D5.53 Mobility and Security Support

The CALIPSO Consortium

TCF, Thales Communications, France
CNRS, Centre National de la Recherche Scientifique, France
SICS, Swedish Institute of Computer Science, Sweden
UPA, University of Parma, Italy
DRZ, Disney Research Zurich, Switzerland
WOS, Worldsensing, Spain
CISCO, Cisco Systems International B.V., Netherlands

©Copyright 2013, the Members of the CALIPSO Consortium
### Document Information

<table>
<thead>
<tr>
<th>Contract Number</th>
<th>288879</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deliverable Name</td>
<td>Mobility and Security Support</td>
</tr>
<tr>
<td>Deliverable number</td>
<td>D5.53</td>
</tr>
</tbody>
</table>
| Editor(s)             | Vladimir Vukadinovic (DRZ)  
                        | Gianluigi Ferrari (UPA) |
| Author(s)             | Jérémie Leguay (TCS)  
                        | Simone Cirani (UPA)  
                        | Pietro Gonizzi (UPA)  
                        | Shahid Raza (SICS)  
                        | Simon Duquennoy (SICS)  
                        | Jun-Young Bae (CNRS)  
                        | Franck Rousseau (CNRS) |
| Reviewer(s)           | Paolo Medagliani (TCS) |
| Dissemination level   | Public |
| Contractual date of delivery | 9/2013 |
| Delivery date         | 9/2013 |
| Status                | Final |
| Keywords              | CALIPSO, security, IPsec, intrusion detection, key distribution |

This project is funded under 7th Framework Program.
Contents

Document Information .................................................. 1

1 Introduction ......................................................... 4

2 Overview, relation to the CALIPSO architecture and application scenarios ........ 4

2.1 Overview of technical contributions ........................................ 4

2.2 Relation to the CALIPSO architecture ....................................... 5

2.3 Relation to the CALIPSO application scenarios .............................. 5

3 Technical Contribution ................................................ 7

3.1 Analysis of network, transport, and application layer security mechanisms for 6LoWPAN .......................................................... 7

3.1.1 Network-Layer Security ...................................................... 8

3.1.2 Transport-Layer Security ................................................. 9

3.1.3 Application-Layer Security ............................................... 10

3.2 Routing Attacks and Countermeasures in the RPL-based Networks ............ 12

3.2.1 IoT Technologies and IDS ................................................. 13

3.2.2 Attacks against RPL .......................................................... 17

3.2.3 IDS and the IoT .............................................................. 23

3.2.4 Conclusions ................................................................. 25

3.3 Real-time Intrusion Detection in the Internet of Things .......................... 26

3.3.1 SVELTE: An IDS for the IoT .............................................. 27

3.3.2 Implementation .............................................................. 35

3.3.3 Evaluation ................................................................. 35

3.3.4 Related Work ............................................................... 40

3.3.5 SVELTE Extensions ....................................................... 41

3.3.6 Conclusions ................................................................. 41

3.4 Compression of IPsec AH and ESP Headers for Constrained Environments [draft-raza-6lowpan-ipsec-01] ...................................................... 42

3.4.1 Introduction ................................................................. 42

3.4.2 Linking IPsec Headers Compression with 6LoWPAN .................... 43

3.4.3 LOWPAN_NHC for Authentication Header ................................ 43

3.4.4 LOWPAN_NHC for Encapsulated Security Payload (ESP) ........... 44

3.4.5 Implementation Considerations .......................................... 45

3.4.6 Security Considerations .................................................. 46

3.4.7 IANA Considerations ..................................................... 46

3.5 Secure distribution of shared keys ........................................... 47

3.5.1 Related Works ............................................................. 48

3.5.2 New Group Key Management Protocol .................................. 50

3.5.3 Comparison of Different Key Distribution Strategies .................... 55

3.6 Distributed key verification and management ................................ 58

3.6.1 Assumptions ............................................................... 58

3.6.2 Accumulator-based Protocol .............................................. 59

3.6.3 Security Analysis .......................................................... 62

3.6.4 Performance Evaluation .................................................. 63

3.6.5 Related Work .............................................................. 65
3.6.6 Conclusion and Future Work

4 Conclusions
1. Introduction

This deliverable presents the CALIPSO contributions in Task 5.3, “Mobility and security support”. We first give a high-level overview of the technical contributions and relate them to the CALIPSO architecture and application scenarios. Then, we present each technical contribution in details.

2. Overview, relation to the CALIPSO architecture and application scenarios

Deliverable D5.53, “Mobility and security support”, proposes IP security mechanisms suitable for resource-constrained smart objects. Mechanisms described in this deliverable complement those described in D5.52, “Secure and large-scale service discovery”, to provide a comprehensive set of tools to enable authentication, confidentiality, availability, and integrity of data in a network of smart objects. Regardless of the title, this deliverable does not include mechanisms for mobility support. Some work on mobility support has already been carried out and it is reported in D4.42, while some is still on-going and it will be reported in later deliverables.

2.1. Overview of technical contributions

In Section 3.1, we provide an overview of network, transport, and application layer security protocols that are the most commonly used for securing IP-based end-to-end communications between smart objects.

In Section 3.2, we study how RPL routing protocol behaves in the presence of some of the most common routing attacks in 6LoWPAN networks. We investigate how security features in IPv6 can be used for intrusion detection or exploited by the attackers. This work has been published in the International Journal of Distributed Sensor Networks [1].

In Section 3.3, we design, implement, and evaluate SVELTE, a novel Intrusion Detection system specifically designed for the IoT. It primarily targets routing attacks, but it is extensible and can be used to detect other attacks. This work has been published in the Ad Hoc Networks journal [2].

In Section 3.4, we describes a header compression mechanisms for the IPsec, which is compliant with the 6LoWPAN header encoding scheme standardized in RFC 6282. The work has been published as an IETF Internet Draft [3].

In Section 3.5, we present a group key distribution protocol for secure distribution of symmetric cryptographic keys. The protocol is tailored for very dynamic ad-hoc networks, either wired or wireless. It provides proper mechanisms to deal with unpredictable leave events and to resist against collusive attacks. This work has been published in the Ad Hoc Networks journal [4].

In Section 3.6, we propose a distributed key verification protocol that does not require certification authorities and certificates. Under the protocol, nodes autonomously verify the authenticity of public keys they exchange using one-way accumulators. This work has been submitted for publication.
2.2. Relation to the CALIPSO architecture

Figure 1 shows the relation between each contribution and the corresponding components of the CALIPSO functional architecture. At the Link layer, we propose an intrusion detection system (IDS) that ensures that only legitimate nodes can be members of the network. At the Network layer, the same IDS is used to prevent routing attacks by malicious nodes. The IDS relies on the existing end-to-end security solutions, such as IPsec and DTLS. We propose a new header compression scheme for IPsec to meet the resource constraints of smart objects. Our mechanisms for secure key distribution, verification, and maintenance span the entire Security plane of the CALIPSO architecture.

2.3. Relation to the CALIPSO application scenarios

Table 1 presents the relation between modules described in this deliverable and the respective CALIPSO application scenarios. The table is given for informational purposes only and it does not yet reflect exactly what the field trials will contain and evaluate. Rather, we expect the field trials to focus on a selected subset of these modules.

For the Smart Toy scenarios, the proposed key distribution and verification mechanisms are of particular value. Group key distribution facilitates the process of supplying keys to toys that belong to the same product line or subscribed to the same service. The distributed
<table>
<thead>
<tr>
<th>Module</th>
<th>Scenario</th>
<th>Justification</th>
</tr>
</thead>
<tbody>
<tr>
<td>IPsec header compression</td>
<td>Smart Parking</td>
<td>The Smart Parking scenario requires payload and header encryption, for which compressed IPsec+6LoWPAN is an appealing option.</td>
</tr>
<tr>
<td></td>
<td>Critical Infrastructures</td>
<td>The Critical Infrastructures scenario requires secure authentication and payload encryption, which can be provided by IPsec.</td>
</tr>
<tr>
<td></td>
<td>Smart Toys</td>
<td>The Smart Toys scenario authentication, for which IPsec (in AH mode) is an interesting solution.</td>
</tr>
<tr>
<td>Real-time intrusion detection</td>
<td>Smart Parking</td>
<td>The Smart Parking scenario relies on RPL, and therefore needs mechanisms to avoid illegitimate nodes to participate in routing.</td>
</tr>
<tr>
<td></td>
<td>Critical Infrastructures</td>
<td>The Critical Infrastructures scenario relies on RPL, and therefore needs mechanisms to avoid illegitimate nodes to participate in routing.</td>
</tr>
<tr>
<td>Group key distribution</td>
<td>Critical Infrastructures, Smart Parking, Smart Toys</td>
<td>All scenarios may benefit from the proposed mechanisms.</td>
</tr>
<tr>
<td>Distributed key verification</td>
<td>Critical Infrastructures, Smart Parking, Smart Toys</td>
<td>All scenarios may benefit from the proposed mechanisms.</td>
</tr>
</tbody>
</table>

Table 1: Relation between modules and application scenarios

key verification mechanism eliminates the need for certificates and certification authorities and therefore streamlines the process of public key verification, which is important for devices and networks whose end-users might not be tech-savvy.

For the Critical Infrastructures and Smart Parking scenarios, which have more strict (technical) security requirements, all proposed mechanisms can be used to build a complete security solution. IPsec header compression and intrusion detection mechanisms provide, respectively, end-to-end security and protect RPL against routing attacks, which are common requirements in these two scenarios. The key distribution and verification mechanisms facilitate the process of installing cryptographic material on a large number of sensor nodes that might not be physically accessible.
3. Technical Contribution

3.1. Analysis of network, transport, and application layer security mechanisms for 6LoWPAN

One of the most important requirements and crucial aspects for a correct deployment and diffusion of IoT is security. Several challenging security goals should be achieved, including data confidentiality, data authentication, integrity, service availability, peer entity authentication, authorization, anonymity, and/or pseudonymity. Since the protocol architecture of smart objects should adhere to the standard IP architecture (for obvious integration reasons), many of the security mechanisms already defined and currently used for the Internet can be reused in IoT scenarios. Moreover, since many of the Internet security protocols have been defined taking into account the possibility to select and properly configure the used security algorithms and other cryptographic primitives, such Internet security protocols can still be reused, possibly with proper algorithmic or configuration modifications.

In this subsection, the main protocols that can be used for securing IP-based end-to-end communications between smart objects are recalled, and the main issues related to this type of communications are discussed. A direct comparison between possible layered architectures of security protocols in Internet and IoT scenarios is shown in Figure 2.

![Figure 2: Comparison between the Internet and the IoT security protocols.](image)

It is important to observe that the IoT protocol suite depicted in Figure 2 represents just the possible choices to enforce data protection (at different layers) by a smart object, rather than the actual set of security mechanisms effectively implemented and simultaneously used at different layers. At the opposite, in order to minimize the used resources, particular attention has to be devoted to avoid the repetition of the same functionalities at different layers, if not strictly required.

Referring to the IoT protocol stack of Figure 2 at the application layer there is the CoAP application protocol that can be used to interact in a request/response manner between smart objects or between a smart object and a non-constrained (standard) Internet node (possibly by using some intermediate relay/proxy node). CoAP itself does not provide primitives for authentication and data protection, so these functions should be implemented directly at the application/service layer (by directly protecting the data encapsulated and exchanged by CoAP) or at one of the underlying layers. Although data authentication, integrity, and confidentiality can be provided at lower layers, such as PHY or MAC (e.g., in IEEE 802.15.4 systems), no end-
to-end security can be guaranteed without a high level of trust on intermediate nodes. However, due to the highly dynamic nature of wireless multi-hop communications expected to be used to form the routing path between remote end nodes, this kind of security (hop-by-hop) is not, in general, sufficient. For such reason, security mechanisms at network, transport, or application level should be considered instead of (or in addition to) PHY and MAC level mechanisms.

### 3.1.1. Network-Layer Security

At network layer, an IoT node can secure data exchange in a standard way by using the Internet Protocol Security (IPsec) [5]. IPSec was originally developed for IPv6, but found widespread deployment, first, as an extension in IPv4, into which it was back-engineered. IPSec was an integral part of the base IPv6 protocol suite, but has since then been made optional. IPSec can be used in protecting data flows between a pair of hosts (host-to-host communication), between a pair of security gateways (network-to-network communication), or between a security gateway and a host (network-to-host communication).

IPSec can provide confidentiality, integrity, data-origin authentication and protection against replay attacks, for each IP packet (it works at network layer). Such security services are implemented by two IPSec security protocols: Authentication Header (AH) and Encapsulated Security Payload (ESP). While the former (AH) provides integrity, data-origin authentication, and optionally anti-replay capabilities, the latter (ESP) may provide confidentiality, data-origin authentication, integrity, and anti-replay capabilities.

IPSec AH and ESP define only the way payload data (in clear or enciphered) and IPSec control information are encapsulated, while the effective algorithms for data origin authentication/integrity/confidentiality can be specified separately and selected amongst a set of available cipher suites.

This modularity makes IPSec usable also in the presence of very resource-constrained devices, if a proper algorithm that guarantees both usability and sufficient security level is selected. This means that, from an algorithmic point of view, the problem moves from the IPSec protocol itself to the actual cryptographic algorithms.

The keying material and the selected cryptographic algorithms used by IPSec for securing a communication are called IPSec Security Association (SA). To establish a SA, IPSec can be pre-configured (specifying a pre-shared key, hash function and encryption algorithm) or can be dynamically negotiated by the IPSec Internet Key Exchange (IKE) protocol. Unfortunately, as the IKE protocol was designed for standard Internet nodes, it uses asymmetric cryptography, which is computationally heavy for very small devices. For this reason, proper IKE extensions should be considered using lighter algorithms.

Other problems related to the implementation of IPSec in constrained IoT nodes include data overhead (with respect to IP), configuration, and practical implementation aspects. Data overhead is introduced by the extra header encapsulation of IPSec AH and/or ESP. However, this can be limited by implementing header compression techniques, similarly to what is done in 6LoWPAN for the IP header. In [6], a possible compression mechanism for IPSec in 6LoWPAN is proposed and numerically evaluated.

Regarding practical aspects, it is worth to observe that IPSec is often designed for VPNs, thus making it difficult for them to be dynamically configurable by an application. Moreover, existing implementations are also hardly compatible with each other and often require manual configuration to interoperate.

An alternative to using IKE+IPsec is the Host Identity Protocol (HIP) [7]. The main objective of HIP is to decouple the two functions of host locators (for routing purposes) and host
identifiers (for actual host identification) currently performed by IP addresses. For this purpose, HIP introduces a new namespace between IP and upper layers specific for host identification based on public cryptography. In HIP, the host identity (HI) is directly associated with a pair of public/private keys, where the private key is owned by the host and the public key is used as Host Identifier (HI). HIP defines also an Host Identity Tag (HIT), a 128-bit representation of the HI based on the hash of HI plus other information, which can be used for example as unique host identifier at the existing IPv6 API and by application protocols. HIP defines also an HIP exchange that can be used between IP hosts to establish a HIP security association that in turn can be used to start secure host-to-host communications based on the (IPSec) ESP protocol [8].

