Schiller. The ScatterWeb MSB-A2 Platform for Wireless Sensor Networks. Technical report, Freie Universität Berlin, September 2008. [Online]. Available: `ftp://ftp.inf.fu-berlin.de/pub/reports/tr-b-08-15.pdf`.

[68] Chipcon / Texas Instruments. CC1100 – Low-Power Sub-1 GHz RF Transceiver, May 2009. `http://www.ti.com/general/docs/lit/getliterature.tsp?literatureNumber=swrs038d&fileType=pdf`.

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

Hamburg, August 20, 2013 Heiner Perrey