In addition to security, HIP provides methods for IP multihoming and host mobility that are important features for an IP-based IoT network architecture. Some works are also being carried on to let the HIP exchange run on very constrained devices, by using proper public-key cryptographic primitives.

3.1.2. Transport-Layer Security

In the current IP architecture, data exchange between application nodes can be secured at transport layer through the standard Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS) protocols. TLS is the widest used secure protocol, running on top of the TCP providing to the application layer the same connection and stream-oriented interface of TCP [9]. In addition, TLS provides complete secure communication through: peer-entity authentication and key exchange (using asymmetric cryptography); data authentication, integrity, and anti-replay (through message authentication code); confidentiality (using symmetric encryption). Peer-entity authentication and key exchange is provided by the TLS Handshake phase, performed at the beginning of the communication.

DTLS, instead, has been introduced more recently in order to provide a security service similar to TLS on top of UDP [10]. Although it is still poorly supported in standard Internet nodes, it is currently the reference security protocol for IoT systems since it uses UDP as transport and does not suffer from the problems originated by the use of TCP in network-constrained scenarios (due to the extremely variable transmission delay and lossy links).

Both IPSec and DTLS provide the same security features with their own mechanisms at different stack layers. Moreover the IPSec IKE key agreement reflects almost the same DTLS Handshake function. The main advantage of securing communications at transport layer with DTLS consists in allowing more precise access control. In fact, operation at the transport layer allows applications to directly and easily select which, if any, security service has to be set up. Another practical advantage is that the adoption of DTLS can allow for the reuse of the large experience and implementations that came with TLS.

For these reasons DTLS has recently received significant attention for securing communication of constrained node/network applications and it has been standardized as the security protocol for CoAP associated to “coaps” URIs [11].

Unfortunately, there are still some few issues that should be faced in order to make DTLS more friendly for constrained devices. The most relevant ones are related to limited packet size imposed by underlying protocols such as IEEE 802.15.4. In fact, as for IPSec, DTLS introduces overhead during both handshake and data transport phases. DTLS offers fragmentation at the Handshake layer, however, this can add a significant overhead. Another solution could be to use the fragmentation offered at IPv6 or 6LoWPAN layer. Moreover, in order to reduce DTLS overhead, some packet optimization and compression mechanism can be introduced. For
example, in [12] the authors propose to use the 6LoWPAN compression mechanisms for the DTLS protocol.

From the security point of view, one problem of using DTLS or IPSec is that end-to-end communication is not guaranteed when intermediate nodes such as proxies or application level gateways are introduced. In fact, both IPSec and DTLS provide secure communications at IP and transport layers respectively, and, in presence of a multi-hop application-level communications, they can assure security only within each hop. In addition, some complications in providing end-to-end security may arise also when connectivity is realized directly at IP and transport layers. There are scenarios in which a part of the network (internal) composed by constrained devices is interconnected at IP level to the rest of the (external) network, for example the Internet. Although data protection can be guaranteed through IPSec or DTLS protocols, other network attacks, like flooding or replay, may occur due to the asymmetry of the resources available at the end systems; for example a full-powered host attached to the Internet may attack a constrained device by trying to consume all power or processing resources of the limited device. In order to guarantee a proper level of protection also against this kind of attacks, an intermediate security gateway may be required at the border of the internal network. A security gateway may act as access controller, granting access to the internal network only to trusted nodes. In [13, 14], the authors specifically face this issue and try to propose a solution. In particular, in case of end-to-end application level communication based on CoAP, a solution may be to require the external node to encapsulate CoAP/DTLS/IP traffic within a proper DTLS tunnel established between the external node and the security gateway.

It is also important to note that, although DTLS provides a datagram-oriented communication service (like UDP), it establishes a point-to-point secure association that is not compatible with multicast communications (in contrast with UDP, which does support multicast). In order to make DTLS applicable in multicast IP-communication scenarios, some protocol extensions for group-key management should be introduced in the future.

### 3.1.3. Application-Layer Security

Providing security at IP layer (through IPSec) or transport layer (through TLS or DTLS) has several advantages. The main ones are: first, the same standard mechanism and the same implementation can be shared by all applications, resulting in code reuse and reduced code size; second, programmers do not have to deal with the implementation of any security mechanism; this significantly simplifies the development of applications, also in presence of secure communications. Unfortunately, as described in the previous sections, both IPSec and (D)TLS have their own drawbacks. Probably the main issue that is common to both IP and transport approaches is due to the impossibility to assure complete end-to-end security when application communications are relayed by intermediate nodes that work at application level (e.g., proxies). In this case end-to-end security can be still provided with transport or IP level mechanisms, but only in the presence of very trusted intermediate systems. However, in this case, the overall security is complicated by the handling of such hop-by-hop trust management.

A different approach aiming at providing complete end-to-end security is to enforce security directly at application level. This of course simplifies the requirements for underlying layers, and probably reduces the cost, in term of packet size and data processing, since only application data have to be secured and per-data and not per-packet overhead is introduced. Moreover, multicast communications, and in-network data aggregation in encrypted domains (for example through homomorphic cryptography) is easier to be implemented at application level.
The main disadvantages of providing security at application level are the complications introduced for application development and the overall code size due to poor reuse of software codes. This is mainly due to the lack of well defined and adopted secure protocols at application level. Examples of standards that can be used for this purpose are S/MIME and SRTP. S/MIME (Secure/Multipurpose Internet Mail Extensions) [15] is a standard for providing authentication, message integrity, non-repudiation of origin, and confidentiality for application data. Although S/MIME has been originally developed for securing MIME data between mail user agents, it is not restricted to mail and can be used for securing any application data and encapsulated within any application and transport protocols. SRTP (Secure Real-time Transport Protocol) [16] is another secure communication protocol that provides confidentiality, message authentication, and replay protection to application data. It is an extension of the Real-time Transport Protocol (RTP) specifically developed for handling real-time data communications (e.g., voice or video communication), but can be re-used also in other application scenarios. It works in a per-packet fashion and is usually encapsulated in UDP. However, more investigation is required to state which is the standard protocol most suitable for securing data at application level in network and node constrained scenarios such as for IoT.
3.2. Routing Attacks and Countermeasures in the RPL-based Networks

Efforts are underway to connect small and large physical objects with the Internet using IPv6 protocols to form the Internet of Things (IoT). The Routing Protocol for Low-Power and Lossy Networks (RPL) [17] is recently standardized as a routing protocol for the IoT. RPL is primarily designed for low-power and lossy networks (LLNs), also called IPv6 over Low-powered Personal Area Networks (6LoWPAN) networks. A 6LoWPAN network [18] is a Wireless Sensor Network (WSN) that uses compressed IPv6 protocol for networking, and IEEE 802.15.4 as a data-link and physical layer protocol. Unlike in typical stand-alone WSNs, the constrained devices in the IoT are accessible from anywhere. Hence, they are exposed to threats both from the Internet and from within the network.

Potentially any physical object can be connected to the IoT using IPv6. There are a large number of applications for the IoT. The application domains include environmental monitoring, home automation and home security management, industrial automation, smart energy monitoring and management, item and shipment tracking, surveillance and military, smart cities and health monitoring. Real world deployments of the IoT require secure communication which is a challenge because of the heterogeneity of the IoT devices: some are resource constrained and others can be powerful IP connected hosts. It is also important that the communication between the IoT devices should be secured end-to-end (E2E) meaning that the confidentiality and the integrity of messages should be enforced between the source and the destination devices. In order to enforce E2E message security in the IoT using standardized protocols we can use IP security (IPsec) or Datagram TLS (DTLS). Research efforts are underway to securely connect constrained nodes in a 6LoWPAN network with the Internet using lightweight compressed IPsec [19], lightweight DTLS [20, 21], and IEEE 802.15.4 link-layer security [22].

Though message security provides confidentiality and integrity of data packets in transit and authentication between devices, an attacker can still launch a number of attacks against the IoT hosts primarily to interrupt the network. Routing attacks are most common in low power wireless networks [23]. In this work we implement common routing attacks in a 6LoWPAN network where nodes run the Contiki OS [24], the RPL protocol (ContikiRPL [25]) for routing and other novel IoT protocols, and show how the RPL protocol behaves in the presence of a particular routing attack.

To counter attacks in a network, Intrusion Detection Systems (IDSs) are used. An IDS analyzes the activities in the network and tries to detect malicious behavior and/or intruders that are trying to disrupt the network. To this end, we investigate the novel IoT protocols/technologies such as CoAP [26], RPL [17], 6LoWPAN [18] and discuss their strengths and weaknesses which can be exploited by security providers or attackers. Finally, we highlight the new features in the IPv6 protocol that can be used by IDSs or by attackers. To exemplify the use of novel IPv6 security features for intrusion detection we propose and implement a lightweight heartbeat protocol that protects the IoT against selective forwarding attacks.

The main contributions of this work are:

- We investigate how novel features of IoT technologies can be exploited by attackers or IDSs.
- We implement and demonstrate attacks against 6LoWPAN networks running IoT protocols, and we show the effectiveness of well known routing attacks against RPL and how RPL’s self healing mechanisms protect against some of these attacks.
- We also highlight new security features in the IPv6 protocol and provide a lightweight heartbeat protocol to exemplify that these novel features can be exploited for intrusion
The next section discusses IoT technologies with relation to intrusion detection. Section 3.2.2 demonstrates attacks against RPL. In Section 3.2.3 we discuss IDS in the IoT where we present a heartbeat protocol for the IoT. Finally, Section 3.2.4 concludes the work.

### 3.2.1. IoT Technologies and IDS

In this section we discuss RPL and other IoT technologies, and the novel features in the IoT technologies that can be exploited either by attacks to disrupt networks or by the IDSs to defend against intrusions.

**Internet of Things (IoT)**

The Internet of Things (IoT) or strictly speaking the IP-connected IoT is a heterogeneous network that consists of the conventional Internet and networks of constrained devices connected together using IP protocol. The networks of constrained devices in the IoT, called 6LoWPAN networks or an IP-connected WSN, are connected to the conventional Internet using 6LoWPAN Border Routers (6BR). Figure 3 shows the interconnection of things in a 6LoWPAN network with the Internet using the 6BR. *Things* is the IoT are uniquely identifiable objects that sense the physical environment and/or the host devices and communicate this data to the Internet. An IoT device (a thing) can be a light bulb, a thermostat, an home appliance, an inventory item, a smartphone, a personal computer, or potentially anything. IPv6 with its potentially unlimited address space can connect billion or even trillion of these devices with the IoT.

The fact that the devices in the IoT are extremely heterogeneous, many of them are resource constrained, and are globally connected makes it much more challenging to secure the IoT. Especially the constrained devices in the IoT are prone to attacks from the Internet and also from the wireless devices within 6LoWPAN networks. The available IDSs for the Internet and/or for the WSNs may not be suitable to protect IoT devices because they are either too heavyweight for the constrained device or they were not developed in the context of the IoT. Therefore, uncovering the novel requirements of the IoT and providing an IDS for the IoT is worth investigating.

**6LoWPAN**

IPv6 over Low-power Wireless Personal Area Network (6LoWPAN) is a low cost and low power communication network which connects resource constrained wireless devices, typically wireless sensors or actuators, using compressed Internet Protocol version 6 (IPv6). It defines IPv6 header compression and specifies how packets are routed in wireless networks that
Figure 4: A sample RPL DODAG where each node has a unique IPv6 address.

use the IEEE 802.15.4 protocol at the link and physical layer. It also defines fragmentation of IPv6 datagrams when the size of the datagram is more than the IEEE 802.15.4 Maximum Transmission Unit (MTU) of 127 bytes.

6LoWPAN networks support multihop communication where nodes can forward packets on behalf of other nodes. Energy is one of the scarce resources in 6LoWPAN networks and usually most of the energy is consumed on idle listening; therefore, 6LoWPAN networks are usually duty cycled meaning that the radio is turned off most of the time and is turned on only for a very short time for listening.

Due to global IP connectivity, 6LoWPAN networks are vulnerable to most of the available attacks against WSNs plus attacks originating from the Internet. Due to the wireless medium and usually unattended deployments, it is easier to compromise 6LoWPAN devices than typical hosts on the Internet. This gives rise to new threats against the core Internet as the compromised 6LoWPAN devices become sources of attacks against conventional Internet hosts. An IDS for 6LoWPAN networks should consider these vulnerabilities. Also, it is important to consider the capabilities of 6LoWPAN devices when designing an IDS.

CoAP/CoAPs

Due to low-power and lossy links, it is hard to maintain a continuous connection between devices in a 6LoWPAN network. Hence, the connection-less User Datagram Protocol (UDP) is mostly used as the transport layer in 6LoWPAN networks. Further, since connection-oriented web protocols such as HTTP or HTTPS are designed to be used over TCP, a new protocol, the Constrained Application Protocol (CoAP), is being standardized for the IoT. The secure version of CoAP is CoAPs that uses DTLS to protect CoAP messages between two applications in the IoT.

Unlike typical WSNs that have no web protocol, 6LoWPAN networks may use CoAP or CoAPs. Reliability in the CoAP protocol is achieved through the use of confirmable messages. An IDS for the IoT can utilize these built-in reliability and security mechanisms in CoAP/CoAPs to protect IoT devices against many known and potential attacks. For example, a well known attack in the WSN and hence in 6LoWPAN is the HELLO flood that can be detected using reliability mechanisms in the CoAP protocol where devices can check the bidirectionality of paths through CoAP acknowledgments.

RPL

The IPv6 Routing Protocol for Low-Power and Lossy Networks (RPL) is a standardized routing protocol for the IoT. RPL is primarily used in a 6LoWPAN network. RPL creates
Figure 5: Communications can be secured at different layers of the protocol stack, and each solution has its own pros and cons and has its own scope and level of interoperability.

a destination oriented directed acyclic graph (DODAG) between the nodes in a 6LoWPAN. It supports uni-directional traffic towards a DODAG root and bi-directional traffic between 6LoWPAN devices and between devices and the DODAG root (typically the 6BR). There may exist multiple global RPL instances for a single 6LoWPAN network, and a local RPL DODAG can be created among a set of nodes inside a global DODAG. In Figure 4 a RPL DODAG is shown where each node has a node ID (an IPv6 address), a list of neighbors, and a parent node. Each node in a DODAG has a rank that indicates the position of a node relative to other nodes and with respect to the DODAG root. Ranks strictly decrease in the up direction towards the DODAG root and strictly increase from the DODAG root towards nodes.

In order to support downward routing either source routing (RPL non-storing mode) or stateful in-network routing tables (RPL storing mode) are used. Source routing means each packet contains the route the packet is supposed to take through the network. This requires that the DODAG root keeps the information about each node in the network. In a non-storing mode, all forwarding nodes in a RPL DODAG must maintain in-network routing tables to know where to send packets; in-network routing tables differentiate between the packets heading upwards and the packets traveling downwards in the network. For both modes described above the RPL DODAG root maintains a complete list of nodes to support downward traffic.

RPL enables each node in the network to determine whether packets are to be forwarded upwards to its parents, or downwards to its children. Typically, as in the case in ContikiRPL [25] that we use to demonstrate attacks in this work, the simplest way a node can determine the direction of a packet is to know all its descendants which determines the route towards leaf nodes, and consider up direction as the default route of a packet. In RPL storing mode, in-network routing tables are used to separate packets heading upwards and the packets heading downwards in the network.

The RPL protocol provides new ICMPv6 control messages to exchange routing graph information. RPL DODAG Information Objects (DIO) are used to advertise information that are used to build the RPL DODAG. Destination Advertisement Object (DAO) messages are used to advertise information required to support downward traffic towards leaf nodes. Each child node upon joining sends a DAO message to its parents; also, parent nodes can explicitly poll the sub-DODAG for DAO messages using DIO messages. Nodes may use DODAG Information Solicitation (DIS) messages to request graph related information from the neighboring nodes.

The RPL protocol could be vulnerable to the routing attacks demonstrated against WSNs [28] and also to the attacks against the IoT [23]; therefore it is worth investigating the routing attacks against RPL, inherent protection mechanisms in RPL, and new intrusion detection mechanisms for RPL-based networks. We discuss attacks in RPL networks in Section 3.2.2.
Self Healing in RPL

RPL has global and local repair mechanisms that can come into action if there is a routing topology failure, a link failure, or a node failure. On a node (parent) or a link failure a local repair mechanism tries to select a new parent or path. If there are more local failures, RPL performs a complementary global repair where the whole DODAG is rebuilt. The RPL protocol uses the link layer metric as a parameter in the calculation of a default route. The path is assumed to be good if link layer acknowledgements are received on it.

RPL also uses a trickle timer to handle inconsistencies in the RPL DODAG. When a RPL network is stable the trickle timer interval is large. However, upon detection of inconsistencies the trickle timer is reset and more DIO messages are sent (by the nodes) in the vicinity of nodes that are subjected to inconsistencies. The following events are considered as inconsistencies in the RPL:

- When routing loops are detected
- When a node joins a DODAG
- When a node moves within a network and changes rank.

Message Security for the IoT

Security is one of the main requirements in real world deployments of the IoT. Security can be provided on a per-hop basis between two neighboring devices in 6LoWPANs and/or it can be provided end-to-end (E2E) between source and destination nodes. Per-hop security is important to grant access to the wireless medium and to detect message integrity violations as early as possible to hinder constrained resource depletion. Message security in the IoT can be enabled at different layers in the stack using standardized mechanisms; security at the data-link layer using standardized IEEE 802.15.4 security protects messages on a per-hop basis but works with any networking and communication protocol at the upper layer. In addition to the actual messages it can protect the data-link layer and upper layer headers as well. Previously, we have implemented and evaluated IEEE 802.15.4 data-link layer security in the 6LoWPAN [22]. RPL also provides per-hop security between two neighboring nodes which protects the RPL messages. Security at the routing layer (i.e., at the RPL layer) is not needed if link layer security (i.e., per-hop security) is enabled. Also, it is more secure to provide security at the link layer because it can protect the integrity of the link layer and upper layers (that include RPL) as well, and can even encrypt the 6LoWPAN layer and upper layer headers and payloads including the RPL messages.

Security at the IP layer using standardized IPsec is E2E between two hosts on the Internet and works with both TCP and UDP protocols. We have previously provided lightweight 6LoWPAN compressed IPsec for the IoT [19]. Transport/session layer security protects messages between two applications on an E2E basis but only works with one of the transport protocols such as TCP or UDP. In the IoT, UDP is mostly used and hence standardized Datagram TLS (DTLS) can be used. Earlier, we have provided 6LoWPAN header compression for DTLS [21] to make it lightweight for the constrained devices in the IoT.

Despite message security with any of the above mechanisms, IoT devices are still vulnerable to network disruptions, such as DoS attacks. An IDS for the IoT should consider and/or utilize the standardized message security technologies discussed above. Figure 5 summarizes the pros and cons of providing security at different layers.

Intrusion Detection Systems

An Intrusion Detection System (IDS) analyzes activities or processes in a network or in a device and detects attacks, reports them, and/or mitigates the harmful effect of the detected
attacks. Due to the diversity of attacks and the unpredictable behavior of novel attacks, IDSs are subjected to false positives (to raise an alarm when there is no attack) and false negatives (not raising an alarm when there is an attack). Generally, there are two categories of IDSs: signature based and anomaly based. Signature based detections compare the current activities in a network or in a device against predefined and stored attack patterns called signatures. This approach cannot detect new attacks, needs specific knowledge of each attack, has a significant storage cost that grows with the number of attacks, and has a high false negative but low false positive rate. Anomaly based detections determine the ordinary behavior of a network or a device, use it as a baseline, and detect anomalies when there are deviations from the baseline. This approach can detect new attacks but has comparatively high false positive and false negative rates because it may raise false alarms and/or cannot detect attacks. An IDS for 6LoWPAN networks requires a trade-off between the storage cost of the signature based detection and the computing cost of the anomaly based techniques, should counter attackers from the conventional Internet, and should consider that the attackers in a 6LoWPAN can harm both the 6LoWPAN network and the Internet. We propose to complement an IDS for the IoT with a firewall that can be typically placed in the 6BR. Unlike a typical one-way firewall, a firewall for the IoT should block malicious activities and allow benign activities from the Internet to 6LoWPAN networks and vice versa.

### 3.2.2. Attacks against RPL

In this section we investigate the protection capabilities of the RPL protocol against the well-known security attacks presented for WSNs. We experimentally study if the RPL protocol can counter these attacks and/or mitigate their impact.

#### Attack Implementation

We implement well know routing attacks in a 6LoWPAN network where nodes run the Contiki OS [24], a well known operating system for the IoT. Contiki has an implementation of RPL, ContikiRPL [25]. We make use of the RPL implementation in the Contiki OS to implement attacks. ContikiRPL storing mode uses in-network routing where nodes keeps track of all descendants. To provide IP communication in 6LoWPAN we utilize µIP, an IP stack in the Contiki OS. We demonstrate attacks against a simulated RPL network using the Cooja simulator [32]. In our simulations we use emulated Tmote Sky nodes [33] running ContikiRPL.

In Table 2 we highlight the standardized IoT technologies at different layers of the protocol stack that are expected to be used in most of the IoT deployments that rely on interoperability.
among different vendors. We also mentioned the corresponding open source implementations\(^1\) of these IoT technologies in the Contiki OS which we use in this work to demonstrate attacks against the RPL protocol.

**Selective-forwarding Attacks**

With selective forwarding attacks \(^{28}\) it is possible to launch DoS attacks where malicious nodes selectively forward packets. This attack is primarily targeted to disrupt routing paths; however, it can be used to filter any protocol. For example, an attacker could forward all RPL control messages and drop the rest of the traffic. This attack has severer consequences when coupled with other attacks, for example, sinkhole attacks.

One of the solutions to guard against selective-forwarding attacks is to create disjoint paths between the source and the destination nodes. However, it is quite hard to create network-wide completely disjoint paths. To counter selective-forwarding attacks, nodes in the RPL may *dynamically* select the paths to parents/children; as there may be multiple parent or child nodes in the RPL DODAG with almost the same link quality. Also, RPL supports source routing, though not widely implemented, that can be used by an IDS for the IoT to verify path availability in the DODAG.

It is generally very difficult to defend against all selective forwarding attacks. One can, however, defend against many with the use of encryption and analysis of application level traffic. That is to detect if any application traffic is lost and report such losses to the underlying RPL system in order to improve path quality. Another effective countermeasure against selective forwarding attacks is to make sure the attacker cannot distinguish between different types of traffic, thus forcing the attacker to either forward all traffic, or none. In IPv6, ICMPv6 messages are protected by IPsec, hence IPsec can be used to secure the RPL control messages DIO, DAO, and DIS.

**Implementing Selective-Forwarding attacks against RPL**

In our implementation of the selective forwarding attacks we let the malicious node drop all packets except RPL packets. As specified in Algorithm 1, we check in the malicious node running Contiki OS and ContikiRPL if the received packet is not destined to the malicious node and is not a RPL packet, it is dropped. Our selective forwarding attack allows for RPL to function normally, but any application data is lost. We simulate this attack in Cooja and through serial output from the nodes we can verify that the application data is in fact lost from children to the attacker. We run the simulation for 24 hours to allow RPL self healing and self management mechanisms to correct this malicious behavior; however, we could see through the output of the malicious node and its parent node the attack is still active. This means the malicious node still drops all packets except RPL messages, which shows that even after running simulation for 24 hours the RPL self healing mechanisms cannot self correct the network. Therefore, an IDS for the IoT running RPL in 6LoWPAN networks should actively provide countermeasures to detect selective forwarding attacks.

---

\(^1\)ContikiMAC is not a true implementation of the 802.15.4 MAC.

---

**Algorithm 1** Selective Forwarding Attacks in RPL

**Require:** `Packet` - The IPv6 packet received

**Require:** `OwnIP` - The IPv6 address of this node

```plaintext
if Packet.protocol ≠ RPL and Packet.destination ≠ OwnIP then
    Drop packet
end if
```
Sinkhole Attacks

In sinkhole attacks \cite{28} a malicious node advertises an artificial beneficial routing path and attracts many nearby nodes to route traffic through it. This attack in itself does not necessarily disrupt the network operation, however when coupled with another attack it can become very powerful. Ngai et al. present an IDS \cite{34} against sinkhole attacks. Their approach requires two way communication with the nodes, and encryption of the messages. RPL already uses IP and has standardized ways to provide bidirectional communication; E2E message security is enforced using IPsec which is mandatory in IPv6.

Routing protocols that do not use metrics provided by neighboring nodes are immune to sinkhole attacks, as there is nothing for the attacker to spoof. For example, this is the case with preprogrammed routes. The RPL protocol provides several mechanisms to nodes in the DODAG to determine which node to use as its default route. One of them is rank, which is calculated and transmitted by the neighboring nodes, though based on the relative position of nodes from the DODAG root. An attacker can launch a sinkhole by advertising a better rank thus attracting nodes down in the DODAG to select it as parent. RPL, however, uses the link-layer quality to calculate routes which makes sinkhole attack less effective in RPL-based networks.

If the geographical location of the nodes in the RPL DODAG are known the effect of sinkhole attacks can be mitigated by using flow control and making sure that the messages are traveling towards the actual destination. RPL also supports multiple DODAG instances which provides alternative routes to the DODAG root. A potential IDS for the IoT could be hosted in the 6BR and can utilize information from multiple DODAGs to detect sinkhole attacks.

Implementing Sinkhole attacks against RPL

We implement a sinkhole attack in a Cooja simulated RPL network by simply changing the advertised rank when sending RPL control messages, specifically the DIO messages. Any delay normally used to reduce network congestion is also removed in order to allow our malicious node be the first node to advertise such a beneficial route. Figure 6 shows the sinkhole attack is very effective against an ordinary RPL network and causes a lot of traffic to get routed through the attacker. Figure 6 shows node number 26 performing a sinkhole attack. Most of the nodes down in the DODAG select it as their parent. We run this simulation for 24 hours to let RPL DODAG correct itself against the malicious behavior; however, we see no noticeable changes in the network state and the sinkhole attack is still effective, except the nodes with bad links to node 26 choose different parents.
HELLO Flood Attacks

The HELLO message refers to the initial message a node sends when joining a network. By broadcasting a "HELLO" message with strong signal power and a favorable routing metric an attacker can introduce himself as a neighbor to many nodes, possibly the entire network; however, some of the nodes in the attacker’s vicinity when trying to join the attacker their messages may get lost because the attacker might be out of range.

In RPL, DIO messages that are used to advertise information about DODAGs to new nodes can potentially be used to launch a HELLO flood attack. If Secure DIO messages are used for advertisements or link-layer security is enabled the attacker has to compromise a node in order to perform this attack.

Karlof et al. suggest a simple solution to this attack where for each HELLO message the link is checked to be bidirectional [28]. This solution is similar to what is already available in the RPL protocol where it uses the link layer metric as a parameter in the calculation of the default route. If no link layer acknowledgements are received the path is assumed to be bad, and a different route is chosen.

If geographical locations of the nodes in the RPL DODAG are known all packets received from a node that is far beyond the transmission capabilities of ordinary network nodes could be discarded to mitigate HELLO flood attacks. The self healing mechanisms in the RPL, discussed in Section 3.2.1.5, may overcome this attack by trying another parent.

Implementing HELLO Flood Attacks against RPL

We implement a HELLO flood attack against an RPL network and and let the RPL self healing mechanism counter the attack. Using the Cooja simulator we alter the connectivity between the simulated nodes in the RPL network. We thus simulate a HELLO flood by letting a malicious node have the ability to send data to all other nodes in the network, however only nodes physically close to the attacker have the ability to respond. In order to increase the efficiency of the HELLO flood attack we combine it with a sinkhole attack, described in Section 3.2.2.4.

At first the HELLO flood attack interrupts the network as almost all nodes in the network choose the attacker (node 26) as its default route, as shown in Figure 7a. However, nodes soon realize the attacker is in fact not a valid route, and choose a different default route. We show in Figure 7b the state of the network changes using RPL inherent mechanisms and the HELLO
flood attack is automatically mitigated within 10 minutes of its launch. However, nodes 3, 10, 19, and 21 are still connected through the malicious node 26 which shows that the sinkhole attack is not fully eliminated.

Wormhole Attacks
A wormhole is an out of band connection between two nodes using wired or wireless links. Wormholes can be used to forward packets faster than via normal paths. A wormhole in itself not necessarily a breach security; for example, a wormhole can be used to forward mission critical messages where high throughput is important, and the rest of the traffic follows the normal path. However, a wormhole created by an attacker and combined with another attacks, such as sinkhole, is a serious security threat.

As we discussed in Section 3.2.3.1, an IDS for the IoT could place processing intensive modules and a firewall in the 6BR. An attacker can create a wormhole between a compromised constrained node in a 6LoWPAN network and a typical device on the Internet, and can bypass the 6BR. Such a wormhole can become a very serious security breach, and is very hard to detect especially when the wormhole is systematically switched on and off. Ways to prevent or at least detect such a wormhole in the IoT is a research challenge that needs to be addressed.

It is comparatively easy to detect wormholes created within a RPL DODAG. One approach is to use separate link layer keys for different segments of the the network. This can counteract the wormhole attack as no communication will be possible between nodes in two separate segments. Also, by binding geographic information to the neighborhoods it is possible to overcome a wormhole [35]. As wormholes are usually coupled with other attacks, detecting the other attack and removing/avoiding the malicious node will ultimately overcome wormhole attacks.

Implementing Wormhole attacks against RPL
We simulate a wormhole attack by using the network simulator Cooja and set up a physical medium where two nodes on opposite sides of the network have a very good connection. As node 2 and 25, shown in Figure 8, are subjected to a wormhole attack they form a high quality route and the neighboring nodes connect through the malicious nodes 2 and 5. We run this simulation for 24 hours to allow RPL inherent mechanisms to self heal the RPL DODAG. However, the network state has shown that the attack is still there after 24 hours which means RPL does not provide any specific mechanisms to counter wormhole attacks.
Clone ID and Sybil Attacks

In a clone ID attack, an attacker copies the identities of a valid node onto another physical node. This can for example be used in order to gain access to a larger part of the network or in order to overcome voting schemes. In a sybil attack, which is similar to a clone ID attack, an attacker uses several logical entities on the same physical node. Sybil attacks can be used to take control over large parts of a network without deploying physical nodes.

By keeping track of the number of instances of each identity it is possible to detect cloned identities. It would also be possible to detect cloned identities by knowing the geographical location of the nodes, as no identity should be able to be at several places at the same time. The location of nodes or similar information could be stored either centralized in the 6BR or distributed throughout the network in a distributed hash table (DHT) [36].

In an IP/RPL network cloned identities will cause trouble when packets are heading to one of the cloned identities. Packets will be forwarded to one of the cloned identities based on the routing metrics in the network, and the rest of the cloned identities will be unreachable from certain nodes in the network. This however does not affect the network otherwise and therefore cloned identities on their own cause no harm on a 6LoWPAN network.

Implementing Clone ID Attacks against RPL

Using the Cooja network simulator we simulate cloned identities by disabling the multiple-id check in the simulator and simply add several nodes with the same ID. In our Cooja simulated IPv6 network running RPL the cloned identities have the same IP address.

Simulations show that there are no inherent mechanisms in RPL to counter cloned identities. In Figure 9 the cloned identities are indicated in purple and have ID 26. The paths shown with blue arrows represent the downward path. The downward paths from the cloned identities are visualized correct as Cooja nodes in such cases are aware of the source node. However, all nodes which have chosen one of the cloned nodes as their parent will all have their upwards route, the black arrows, pointing towards the leftmost cloned node.

RPL is also subject to alteration and spoofing routing attacks. In RPL a malicious node can send modified rank information to the neighboring nodes. It can also send modified or spoofed DIS, DIO, DAO messages if RPL security is disabled which is the typical case. The 6LoWPAN networks can also suffer from traffic analysis that in itself is not disrupting but the information obtained from analyzing the traffic could be used to launch other sophisticated attacks. Typically, IPsec in tunnel mode or traffic randomization with extra generated traffic is
used to counter these attacks. However, the constrained nature of the IoT devices precludes the applicability of these countermeasures. Usually attacks are not performed in isolation and are combined to get more gains. An IDS for the IoT should consider different possible combinations of these attacks and device solutions to protect network against multiple attacks.

3.2.3. IDS and the IoT

In this section we present a placement of an IDS in a novel IoT setup and propose a mechanism to eliminate malicious nodes in the RPL network. We also discuss intrusion detection capabilities of IPv6 through the heartbeat protocol.

Placement of an IDS in the IoT

Unlike typical WSN that assume no constant connectivity with the sink node, in the IoT the sink node (the 6BR) is assumed to be always available and is not the end point of communication rather things are globally recognizable. This novel architecture, as shown in Figure 3, based on standardized protocol such as RPL and 6LoWPAN gives us more flexibility in the placement of IDSs.

An IDS for the IoT can better utilize this architecture and place processing intensive IDS modules, such as anomaly based detections, in the 6BR, and the corresponding lightweight modules, such as rule or signature based detections, in the constrained sensor nodes. As already discussed, the 6BR has more capacities than a typical resource-constrained sensor node. Any such distributed architecture, however, requires a trade off between the local storage/processing and network communication. Placement of IDS modules in constrained devices will require more storage and processing capabilities; however, these devices have limited resources. On the other hand, a placement of an IDS in the 6BR requires a fresh state of the network, which ultimately incurs more communication overhead between sensors and the 6BR. In LLNs, sending and receiving bits is more power consuming than local processing. Hence it is worth evaluating an IDS approach in both the centralized and distributed placements to better understand its applicability in the IP-connected LLNs. IDS modules in the 6BR have the additional advantage that they can stop intrusion attempts from the Internet. Also, they can block intrusion attempts from inside LLNs against critical infrastructure on the Internet. This is useful since it is easier to physically access and compromise wireless nodes than typical Internet hosts.

Eliminating Malicious Nodes from RPL

Once nodes are detected as malicious it is important to eliminate these nodes from the network. The simplest approach to avoid a fake node is to ignore it which requires identification. In the IoT, both IP addresses and MAC addresses are vulnerable and can be easily spoofed. One possible way to ignore malicious nodes is to use either a whitelist or a blacklist. A whitelist contains all legitimate nodes, whereas a blacklist would include all malicious nodes. On one hand maintaining a whitelist is easier but on the other hand it is not very scalable. Considering that there will be limited devices under one 6BR or in a single RPL DODAG we propose to use a whitelist as it is easy to manage in the presence of many attackers. However, there can be potentially thousands of devices in a RPL network. In such large networks blacklists are easier to manage. In either way it is important that an attacker should not be able to obtain another valid identity since that would enable sybil or clone ID attacks [28].

Intrusion Detection and IPv6

Compared to IPv4 that is mostly used in the Internet today and is well tested, IPv6 is a new protocol and is not yet widely deployed. IPv6 also provides some novel features that can be exploited by both the security provider and the attacker. For example, the Flow Label field in
IPv6 is not protected by IPsec E2E security. Unlike in IPv4, IPsec is mandatory in IPv6. Further, in IPv6 ICMPv6 is protected by IPsec. In this section we use IPsec protected ICMPv6 echo messages and provide a lightweight solution to defend against selective forwarding attacks in the IoT that are otherwise difficult to detect.

**Lightweight Heartbeat**

For a 6LoWPAN network, running RPL or any other IPv6 based routing scheme, we can use a simple heartbeat. Our heartbeat protocol is described in Algorithm 2. In this algorithm, we simply send an ICMPv6 echo request from the 6BR to each node and expect a response. We will notice if traffic is being filtered to and/or from that node if we do not receive an ICMPv6 echo reply. We do this with regular intervals, called heartbeats, to have an up to date picture of the state of the network. ICMPv6 echo/reply mechanisms are widely available in IPv6 networks, hence it is not required that the nodes should be re-programmed to support ICMPv6. For example, in many Contiki OS configurations it is enabled by default.

**Algorithm 2 Lightweight Heartbeat**

Require: Hosts - A list of hosts in the RPL DODAG

Require: Responses - A list of ICMPv6 Echo Replies from the previous iteration of this algorithm

for Host in Hosts do
    ICMP.sendEchoRequest(Host)
end for

for Respons in Responses do
    Hosts.remove(Respons.source)
end for

for Host in Hosts do
    Alarm.raise("Host is offline or filetered", Host)
end for

The heartbeat protocol will work with its full potential if IPsec with ESP [37] is used. Without IPsec, this method will only be able to detect the most simple attacks or if there are faults in the networks for other reasons, for example a broken node. This is because without IPsec, it is possible for an attacker to simply choose to not filter ICMPv6 packets and therefore avoid being detected by this technique. The lightweight heartbeat in an IPsec enabled network would be able to detect selective forwarding attacks as there is no way to distinguish between ICMPv6 traffic and normal traffic as everything after the IPv6 ESP extension header is encrypted, including the ICMPv6 extension header [38]. The heartbeat concept can be extended to potentially detect many attacks, for example, jamming or physically damaging nodes since the nodes would stop responding to ICMPv6 requests.

As a proof-of-concept we implement the heartbeat protocol in a 6LoWPAN network running ContikiRPL and other IoT technologies shown in Table 2. We measured the ROM/RAM and energy overhead of the heartbeat protocol. No additional ROM and RAM is used in the constrained nodes as ICMPv6 is already available in most of the IPv6 implementations including the µIP in the Contiki OS. However, each constrained node in the 6LoWPAN network consumes 0.1158mJ of additional energy to process a single ICMPv6 message. The heartbeat protocol has a little ROM/RAM overhead in the 6BR that sends ICMPv6 messages to the nodes in the 6LoWPAN; however, in the IoT the 6BR is not assumed to be a constrained device.

We also evaluate the network-wide energy overhead of our lightweight heartbeat where the 6BR sends ICMPv6 echo requests to all nodes and each node handles its ICMPv6 reply and
routes replies on behalf of other nodes. In this experiment the RPL DODAG is consist of 16 emulated Tmote Sky nodes. Total energy and power usage by a single node (on average) for one ICMPv6 message from the 6BR to all nodes is shown in Table 3.

An IDS for the IoT should take into account the other unexplored IPv6 features to protect the IoT devices against potential malicious activities. We plan to explore this in the future.

### 3.2.4. Conclusions

In this work we have reviewed novel IoT protocols and highlighted their strengths and weaknesses that can be exploited by the IDSs. We have shown that while the RPL protocol is vulnerable to different routing attacks it has inherent mechanisms to counter HELLO flood attacks and mitigate the effects of sinkhole attacks. An IDS for the IoT can be complemented with the novel security mechanisms in the IPv6 protocol; for example, our heartbeat protocol can defend against selective forwarding attacks.

The aim of this work is to highlight the importance of security in the RPL based IoT and to provide grounds to the future researchers who plan to design and implement IDSs for the IoT.

Table 3: Energy and power usage of one node in an RPL network for one heartbeat compared with RPL only (no heartbeat).

<table>
<thead>
<tr>
<th>Overhead</th>
<th>Energy (mJ)</th>
<th>Power (mW)</th>
</tr>
</thead>
<tbody>
<tr>
<td>RPL only</td>
<td>202.6</td>
<td>1.702</td>
</tr>
<tr>
<td>RPL and Heartbeat</td>
<td>225.3</td>
<td>1.893</td>
</tr>
<tr>
<td>Heartbeat only</td>
<td>22.7</td>
<td>0.191</td>
</tr>
</tbody>
</table>
3.3. Real-time Intrusion Detection in the Internet of Things

With IPv6 over Low-power Wireless Personal Area Network (6LoWPAN) [27, 18] it is possible to connect resource constrained devices, such as sensor nodes, with the global Internet using the standardized compressed IPv6 protocol. These networks of resource constrained devices, also called 6LoWPAN networks, and the conventional Internet form the Internet of Things or strictly speaking the IP-connected Internet of Things (IoT). A 6LoWPAN Border Router (6BR) is an edge node that connects 6LoWPAN networks with the Internet. Due to the resource constrained nature of the devices or things, 6LoWPAN networks mostly use IEEE 802.15.4 as link and physical layer protocol.

Unlike typical wireless sensor networks (WSN), 6LoWPAN networks or IP-connected WSN are directly connected to the untrusted Internet and an attacker can get access to the resource-constrained things from anywhere on the Internet. This global access makes the things vulnerable to intrusions from the Internet in addition to the wireless attacks originating inside 6LoWPAN networks. Potential applications of the IoT are smart metering, home or building automation, smart cities, logistics monitoring and management, etc. These applications and services are usually charged and the revenue is based on data or services used. Hence, the confidentiality and integrity of the data and timely availability of services is very important.

Researchers have already investigated message security for the IoT using lightweight DTLS [39], IPsec [19], and IEEE 802.15.4 link-layer security [22]. Even with message security that enables encryption and authentication, networks are vulnerable to a number of attacks aimed to disrupt the network. Hence, an Intrusion Detection System (IDS) is necessary to detect intruders that are trying to disrupt the network.

The available IDSs for WSNs could be used in the IoT. However, most of these approaches are built on the assumptions that (i) there is no central management point and controller, (ii) there exists no message security, and (iii) nodes cannot be identified globally. The IoT has a novel architecture where the 6BR is assumed to be always accessible, end-to-end message security is a requirement [22], and sensor nodes are globally identified by an IP address. Besides these opportunistic features, an IDS for the IoT is still challenging since the things (i) are globally accessible, (ii) are resource constrained, (iii) are connected through lossy links, and (iv) use recent IoT protocols such as CoAP [40], RPL [17], or 6LoWPAN [18]. Therefore, it is worth investigating and providing an IDS for the IoT exploiting these opportunities and threats.

To this end, we design, implement, and evaluate a novel Intrusion Detection system for the IoT that we call SVELTE\(^2\). To the best of our knowledge this is the first attempt to develop an IDS specifically designed for the IoT. Network layer and routing attacks are the most common attacks in low power wireless networks [22], and in this work we primarily target these attacks. SVELTE is also inherently protected against sybil and clone ID attacks; we discuss these attacks in Section 3.3.1.3. We evaluate SVELTE against sinkhole and selective-forwarding attacks. Our approach is, however, extensible and can be used to detect other attacks as we discuss in Section 3.3.5.

The IPv6 Routing Protocol for Low-Power and Lossy Networks (RPL) [17] is a novel standardized routing protocol primarily designed to meet the specific routing requirements of the IoT. SVELTE uses RPL as a routing protocol. It has two main components: the 6LoWPAN Mapper (6Mapper), and intrusion detection modules. The 6Mapper reconstructs RPL’s current routing state, i.e., its directed acyclic graph, at the 6BR and extends it with additional intrusion detection parameters.

One of the important decisions in intrusion detection is the placement of the IDS in the

\(^{2}\text{SVELTE literary means elegantly slim.}\)
network. We use a hybrid approach, see Section 3.3.1, and place the processing intensive SVELTE modules in the 6BR and the corresponding lightweight modules in the constrained nodes. Figure 10 presents an overview of our IDS that we explain in more detail in Section 3.3.1.

One of our main design goals is that the IDS should be lightweight and comply with the processing capabilities of the constrained nodes.

In addition to the 6Mapper and the intrusion detection techniques, we also propose and implement a distributed mini-firewall to protect 6LoWPAN networks against global attackers from Internet. We implement SVELTE in the Contiki operating system [24].

The main contributions of this work are:

- We present SVELTE, a novel IDS with an integrated mini-firewall for the IP-connected IoT that uses RPL as a routing protocol in 6LoWPAN networks.
- We implement SVELTE and thoroughly evaluate it for 6LoWPAN networks that consist of resource-constrained things and have lossy communication links.

The next section of this work gives an overview of the technologies used in SVELTE. Section 3.3.1 describes SVELTE that includes 6Mapper, the actual intrusion detection techniques, and the firewall. In Section 3.3.2 we detail SVELTE’s implementation for the Contiki OS. Section 3.3.3 presents our detailed performance evaluation of SVELTE. We highlight the current IDSs and their applicability in the IoT in Section 3.3.4. Section 3.3.5 discusses the possible extensions in SVELTE, and finally we conclude the work in Section 3.3.6.

### 3.3.1. SVELTE: An IDS for the IoT

Recall that a 6LoWPAN network is a lossy and wireless network of resource constrained nodes which uses IPv6 as networking protocol and often RPL as a routing protocol. One of the design goals of any protocol for the IoT is its ability to be deployed and run on constrained nodes in 6LoWPAN networks. Based on the novel requirements of the IoT, we propose SVELTE: a lightweight yet effective intrusion detection system for the IoT. We also compliment SVELTE with a distributed mini-firewall in order to filter malicious traffic before it reaches the resource constrained nodes.

We design SVELTE for a 6LoWPAN network that uses message security technologies, such as IPsec [19] and DTLS [39] to provide end-to-end message security. In the rest of this section we present our intrusion detection system.

**Placement of SVELTE**

The placement of an IDS is an important decision that reflects the design of an IDS and the detection approaches. Keeping in view the resource constrained nature of the devices and the IoT setup shown in Figure 10 we use a hybrid, centralized and distributed, approach and place IDS modules both in the 6BR and in constrained nodes.

Figure 10: An IoT setup where IDS modules are placed in 6BR and also in individual nodes.
SVELTE has three main centralized modules that we place in the 6BR. The first module, called 6LoWPAN Mapper (6Mapper), gathers information about the RPL network and reconstructs the network in the 6BR, as we describe in Section 3.3.1.1. The second module is the intrusion detection component that analyzes the mapped data and detects intrusion; Section 3.3.1.3 discusses this. The third module, a distributed mini-firewall, is designed to offload nodes by filtering unwanted traffic before it enters the resource constrained network; Section 3.3.1.4 details this. The centralized modules have two corresponding lightweight modules in each constrained node. The first module provides mapping information to the 6BR so it can perform intrusion detection. The second module works with the centralized firewall. Each constrained node also has a third module to handle end-to-end packet loss; this is discussed in Section 3.3.1.3.

6LoWPAN Mapper

A vital component of SVELTE is the 6LoWPAN Mapper (6Mapper) that reconstructs the RPL DODAG in the 6BR and complements it with each node’s neighbor and parent information. To reconstruct the DODAG, the 6Mapper sends mapping requests to nodes in the 6LoWPAN network at regular intervals. The request packet contains the information necessary to identify an RPL DODAG. It includes the RPL Instance ID (IID), the DODAG ID, and the DODAG Version Number \[17\]. It also includes a timestamp (\(Ts\)) to know the recency of the mapping information received. The total size of a mapping request packet is 5 bytes.

Each node responds to the mapping request by appending a Node ID to the request packet and by appending node rank, parent ID, and all neighbor IDs and ranks. An illustration of the mapping response packet format is shown in Figure 11. The basic response packet is 13 bytes long and requires an additional four bytes for each neighbor.

6Mapper with Authentic and Reliable Communication

It is likely that IPsec Authentication Header (AH) \[19\] or IEEE 802.15.4 link-layer security are enabled in the IoT to protect the integrity of the IP headers. In this case there is no need to include the node ID in the response packet, as that would be the source address in the IP header. When the 6Mapper host, i.e., 6BR, has the same IPv6 address as the DODAG root it is also unnecessary to include the DODAGID that corresponds to the destination IP in the IP header. In the request packet the source and destination fields in the IP header have the opposite meaning, i.e., the IP source corresponds to the DODAGID and the destination corresponds to the node ID.

If mapping-packets are transferred reliably, for example, by using CoAP that employs acknowledgements, there is no need to send a timestamp with the mapping data as we can be sure that the packets arrive within the timeout specified for the underlying protocol. When the communication in the 6LoWPAN is authentic and reliable, the size of the 6Mapper request and response packets is reduced to 1 byte and 8 bytes, respectively.
**Unidirectional RPL 6Mapper**

Some RPL implementations only support traffic destined to the DODAG root, typically the 6BR. To provide network mapping for these 6LoWPAN networks it is possible to alter the 6Mapper and let it wait for the periodic mapping response packets from each node without sending the explicit request packet. This solution has the additional advantage that it reduces traffic in the network which reduces power consumption. However, slightly more logic has to be added in each node which increases the memory consumption.

**Valid inconsistencies in 6Mapper**

In our 6Mapper there is a possibility that mapping responses are inconsistent with each other, which can lead to false positives if not handled properly. This can happen if the information a node sends to the network mapper has become outdated or when an attacker deliberately changes the information. Below we show how valid routing graph inconsistencies occur. Consider a RPL DODAG where Node $P$ is the parent of node $C$, the function $R_a(Node)$ represents the actual rank of Node and $R_m(Node)$ represents the rank known to the 6Mapper.

- Node $P$ sends its rank to the 6Mapper, $R_a(P) = 1024$ and $R_m(P) = 1024$
- Node $P$ recalculates its rank and advertises it, $R_a(P) = 512$ and $R_m(P) = 1024$
- Node $C$ receives the updated rank from $P$
- Node $C$ recalculates its rank. $R_a(C) = 768$
- Node $C$ sends its rank to the 6Mapper, $R_a(C) = 768$ and $R_m(C) = 768$

As can be seen the state of the network is:

$R_a(P) = 512$
$R_a(C) = 768$
$R_m(P) = 1024$
$R_m(C) = 768$

This state is perfectly valid as node $P$ has a better rank than node $C$, $R_a(P) < R_a(C)$. However, the 6Mapper assumes that the child, node $C$, has a better rank than its parent, which is inconsistent as $R_m(P) > R_m(C)$.

This is a problem which needs to be taken into consideration when designing methods for analyzing the mapped data. Leveraging the amount of sensors in a 6LoWPAN we improve the accuracy when faced with both natural and artificial inconsistencies; Section 3.3.1.3 discusses methods to overcome such inconsistencies.

**Mapping requirements**

For our 6Mapper to be fully effective the packets used to map the network need to be indistinguishable from other packets. If an adversary can distinguish the traffic used by the 6Mapper from other traffic it is possible for an adversary to perform selective forwarding and only forward traffic necessary for the mapper, while dropping other traffic.

The first step to prevent this is to encrypt the data, to avoid that the packet content is revealed to an eavesdropping adversary. As mentioned earlier we assume that the message contents are protected with upper-layer security protocols such as IPsec or DTLS. Secondly, headers should not reveal any information that enables an eavesdropper to determine that the packet is used by the 6Mapper. Therefore it can be problematic if the source of the 6Mapper is the same for all nodes, as the IP header must be readable for all nodes. The adversary could use the IP header and the knowledge about the 6Mapper’s host address to identify network mapping traffic. A simple solution to prevent this is to assign as many IPv6 addresses to the 6Mapper as there are nodes in the network. This is possible for RPL as IPv6 has a potentially
Algorithm 3 Detect and Correct the RPL DODAG Inconsistencies

```plaintext
Require: N - A list of nodes
for Node in N do
    for Neighbor in Node.neighbors do
        Diff = |Node.neighborRank(Neighbor) - Neighbor.rank|
        Avg = (Node.neighborRank(Neighbor) + Neighbor.rank) / 2  {If the absolute difference is greater than 20% of the ranks average}
        if Diff > Avg * 0.2 then
            Node.fault = Node.fault + 1
            Neighbor.fault = Neighbor.fault + 1
        end if
    end for
end for
for Node in N do
    if Node.fault > FaultThreshold then
        Node.rank = Rank reported for Node by any neighbor
    end if
end for
```

unlimited address space of $2^{128}$ addresses. Thus, when an adversary compromises a node it will only know the node’s mapping address and no other mapping addresses. Hence, it is not able to distinguish between ordinary traffic and mapping traffic for other nodes.

However, if the attacked node has more resources it may use more advanced traffic patterns and node behavior analysis techniques, and it might still be possible for an adversary to distinguish between ordinary and mapper-related traffic.

**Intrusion Detection in SVELTE**

We design and implement three detection techniques which use the 6Mapper. The detection techniques primarily detect spoofed or altered information, sinkhole, and selective forwarding attacks. However, our approach is extensible and more attacks can be detected; we discuss some of the possible extensions in Section 3.3.5.

**Network Graph Inconsistency Detection**

In the IoT individual nodes may be compromised by an attacker and later used to launch multiple attacks. For example, in RPL-based 6LoWPAN networks the attacker can use compromised nodes to send wrong information about their rank or one of their neighbor’s rank to the 6Mapper. It is also possible to get an incorrect or inconsistent view of the network because of the lossy links in the IoT. It is therefore important to detect the inconsistencies, distinguish between valid and invalid consistencies, and correct the invalid information. The complete algorithm to detect and correct the routing graph inconsistencies is described in Algorithm 3.

In order to detect incorrect information and to make sure that information is consistent across the network, each edge in the network is checked. The 6Mapper provides node ID and rank of each node, of its parents, and of its neighbors. We iterate over each edge in the network, checking that both nodes agree with each other about their rank and detect the inconsistencies. It is possible that a false alarm is raised because the detected incorrect information is a result of valid mapping inconsistencies described in Section 3.3.1.1.

In order to distinguish between valid and invalid inconsistencies, or to avoid false positives, we rely on (i) the number of reported faulty ranks and (ii) the difference between the two reported ranks. We use a simple threshold, referred to as FaultThreshold in Algorithm 3 and classify a node as faulty if the number of disagreements this node has with other nodes are
larger than the threshold. Most of the disagreements between two nodes are small and a result of varying link quality and ultimate RPL adjustments. To accommodate valid inconsistencies, we only consider disagreements where the difference of the two nodes ranks is greater than 20% of the ranks average; this value is based on our empirical evaluation of SVELTE.

We correct the faulty information when both of the above conditions are met, i.e., once we have large inconsistencies towards a node. The faulty information corresponding to a node is corrected by changing the rank known to 6Mapper by substituting it with the information reported by one of its neighbors. The neighbor information is updated with the information reported directly by its neighbors.

Once it is detected that a routing inconsistency is a result of a deliberate attack, SVELTE either removes the faulty node or corrects the inconsistency. SVELTE keeps track of inconsistencies and if it is the first time a node is detected as malicious it is not immediately removed as it may be a false alarm or result of a passive attack; in this case the faulty information is corrected as described above. However, if the same node is detected as faulty again it is removed by deleting its entry from the whitelist maintained in the 6Mapper.

Checking Node Availability

It is important to detect if a node or set of nodes are available and operating properly. When a particular node is compromised it may launch multiple attacks to disrupt the network. For example, it may launch a selective forwarding attack and intelligently drop messages. If an RPL network uses CoAP to send application data the attacker could forward RPL traffic but drop CoAP traffic. This would result in a seemingly working network even though no useful traffic gets through.

Depending on the RPL implementation and the configuration, we can use the RPL routing table in the RPL DODAG root as a basis for available nodes in the network. As we require a whitelist of valid nodes in the network for access control we could also use that list as a basis for detection.

When we compare the whitelisted nodes with the nodes in our RPL DODAG all differences are offline nodes or unauthorized nodes. Let $W$ be a set of all whitelisted nodes and let $R$ be the nodes known to RPL in the RPL DODAG root, the offline nodes, $O$ are thus:

$$W \setminus R = O$$

where $O$ is the relative complement ($\setminus$) between two sets $W$ and $R$ meaning that $O$ contains all elements of $W$ that are not in $R$.

It is however not possible to determine if nodes excluded from $O$ are being filtered or are simply offline. That is, if an attacker performs a selective forwarding attack and filters everything but RPL messages it would with the previous method appear as if the nodes are still online, even though all application data is being filtered. By extending the above method with the information available through 6Mapper it is also possible to detect selective forwarding attacks. Let $M$ represent nodes known to 6Mapper and $F$ be the filtered nodes we get the following relationship:

$$W \setminus M = F$$

As the 6Mapper for each node keeps track of the last time it received a packet from a node we can detect filtered nodes by simply checking if we have not recently received any packets from them. In order to mitigate the effects of packet loss or other similar events common in lossy networks we introduce a threshold on the time since our last packet. We define the threshold as a number of mapping-requests allowed to be unanswered. With this threshold it is
possible to alter the sensitivity of the filtered node detection to be easily adaptable to specific deployments. Algorithm 4 describes this behavior and finds all filtered nodes $F$ in a network. 

**Routing Graph Validity**

By artificially altering the routing graph, an attacker can reshape the topology of the network and can control the traffic flow to his advantage. For example, an attacker performs a sinkhole attack by advertising a very good rank to its neighbors. The problem becomes more severe if the sinkhole attack is coupled with other attacks. A sinkhole attack can, for example, enable the attacker to intercept and potentially alter more traffic than otherwise. If combined with a selective forwarding attack a much larger part of the network can be controlled. It is therefore important to detect such attacks.

With SVELTE, it is possible to detect most sinkhole attacks by analyzing the network topology. If the routing graph is inconsistent it is likely an attack is in place. In RPL, the rank in the network should be decreasing towards the root, i.e., in any child-parent relation the parent should always have a lower rank than the child. All cases where a child has a better rank than its parent is an indication of routing graph incoherency, as specified in [17].

When an incoherency is found the child in the relation is at fault, as a node should never have a lower rank than its parent. With such a simple approach false positives are likely to arise, i.e., we detect inconsistencies while in fact all nodes are working properly.

In order to minimize the effects of valid inconsistencies, that can raise false positives, we require several consecutive inconsistencies to be reported for the same nodes. That is we require more than one sample of the network to have the same incoherency to raise an alarm. This is described in the Algorithm 5 as $FaultThreshold$ which is a global state kept between consecutive runs of the detection algorithm. In RPL the rank between any host and its parent is at least $MinHopRankIncrease$ [17]. We utilize this in our algorithm to better conform to the RPL standard.

A sinkhole attack would in most cases be detected by this algorithm. As the attacker advertises a beneficial rank it will most likely have to advertise a better rank than its parent and as such would be detected by the detection scheme described above. If a sinkhole attack is
Algorithm 6 Adapt to End-to-end Losses

Require: dest - The destination with packet loss

\( \text{nexthop} = \text{getNexthop}(\text{dest}) \)
\( \text{nexthop.metric} = \text{nexthop.metric} \times 0.8 \)

Figure 12: Network configurations and node placement that are used in the experiments in this section

to remain undetected the advertised rank of a malicious node must not be better than that of its parent. This would in turn result in the adversary’s rank only being slightly improved over a non-adversarial node and thus yield little benefit.

In RPL, the rank as well as the parent selection is calculated via an objective function, which might use factors such as link quality in its calculation; for example when the Expected Transmission Count (ETX) \([41]\) is used to calculate rank. The ETX is an approximation of the link quality and as such a bad link might affect the choice of parent more than a slight difference in rank. This would further lower the impact of a sinkhole attack that is undetectable by Algorithm 5.

End-to-end Packet Loss Adaptation

We design an intentionally simple system to take end-to-end losses into account when calculating the route and to mitigate the effects of filtering hosts. If a reliable higher-layer protocol such as TCP or CoAP (with confirmable messaging) is used, packet loss can be detected using the protocol’s acknowledgement mechanism. The reasoning behind a host-to-host packet loss indication is that if an attacker is filtering packets some hops down the path we want to be able to adapt to it. In the RPL-based network, if a packet is filtered somewhere on the path a new parent should eventually be tried.

The approach is not able to adapt to every form of filtering, for example, when the attacker is located such that all packets have to go through it. If however a collection-scheme with acknowledgements is also running in the network all data losses should be corrected for. Since all nodes will try to send data to the sink all nodes with a path through the attacker will also notice the losses and correct for them, given that the attacker filters all application data. If a packet is not able to reach its destination, we slightly alter the route metric of the route, that is the next-hop neighbor for that packet, (20% in Algorithm 6) to reflect that there might be an attacker along the path. Algorithm 6 describes end-to-end packet loss adaptation.

Sybil and CloneID Attacks Protection

In a Sybil attack an attacker copies several logical identities on one physical node whereas in a cloned identity (CloneID) attack the attacker copies the same logical identity on several physical nodes. Both attacks are aimed to gain access to a large part of the network or in order
Algorithm 7 Mini-firewall

Require: Host - The host to report
Require: Source - The node that sent the report
Require: GlobalFilter - A set of external hosts to filter towards all nodes
Require: LocalFilter - A map mapping an external host to a set of local nodes. The set describes all nodes that have reported that specific external host.

if Host in GlobalFilter then
    return Host already filtered
end if

if Host in LocalFilter then
    Filter = LocalFilter.get(Host)
    \{Add Source to the list of nodes blaming Host\}
    Filter.add(Source)
    if Filter.size() ≥ ReportThreshold then
        GlobalFilter.add(Host)
        LocalFilter.remove(Host)
    end if
end if

to overcome a voting scheme. The 6Mapper only considers the latest information received from each host in the network where a host is identified by an IP address. A sybil attack has no direct effect on the 6Mapper as it makes no difference if the identities are on the same physical node as if they are separate physical entities, each host is treated individually in both cases. While cloned identities can interrupt the routing in a network it does not affect the 6Mapper directly as the 6Mapper only considers the latest information received from one of the identity. As a result if two cloned nodes send information to the 6Mapper there is no difference compared to if one node sends the information twice, thus not directly affecting the operations of the 6Mapper. Sybil attacks and cloned identities are both often used to disrupt different voting schemes by giving an attacker more votes. Voting schemes based upon 6Mapper collected data will be unhindered by both sybil attacks and cloned identities.

Distributed Mini-firewall

Though SVELTE can protect 6LoWPAN networks against in-network intrusion, it is also important that the resource constrained nodes are protected against global attackers that are much more powerful. For example, it is easier for hosts on the Internet than constrained nodes in 6LoWPAN networks to perform denial of service attacks. Firewalls are usually used to filter external hosts and/or messages destined to local networks. As the end-to-end message confidentiality and integrity is necessary in the IoT, the SVELTE module in the 6BR or a firewall cannot inspect the contents of the encrypted messages; therefore, it is hard to distinguish between the legitimate and malicious external traffic.

We propose a distributed mini-firewall that protects a 6LoWPAN network from external hosts. The firewall has a module in the 6BR and in the constrained nodes, and is integrated with SVELTE. Our firewall, besides providing typical blocking functionally against well-known external attackers specified manually by the network administrator, can block the external malicious hosts specified in real-time by the nodes inside a 6LoWPAN network.

The destination host inside a 6LoWPAN node can see the encrypted contents and hence analyze the malicious traffic and notify the 6BR in real-time to filter traffic coming from the compromised host, therefore stopping the traffic before it reaches the constrained nodes. When a constrained node notices an external host being abusive it sends a packet with the host IP to the firewall module in the 6BR. As is the case with the 6Mapper, if IPsec with Authentication Header is used the nodes own ID can be omitted. Otherwise, the nodes own ID need to be included. If the node ID is included it can be compressed down to 2 bytes using 6LoWPAN
header compression mechanisms. The external host however can neither be compressed nor omitted as it can be any valid IPv6 address. Therefore the minimal size of the filtering-request packet is 16 bytes. With the node ID the size of the packet is 18 bytes.

In order to make sure that no internal compromised node can abuse this mechanism by requesting filtering of traffic from a legitimate external host, both the source and the destination is taken into account when filtering. The node inside a 6LoWPAN network can only choose to filter the traffic destined to itself. Such a firewall is still easy to circumvent as the attacker can simply target another node in the network and start the attack again; therefore, we extend the firewall to adapt and block any external host if a minimum set of nodes complain about the same external host. Our mini-firewall is described in Algorithm 7.

To be more preventive against global attackers, our mini-firewall can be extended with AEGIS [42], a rule-based firewall for wireless sensor networks.

3.3.2. Implementation

We implement SVELTE and the mini-firewall in the Contiki OS [24], a well known operating system for the IoT. Contiki has a well tested implementation of RPL (ContikiRPL). As SVELTE is primarily designed to detect routing attacks we make use of the RPL implementation in the Contiki operating system to develop the 6Mapper, the firewall, and the intrusion detection modules. The RPL implementation in Contiki utilizes in-network routing where each node keeps track of all its descendants. We borrow this feature to detect which nodes should be available in the network. To provide IP communication in 6LoWPAN we use µIP, an IP stack in Contiki, and SICSLoWPAN- the Contiki implementation of 6LoWPAN header compression. We also implement the sinkhole and selective forwarding attacks against RPL to evaluate SVELTE. SVELTE is open source\footnote{For the source code visit: http://www.shahidraza.info} and is available to researchers and industry.

3.3.3. Evaluation

In this section we present the empirical evaluation of SVELTE. After describing our experimental setup, we quantitatively evaluate the detection rate and the true positives for each experiment. We also measure the overhead of SVELTE both at the node-level and network-wide. We evaluate the overhead in terms of energy consumption and the memory footprint.

Experimental Setup

We run our experiments in Contiki’s network simulator Cooja [32] that has shown to produce realistic results [43]. Cooja runs deployable Contiki code. In our simulations, we use emulated Tmote Sky [33] nodes.

In general, we expect that the 6BR is not a constrained node and it can be a PC or a laptop; however, currently there exists no PC equivalent 802.15.4 devices, therefore we run the 6Mapper natively, i.e., on Linux, and communicate with Cooja using a serial socket. For RPL with 6Mapper we run each test 10 times, and calculate the average and standard deviation to show the accuracy and precision of our results. On the other hand, the experiments with RPL only (without the 6Mapper) have no processing intensive components and hence require no native parts. Therefore, the experiments with RPL-only yield the same results for all experiments as we use the same seed.

SVELTE Detection and True Positive Rate

Here we quantitatively evaluate the detection rate, i.e., the number of malicious nodes successfully detected against the total number of malicious nodes present in the system, and the true
Figure 13: For the smaller lossy network, SVELTE has 90% true positive rate against sinkhole attacks which decreases for larger networks but gets better when RPL becomes stable.

Figure 14: SVELTE has acceptable true positive rate in both lossy and lossless network considering that we have almost 100% detection rate for selective forwarding attacks.

positives rate, i.e., the total number of successful alarms divided by the total number of alarms. We use three different configurations shown in Figure 12a, 12b and 12c. In each configuration node no 1 (green) is the 6BR. Using these settings, we run experiments for 5, 10, 20, and 30 minutes. In all experiments, the 6Mapper is configured to request data and to perform analysis every two minutes. Therefore, the first 6Mapper request will be sent after two minutes. The first analysis is also performed after 2 minutes but will however not yield any results as no data is yet gathered. Therefore the earliest possible detection time is after four minutes. It is important to note that these are the settings in our experiments and not the requirements for SVELTE. The malicious nodes can spoof or alter information, and/or can perform sinkhole or selective forwarding attacks. In the following experiments SVELTE first performs network graph inconsistency detection as described in Section 3.3.1.3 before detecting sinkhole or selective forwarding attacks. Each experiment is run in a lossy and in a lossless network. Lossless links provide the perfect scenario for 6Mapper, as all requests and responses return without delay and loss, and we get a true picture of the network. This is further improved by the fact that nodes more quickly can propagate their ranks down in the network graph. The real 6LoWPAN networks are mostly lossy, therefore we consider both cases in our evaluation. The loss model is Cooja’s default radio model that uses a Unit Disk Graph Medium (UDGM): Distance Loss. UDGM models the transmission range as a circle in which only the nodes inside the circle receive packets. The UDGM Distance Loss model, an extension of UDGM, also considers interference.
Sinkhole Attacks with and without losses

The results for the sinkhole attack in a lossless network scenario show almost 100% true positive rate on the first possible attempt to analyze the network and no false positives are detected during the simulations. A lossless network configuration means that all requested data is gathered quickly and without losses, which implies that the map of the network is a perfect representation of the actual network. Because of this it is very easy to detect all sinkhole attacks without any false positives. In the lossy network configuration, Figure 12a, the true alarm rate is approximately 90%, as shown in Figure 13. However, with the increase in network size the true alarm rate decreases; this is because for the larger network configurations it takes some time before the RPL network and our map of the network become stable and complete enough to arrive at a higher true positive rate. For example, in the scenario with 16 nodes it takes 30 minutes to arrive at the same true positive rate as is done with 8 nodes after 10 minutes. The reason Figure 13 shows a non-existent detection rate for the case of 5 minutes is because we only raise an alarm if the same node has been misbehaving for more than two consecutive executions of our algorithm. Hence, the current configuration implies that a sinkhole attack can be detected after 6 minutes. Our approach does not require collection of two consecutive messages or executions to work. Collecting multiple messages is advantageous to make sure that it was actually an attack and not a sudden link fluctuations, for example due to interference. If the attack persists for two consecutive executions of our algorithms then we raise an alarm; this is done primarily to reduce false positives.

From these results it is evident that SVELTE is very effective against sinkhole attacks in a network with no or few losses, and in lossy networks it is more effective when the RPL network has become stable.

Selective Forwarding Attack with and without losses

In a selective forwarding attack a malicious node filters traffic going through it. Hence, the 6Mapper will not be able to get any data from any children of the malicious nodes in the network. This in turn has the effect that the results of the 6Mapper depend on the actual network topology, i.e., in the lossless case, unlike with sinkhole attacks, the results are not always 100%. We can see the effects of this phenomenon in Figure 14a. In a lossy network, as shown in Figure 14b, there is a gradual increase in the true positive rate going towards a bit over 80% in all cases. As the network is lossy messages are naturally lost, and if that happens several consecutive times when mapping we are going to get more false positives. If we raise the various thresholds in our detection algorithms it is possible to lower the number of false alarms, possibly at the cost of a decreased detection rate. In order to reduce number of false positives we may use location information of the nodes as discussed in Section 3.3.5.

We also measure the detection rate during all of the above experiments. We achieve 100% detection rate meaning that we can detect all malicious nodes that launch sinkhole and/or selective forwarding attacks. It should be noted that the 100% detection rate is for the current set of experiments with the current setting; we do not claim that SVELTE should achieve 100% detection rate in all settings. As can be seen in Figure 13 and 14 the true positive rate is not 100%, i.e., we have some false alarms during the detection of malicious nodes. This is mostly caused by our configuration. It might be possible to alter the behavior of our detection algorithm, for example, by changing the threshold used in Algorithm 4 and thus possibly get a different result with regards to detection rate and/or false alarm rate.
Energy Overhead

The nodes in the IoT are usually battery powered and hence energy is a scarce resource. Here we measure SVELTE's power consumption both at node-level and at system-level. We use Contiki Powertrace [44] to measure the power consumption. The output from the Powertrace application is the total time the different parts of the system were on.

We calculate the energy usage and power consumption using the nominal values, the typical operating conditions of the Tmote sky, shown in Table 4. We use 3V in our calculations. In the rest of this work MCU idle while the radio is off is referred to as low power mode, or LPM. The time the MCU is on and the radio is off is referred to as CPU time. The time the radio is receiving and transmitting with the MCU on is referred to as listen and transmit respectively. We measure energy in both duty cycled 6LoWPAN networks, where the radio is mostly off, and in non duty cycled networks where the radio is always on for listening and transmitting.

Network-wide with Duty Cycling

Here we evaluate network-wide energy consumption of an RPL network with and without the 6Mapper and intrusion detection mechanisms in a duty cycled network. We use ContikiMAC [31], a duty cycling MAC protocol in Contiki. We use the default ContikiMAC setting that has 8 wakeups per second and without traffic the radio is on for 0.6% of the time. We run each experiment in a network of 8, 16, 32 and 64 emulated Tmote sky nodes, with nodes placed at the same locations.

Figure 15 shows the network-wide energy usage for 30 minutes by all the nodes, calculated as follows

\[
\text{Energy(mJ)} = (\text{transmit} \times 19.5\text{mA} + \text{listen} \times 21.8\text{mA} + \text{CPU}\times 1.8\text{mA} + \text{LPM}\times 0.0545\text{mA}) \times 3\text{V}/4096 \times 8
\]
Table 5: Energy consumption for handling a single event inside a constrained node.

<table>
<thead>
<tr>
<th>Event</th>
<th>Energy (mJ)</th>
</tr>
</thead>
<tbody>
<tr>
<td>6Mapper Response Handling</td>
<td>0.1465</td>
</tr>
<tr>
<td>Firewall handling</td>
<td>0.0478</td>
</tr>
<tr>
<td>Packet lost correction</td>
<td>0.0483</td>
</tr>
</tbody>
</table>

Table 6: Out of total 48k of ROM size in a constrained device (Tmote sky), SVELTE requires 1.76k. However, in the 6BR (typically a PC) the size grows when the number of nodes increases.

From the network wide energy usage, we calculate the average power as,

\[
\text{Power (mW)} = \frac{\text{Energy (mJ)}}{\text{Time (s)}}
\]

which when divided by the total number of nodes gives us the per node average power consumption during the experiment. Figure 15b shows the power consumption per node. As can be seen in Figure 15a and 15b the overhead of the 6Mapper is negligible for small networks (up to 16 nodes) and increases with the number of nodes. The total overhead of SVELTE is approximately 30% more than running RPL only for networks with 64 nodes. Recall that with duty cycling the radio is off for approximately 99% of the time.

Network-wide without Duty Cycling
We use the same network settings as in Section 3.3.3.5 and run the experiments in a non duty cycled network where the radio is always turned on to receive and transmit packets. When we compare the results of RPL with the 6Mapper plus intrusion detection algorithms we see that the overhead is negligible. This is because the radio is always on and most of the nodes’ energy is consumed on idle listening.

In-node Energy Overhead
Here we measure the energy consumption of handling a single event of the 6Mapper and the firewall inside a constrained node. Table 5 lists the energy required to perform different tasks; this does not include the energy needed to send/receive packets which we have included in Section 3.3.3.5 and 3.3.3.5. As can be seen in Table 5, a constrained node consumes very little energy for local processing as most of the processing intensive tasks are performed in the 6BR where the 6Mapper and the main SVELTE detection modules reside. Therefore, the energy consumed for in-node processing is clearly negligible.

Memory Consumption
In Table 6 we show the extra ROM requirements of SVELTE’s different modules. The baseline for each configuration is different as some depend on different parts of the Contiki system. For example, the 6Mapper that resides in the 6BR (typically a PC) requires more ROM than other nodes. However, the total additional ROM required to host SVELTE’s modules inside a constrained node is 1.76k which is well below the total available ROM in constrained devices such as 48k in Tmote sky. In Table 6 it is important to note the overhead column which shows the pure overhead of SVELTE modules in Contiki. Even though 6Mapper is not targeted
Table 7: Additional RAM usage by SVELTE for handling a single event inside a constrained node.

<table>
<thead>
<tr>
<th>Event</th>
<th>RAM (byte)</th>
</tr>
</thead>
<tbody>
<tr>
<td>6Mapper Response Handling</td>
<td>162</td>
</tr>
<tr>
<td>Firewall handling</td>
<td>24</td>
</tr>
<tr>
<td>Packet lost correction</td>
<td>188</td>
</tr>
</tbody>
</table>

towards running on constrained nodes it is still lightweight enough and can be used for small networks.

We also measure the RAM size of 6Mapper response handling, firewall, and packet loss correction which we show in Table 7. The total RAM size in the Tmoke sky is 10kb, hence SVELTE modules with 0.365k additional RAM requirement can easily run in constrained nodes.

3.3.4. Related Work

The IoT is a rather old concept and for many years RFID-based sensors were considered as things in the IoT. With the inception of 6LoWPAN, lightweight IP is being standardized and used in the IoT for the unique identification and global connectivity of the things. Even when confidentiality and integrity are enforced by message security solutions such as IPsec it is possible to disrupt the IoT. A number of attacks against the IoT have been identified in addition to those against WSN that are also applicable to the IoT. Therefore, it is important to have systems that detect such attacks.

The concept of intrusion detection is quite old and extensive research is carried out in this field mostly against the Internet attacks and attacks against WSN. However, no IDS are specifically designed in the context of IoT. Most of the IDS approaches for WSN are based on a distributed architecture and are built on the limitation that there is no centralized management and control point. A common IDS approach for WSNs is to utilize several special nodes distributed evenly throughout the network. These special nodes can either be physically different or dynamically distributed throughout the network. In real deployments, however, it cannot be guaranteed that particular nodes are always present in specific locations in the network; also, the cost of employing mobile agents that move through the network might be too high. Clustering based approaches have similar issues as each cluster often requires a powerful entity for coordination. The IoT has a novel architecture where the 6BR is always assumed to be accessible and is a potential place for centralized management and control. SVELTE make use of this novel IoT architecture and presents a new placement for IDS. Using a mix of centralized and distributed architecture SVELTE takes advantage of both realms.

Many IDS approaches are based upon watchdog techniques which could be used in the IoT. In addition to being distributed and fully deployed on sensor nodes, a general problem with watchdog based approaches is that they require promiscuous listening, which consumes a lot of power and therefore is not suitable for constrained devices. Advanced anomaly detection approaches are proposed, not primarily for WSNs, which on one hand can detect many intrusions efficiently but on the other hand requires intelligent learning, which is both expensive and difficult in low powered 6LoWPAN networks.

Most current IDS approaches require different routing schemes that are not based on standardized mechanisms. As far as we are aware, no approach is built around 6LoWPAN and RPL in the context of the IoT. Our approach considers RPL to decrease the cost of performing intrusion detection. Likewise, we have taken into account the fact that there is a central entity, the 6BR, that connects the sensor network with the conventional Internet, which is a standard
based networking solution [27, 18, 17].

We do not claim that no other IDS approach can be used in the RPL-connected IoT. Rather we argue that these approaches are built on different assumptions that do not fully hold in the IoT architecture. Also, the IoT gives rise to new challenges that do not exist in typical WSNs. However, there is a potential to incorporate already available approaches in the SVELTE architecture. We discuss below the possibilities to integrate available lightweight IDS approaches in SVELTE.

### 3.3.5. SVELTE Extensions

One of the main advantages of our approach to intrusion detection is that the proposed and developed system is very easy to extend. There are a number of potential attacks against the Internet of Things and it is likely that more attacks will be discovered. As such extendability is very important for an IDS. The 6Mapper is easy to extend both conceptually and in practice. If a new detection scheme requires more data to be added to the network graph the response packets can easily be extended. Also, using the already available data that we collect through the 6Mapper it is possible to apply anomaly detection techniques, for example via the use of Support Vector Machines [52], feature vectors [53], or automata based approach [54].

**Wormhole Detection** One of the important to detect attacks in wireless networks is wormhole [55]. If the 6Mapper is extended with the signal strength of each node’s neighbor it is also possible to detect wormhole attacks [56].

**Pinpointing filtering node** If a node is filtering traffic it is beneficial to be able to pinpoint more accurately which node is performing the filtering. The most straightforward approach is to perform a traceroute [57] towards one of the missing nodes.

**Location Information** RPL is primarily designed for static networks, though it can be extended to support mobility [58], it is possible to add node’s location in the 6Mapper at the deployment time. The location of a node can also be estimated in real-time using localization techniques [59]. These location information help SVELTE to build a physical map of the network that will ultimately enhance its intrusion detection capabilities. For instance, with this physical map rank modification and hence the sink-hole attack can be detected with even lesser false positives alarms. The location information of nodes will also help SVELTE to mitigate the sybil and CloneID attacks aimed to disrupt the routing information [36].

### 3.3.6. Conclusions

6LoWPAN networks will be an integral part of the IoT. Considering the potential applications of the IoT it is important that 6LoWPAN networks are protected against internal and external intrusions. To this end we present SVELTE, the first IDS for the IoT which consists of a novel architecture and intrusion detection algorithms. We implement and evaluate SVELTE and show that it is indeed feasible to use it in the context of RPL, 6LoWPAN, and the IoT. To guard against global attacks we also design and implement a mini-firewall.

The detection algorithms in SVELTE currently target spoofed or altered information, sink-hole and selective forwarding attacks. However, it is flexible and can be extended to detect more attacks. Therefore, we plan to complement SVELTE with novel and/or available intrusion detection techniques that are feasible to use in the context of the IoT.
3.4. Compression of IPsec AH and ESP Headers for Constrained Environments

[draft-raza-6lowpan-ipsec-01]

This document describes the header compression mechanisms for the IPsec [RFC4301] based on the encoding scheme standardized in RFC 6282. The IPsec Authentication Header (AH) and Encapsulated Security Payload (ESP) headers are compressed using Next Header Compression (NHC) defined in RFC 6282. This document does not invalidate any encoding schemes proposed in 6LoWPAN RFC 6282 but rather complements it with compressed IPsec using the free bits in the IPv6 Extension Header encoding.

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79. Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/. Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as “work in progress”. This Internet-Draft will expire on March 7, 2014.

Copyright and License Notice

Copyright (c) 2013 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

3.4.1. Introduction

RFC 6282 defines how IPv6 datagrams can be routed over IEEE802.15.4-based networks. RFC6282 defines a header compression schemes that can significantly reduce the size of IP, IP extension, and UDP headers. This enables the routing of heavy-weight IP traffic to resource-constrained IEEE802.15.4-based wireless network. The security in IEEE802.15.4-based IP network or what is more commonly known 6LoWPAN networks is particularly important as we connect the insecure Internet with the vulnerable wireless network. The standardized and mandatory security solution for IPv6 is IP security (IPsec) [RFC4301]. This means that every IPv6 host on Internet is able to process IP packets secured with IPsec. IPsec, in transport mode, can provide end-to-end (E2E) secure communication between the two hosts in the IP network. Thus, it is beneficial to extend 6LoWPAN so that IPsec communication between an IPv6 device (e.g. a sensor node) in 6LoWPAN and IPv6 nodes on conventional Internet becomes possible. This document does not cover the tunnel mode of IPsec.

With IPv6 architecture it is possible to protect ICMPv6 messages, using IPsec. As the RPL Control Message [RFC6550] is an ICMPv6 message, it is therefore possible to protect it with IPsec. However, all RPL Control Message, except DAO / DAO-ACK messages in non-storing mode, are exchanged between two neighboring devices and have the scope of a link. Though IPsec security associations can be created between two neighboring devices, IEEE 802.15.4
security at the link layer is more suitable for per-hop protection, and IPsec in transport mode can be used to protect DAO/DAO-ACK messages in non-storing mode.

It is desirable to complement 6LoWPAN header compression with IPsec to keep packet sizes reasonable in resource constrained IEEE802.15.4-based network. There are no header compression specified for IPsec’s AH [RFC4302] and ESP [RFC4303] extension headers. This draft therefore proposes AH and ESP extension header encoding schemes.

**Terminology**
The key words “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY”, and “OPTIONAL” in this document are to be interpreted as described in RFC 2119.

### 3.4.2. Linking IPsec Headers Compression with 6LoWPAN

RFC 6282 defines the general format of NHC that can be used to encode IP extension headers. RFC 6282 already defines an NHC encoding for IPv6 Extension Headers (NHC_EH) that can be used to link uncompressed AH and ESP headers to the 6LoWPAN header compression. In order to compress the IP extension headers a GHC byte for Extension Header (GHC_EH) [draft-bormann-6lowpan-ghc-05] is proposed which has the same layout as NHC_EH with different ID bits. NHC_EH and GHC_EH consist of an octet where three bits (bits 4, 5 and 6) are used to encode the IPv6 Extension Header ID (EID). Out of eight possible values for the EID, six are assigned and the remaining two slots (101 and 110) are currently unassigned. As AH and ESP are IP extension headers it makes sense to use one of these unassigned slots for the IPsec headers. We propose to use the reserved slot 101 for the IPsec headers, AH or ESP. The corresponding ID field in the AH or ESP will distinguish these headers from each other. It is also necessary to set the NH bit in NHC_EH or GHC_EH to 1 to specify that the next header (a header after AH or ESP, e.g. UDP) is NHC-encoded.

#### 3.4.3. 6LoWPAN_NHC for Authentication Header

6LoWPAN can be used to compress a significant number of bits in AH. The next header is decided based on the value of NH bit in the IPv6 Extension Header Encoding in RFC 6282. This draft proposes to always elide the length field. The payload length field (the length of AH header in 32-bit words units minus “2” [RFC4302]) in the AH header is always elided, as it can be inferred from the lower layers: either from the IEEE 802.15.4 header or the 6LoWPAN header. The size of ICV can be obtained from the SPI value because the length of the authenticating data depend on the algorithm used and are fixed for any input size. The RESERVED field in the AH header is also always elided. The SPI and SN are compressed using the proposed NHC encoding for the AH header shown in Figure 16 and are explained below.

![Figure 16: Proposed 6LoWPAN NHC encoding for AH.](image)

- The first four bits in the NHC AH represent the NHC ID we define for AH. These are set to 1101.
- If SPI = 00: the default SPI for the 802.15.4 network is used and the SPI field is omitted. We set the default SPI value to 1. This does not mean that all nodes use the same security
association (SA), but that every node has a single preferred SA, identified by SPI 1. If SPI = 01: the least significant 8 bits of the SPI are carried inline; the remaining 24 bits are elided. If SPI = 10: the least significant 16 bits of the SPI are carried inline; the remaining 16 bits are elided. If SPI = 11: All 32 bits of the SPI are carried inline.

- If SN = 00: the least significant 8 bits of sequence number are carried inline. The remaining bits are elided. If SN = 01: the least significant 16 bits of the SN are carried inline; the remaining 8 bits are elided. If SPI = 10: the least significant 24 bits of the SPI are carried inline; the remaining 8 bits are elided. If SPI = 11: All 32 bits of the SPI are carried inline. The sequence number field in the AH header [RFC4302] contains a value 1 for the first packet sent using a given Security Association (SA), and it is incremented sequentially for the subsequent packets. Note that by using 8-bit sequence number we do not limit the size of sequence number to 255, but propose to use 8 bits for the sequence number prior to the transmission of the 256th packet on an SA. From the $2^8$ to $2^{(16-1)}$ we propose to use 16-bit sequence number. Follow the same procedure for the 24-bit sequence number as well. However, the sender and the receiver sequence number counters must be reset prior to sending $2^{32}$nd packet as proposed in RFC 4302.

Note that even when used in 6LoWPAN, AH calculates the ICV on the uncompressed IP header, thus allowing authenticated communication with Internet hosts. The minimum length of a standard AH, supporting the mandatory HMAC-SHA1-96 [RFC4835], consists of 12 bytes of header fields plus 12 bytes of ICV. Figure 17 shows a sample NHC compressed IP/UDP packet secured with AH. Using NHC encoding for the AH we can reduce the AH header overhead from 24 bytes to 14 bytes: 1 byte of next header, 1 byte of length, 2 bytes of Reserved field, 4 bytes of SPI, and 2 bytes of sequence number. However, two additional bytes are used to define NHC_EH and NHC_AH. Therefore, in the best case, with AES-XCBC-MAC-96 [RFC3566] or HMAC-SHA1-96 ciphers (when 12 bytes are used for ICV), applying NHC encoding for AH saves 8 bytes in each data packet secured with IPsec AH.

![Figure 17: A sample NHC compressed IP/UDP packet secured with AH.](image)

### 3.4.4. LOWPAN_NHC for Encapsulated Security Payload (ESP)

The encryption in the IPsec ESP includes Payload Data, Padding, Pad Length and Next Header fields in the ESP. Therefore, we cannot compress these fields at the 6LoWPAN layer, and these fields are always carried inline. Also, when using ESP the UDP header and payload is also
encrypted, hence cannot be compressed using NHC encodings for UDP defined in the RFC 6282. However, we can compress the SPI and and sequence number (SN) fields in the ESP header. Figure 18 shows a proposed NHC encodings for the ESP that are explained below.

![Figure 18: Proposed LOWPAN NHC encoding for ESP.](image)

- The first four bits in the NHC ESP represent the NHC ID we define for ESP. These are set to 1110.
- The SPI and SN bits are encoded exactly the same way as in Section 3.4.3 for the AH header.

In case of ESP we cannot skip the next header unless the end hosts are able to execute 6LoWPAN compression/decompression and encryption/decryption jointly. The nodes in the 6LoWPAN network make their decision about the next header based on the NH value not the actual header that is carried inline. In the case of ESP we MUST set the NH value in the NHC_EH or GHC_EH to zero to indicate that the full 8 bits of next header field are carried inline. With perfect block alignment, the minimum ESP overhead without authentication is 10 bytes

![Figure 19: A sample NHC compressed IP/UDP packet secured with ESP.](image)

[RFC4303]. After optimal compression this header overhead is reduced to 6 bytes, considering that two bytes are used for NHC_EH and NHC_ESP. ESP also includes an IV which is equal to the size of an encryption block; 16 bytes in the case of AES. If authentication is enabled in the ESP, additional 12 bytes of ICV are also required. Figure 19 shows an UDP/IP packet secured with compressed ESP.

### 3.4.5. Implementation Considerations

We provide an open source implementation of the proposed compression scheme in the Contiki operating system. The implementation is released under BSD license and can be obtained
through the contikiprojects repository at the following URI:
svn://svn.code.sf.net/p/contikiprojects/code/sics.se/ipsec

3.4.6. Security Considerations

The compression scheme proposed in this document does not compromise any of the security provided by IPsec AH and ESP. In particular, the SN field is compressed in an on-demand fashion, as described in Section 3.4.3. In order to overcome replay attacks, it is recommended that the communication end-points should re-establish a security association before the sequence number overflows. However, in constrained environments, different implementations can decide the overflow size; $2^8$, $2^{16}$, $2^{24}$, or $2^{32}$. This leads to a trade-off between the overhead incurred by establishing a new security association and by sending more bits of sequence number. The Initialization Vector (IV) and Integrity Check Value (ICV) are also not compressed to take full advantage of IPsec AH and ESP security.

3.4.7. IANA Considerations

RFC 6282 creates a new IANA registry for the LOWPAN_NHC header type where the two slots, 1110101N and 1110110N, in LOWPAN_NHC for the IPv6 Extension Header are unassigned. This document requests the assignment of one of these two unassigned values, 1110101N, to IPsec AH and ESP. This document also requests the assignment of following contents:

- 1101XXYY: The 6LOWPAN_NHC encoding for the IPsec Authentication Header.
- 1110XXYY: The 6LOWPAN_NHC encoding for the IPsec Encapsulated Security Payload Header.

Capital letters in bit positions represent class-specific bit assignments. The letters XX and YY represent SPI and SN respectively, as defined in Section 3.4.3.
3.5. Secure distribution of shared keys

According to a group communication paradigm, a single member can originate and deliver a message to the whole group of nodes, through multicast (or broadcast) communication services [60], and thus in a more efficient manner than an equivalent unicast-based solution. The first applications taking benefit of the group communications model, such as online gaming and audio/video streaming [61], have historically operated on the Internet. In recent years, the ever increasing diffusion of ad-hoc (mostly wireless) networks has offered a new fertile ground for the development of new types of group-based applications. In scenarios such as wireless sensor networks [62], mobile ad-hoc networks [63], and Internet of Things (IoT), a large number of applications (e.g., data dissemination, data gathering, peer-to-peer communications) need an underlying multicast data delivery service.

Securing group communications consists in providing confidentiality, authenticity, and integrity of messages exchanged within the group, through suitable cryptography services [64], and without interfering with the data path of the multicast data flow [66]. The achievement of this goal in an efficient and scalable manner is a challenging task since it requires that a large and dynamically varying number of users share cryptographic materials, even in the presence of unpredictable group membership changes due to new users entering (joining) the network and to old users leaving the network. In fact, after any membership change, the shared cryptographic materials should be refreshed through a suitable rekeying operation, so that a former group member has no access to current communications (forward secrecy) and a new member has no access to previous communications (backward secrecy) [67, 68].

While authenticity and integrity protection in group communications can be easily achieved through asymmetric cryptography, like in traditional point-to-point communications (e.g., through digital signatures), the simplest and most scalable way to provide data confidentiality within a multicast group is to encrypt the data through symmetric cryptography, with a secret key shared (only) by all users belonging to the group. Such symmetric key is normally referred to as group key.

Under the assumption of using security primitives unbreakable for an attacker with limited computational power, the main issue in group communications consists in distributing such secret group key to all the legitimated users and updating it at any group membership change. This problem is known as Group Key Distribution (GKD) and it can be tackled by following two different models [69]: (i) Broadcast Encryption (BE) [70, 71], which assumes that current data be decipherable independently of past transmissions (the receivers are stateless); (ii) Multicast Key Distribution (MKD), which allows the users to maintain state of the past cryptography material [72] (stateful). There are two main categories of MKD protocols: centralized [73] or distributed [74]. According to the former, the keys’ distribution task is assigned to a single entity, denoted as Key Distribution Center (KDC). In the case of the distributed approach instead, the group key is established and maintained by the users themselves, in a distributed fashion. The centralized MKD approach has several advantages: (i) simplicity; (ii) a small number of exchanged messages compared to other methods; (iii) the possibility of operating on intrinsically broadcast channels, where the source (which also acts as MKD server) sends data to all the possible destinations. Distributed methods typically offer greater reliability, since they do not require any centralized entity to trust, but they have higher communication and computational costs and are not applicable to asymmetric communication scenarios where data cannot be exchanged between any pair of nodes. For these reasons, in the rest of this paper we

---

*Iouis* [65], for example, is a scheme that interferes with the normal packet stream, since a group security intermediary has to decrypt and encrypt all the packets transiting in its own group.
will focus on centralized approaches.

In a centralized MKD protocol, among the different methods to achieve secure communications in a group of $n$ members, one of the simplest consists in having: (i) a group key, shared by group members only and changed every time a user joins or leave the group [75]; (ii) $n$ individual long-term keys shared (pairwise) between the KDC and every group member. A message sent by a member to the whole group is encrypted with the group key, so that only the remaining members can decrypt it. Instead, the individual keys are used for securing unicast communications and for reeking. The management of the group keys has a cost, in terms of number message exchanges, that varies according to the protocol used to update the group key to all members (rekeying). In the simplest case, the KDC separately sends the new group key, encrypted with the member’s long-term key, to all the group members, thus determining a number of exchanged messages proportional to $n$. The overall number of communications also depends on the average number of rekeying. In some protocols, the group key is refreshed suddenly after any join or leave events, in these cases the number of rekeying is directly proportional to the number of change of membership events. In other cases, the group key is refreshed programmatically according to a slotted schedule. These protocols have a constant number of rekeying operations, but they reduce the freedom of the nodes, and make impossible to perform instant evictions of malicious or dangerous users. Besides this classification, it is possible to define hybrid protocols, where join and leaves are performed programmatically, while evictions are performed immediately.

The technique described in this work is based on a key derivation scheme properly extended in order to deal with both unpredictable leave events and collusive attacks. In particular, we present a MKD protocol tailored for very dynamic ad-hoc networks, either wired or wireless. Time is partitioned in fixed-length intervals, each of them associated with a different group key. Even if a user can join anytime (asynchronously), it shall wait until the beginning of the next slot before becoming a group member. This introduces a delay, on average equal to half of the slot interval, but allows to reduce the number of rekeying acts. Similarly, the planned leave of a legitimate member shall also happen at the beginning of a slot period. In other words, the protocol is slotted and adopts a synchronous batch rekeying mechanism [75] that improves efficiency without posing security threats. The protocol also provides proper mechanisms to deal with unpredictable leave events and to resist against collusive attacks.

The aim of the protocol is to minimize the computational burden of group members and the overhead, expressed in terms of number of exchanged messages, while achieving a sufficiently high security level. The proposed protocol can operate on very dynamic scenarios with a large number (thousands) of nodes and offers excellent performance under the assumption of low rate of evictions.

### 3.5.1. Related Works

In multicast group communications, a proper MKD protocol is required for generating and distributing a secret group key that can be used to secure (encrypt) data sent from one source to all destinations that are member of the same group. Since multicast groups are often very dynamic, due to the join of new members and the leave of old members, the MKD has to handle such group membership changes by re-generating and re-distributing new group keys.

More precisely, the group key should be changed after every join and leave through a suitable rekeying operation, so that a former group member has no access to current communications and a new member has no access to previous communications [67]. These requirements can be expressed by introducing the concepts of forward secrecy and backward secrecy [68]. According
to the former, non-members should not be able to obtain the group key at any instant based only on the information obtained at or before that instant. A more strict requirement, is the concept of backward secrecy, according with the group key at any instant should not be computable by non-members even after that instant (in other words, the new comers cannot compute past group keys). Moreover, group communication should be resistant to collusion attacks, in which (past or current) member of the group exchange information “out-of-band” in order to illicitly have access to information.

Join and leave operations can happen anytime (in an asynchronous and dynamic fashion), or alternatively, they can be synchronized at specific instants (in a slotted manner). In the second case, a number of join/leave operations should be jointly managed in the same temporal slot and for this reason these mechanisms are also referred as “batch” methods. In this work, we focus on this kind of mechanisms, since they significantly reduce the complexity (quantified in terms of number of exchanged messages) and they fit well the characteristics of realistic services, such as online game, where the join/leave operations have a daily or an hourly granularity.

MKD protocols have been previously classified in two main categories: centralized or distributed. While centralized key distribution protocols rely on a centralized key server (KDC) to efficiently distribute the group key, in distributed key agreement protocols there is no centralized server and the key is generated in a shared and contributory fashion by the member of the group, usually named peers. Such distributed protocols are often based on the multi-party extension of the well-known Diffie-Hellman key agreement protocol. For example Kim et al. [76] propose a Tree-based Group Diffie-Hellman protocol (TGDH) where each member maintains a set of keys, which are arranged in a hierarchical binary tree. A secret key and a blinded (or public) key are associated to every tree node. The secret key of a non-leaf node can be generated by the secret key of one child node and the blinded key of the other child node, similar to the classical two-party Diffie-Hellman protocol. Each group member is associated to a leaf node and the corresponding secret and blinded keys, that can be used to generate all secret keys along the path toward the root node secret key that, hence, can be generated by all nodes and corresponds to the group key. The TGDH protocol has been further optimized in [74]. However, distributed protocols need a larger amount of exchanged messages and operations, and they typically require full peer-to-peer communications, thus leading to a greater complexity than the more used centralized protocols. For these reasons centralized protocols are often preferred and used instead.

As described previously, in a common centralized MKD scenario, the KDC may share an individual long-term secret key with every user of the network, while a shared short-term key is used as group key and refreshed after any membership change (or programmatical) using the long-term keys. However, this plain centralized solution is not scalable since the number of communications required for the rekeying operation is linear in the size of the current group (denoted as n). These results should be compared with the known lower bound $O(\log_2 n)$ [68].

Wong et al. [67] proposed the Logical Key Hierarchy (LKH) approach, based on key graphs, where keys are arranged into a hierarchy, and the key server maintains all the keys. The LKH scheme makes use of symmetric-key encryption (as the only cryptographic primitive), and has a number of communications approaching the lower bound in [68].

If a user wants to join the group, it sends a join request to the key server. The user and key server mutually authenticate each other using a protocol such as Secure Socket Layer (SSL). If authenticated and accepted into the group, the user shares with the key server a symmetric key, called the user’s individual key.

In [77], the authors propose the MARKS protocol, which is scalable and requires no key update messages. However, MARKS only works if the leaving time of a member is set when
the member joins the group, so that members cannot be expelled. Besides the scheduled leaves, there is also the possibility of unpredictable leaves, which occurs when a user is evicted from the group. In this case, it is unsafe to delay the rekeying until the next time slot, and it is necessary to provide a mechanism which allows immediate revocation of all the cryptographic materials known by the evicted user.

In a previous work [78], we presented a very simple algorithm for key derivation, which however does not specify any management scheme to deal with unpredictable leave events and does not protect against collusive attacks. In this work, in order to allow rapid unpredictable evictions, we superimpose an existing asynchronous key management mechanism, the LKH scheme [67], to our slotted protocol.

### 3.5.2. New Group Key Management Protocol

In this section, a new group key distribution protocol is presented. The proposed protocol allows a server (KDC) to efficiently distribute a group key to all members of a multicast group dealing with dynamic joins and leaves of users as group members. The proposed solution is summarized in Subsubsubsection 3.5.2.1 and detailed in Subsubsubsections 3.5.2.2 and 3.5.2.3.

#### Protocol Overview

Let us consider a multicast group communication scenario in which the same data has to be securely sent to a group of destinations. In order to guarantee data confidentiality, the sent message has to be encrypted with a secret (group) key shared by, and only by, all group members. We consider a dynamic scenario in which, at any time, a new user may join the system as new group member and an old user may leave the group. As described in the previous sections, this requires a suitable group key distribution protocol, able to distribute a new key to all members upon every change of group membership. We consider a key distribution scenario based on a trusted KDC that takes care of: (i) maintaining a secure association with all users belonging to the system; (ii) generating a new group key every time the group membership changes; (iii) efficiently managing the distribution of the new group key to all group members, guaranteeing both forward and backward secrecy.

In a more general scenario, join and leave operations occur unpredictably, in a completely asynchronous and dynamic way. However, in order to optimize and significantly reduce the complexity and the number of exchanged messages required to handle group member changes and group key re-distribution (rekeying), a more practical method is to allow the KDC to handle simultaneously a number of membership changes. This can be achieved by splitting time into intervals (sometimes referred to as “time slots” or, simply, “slots”) and letting the KDC handle all membership changes that occur in the same time interval. Key distribution mechanisms that work in this way are often referred to as “batch” methods. Note that our proposed method applies when these time intervals have the same length or different lengths. However, very common scenarios are those in which membership changes are handled, for practical reasons, in a daily or monthly manner: this is the case, for example, of applications that consider service subscriptions with specific durations (expressed exactly in days or months). Other common possible time slot units can be minutes, seconds, or years.

Although the time slot in which a new user wants to join the system is in general difficult (or impossible) to predict (as it can apply at any time), there are many application scenarios in which the duration of the membership of a user is specified at the moment when the user joins the system, possibly further extended on the basis of a renewal strategy. Service subscriptions are often handled by applications in this way, with the possibility (in a limited subset of cases) of considering some form of revocation mechanisms in order to handle situations (often seen as
exceptions) in which a membership has to be revoked in advance before its natural expiration
time (for example, if a user unexpectedly leaves the system or if he/she is removed due to a
misuse or for administrative reasons).

In spite of the above considerations, the majority of the proposed key management mech-
anisms do not take advantage of this operation and simply consider any leave event as not
pre-determined, as it always occurs randomly.

On the opposite, we explicitly consider two different kinds of leave events: (i) “pre-determined”
leave events, when the leave time is selected in advance when the user joins the network or when
it refreshes his/her membership, as in the case of a natural membership expiration; (ii) “un-
predictable” leave events, when the time of leave does not coincide with the one selected at the
time of joining or refreshing, for example in the case of explicit membership revocation. In our
method, like in [77], both kinds of leave events are explicitly considered, taking the advantage
of the balance of the former leaving strategy with respect to the latter.

We consider a different group key $K_i$ for each time slot $i$ with $i = 0, 1, 2, \ldots, N$. In order
to efficiently handle both kinds of leave events, the group key is obtained through a one-way
function of two sub-keys $K_1$ and $K_2$:

$$K_i = f(K_1, K_2)$$

with $K_1$ and $K_2$ properly managed in order to handle both kind of leaves. In particular,
the values $K_1$ are associated to every time slots $\Delta t_i$ (with $i = 0, 1, 2, \ldots, N$); they are pre-
determined and provided to group members according to their assigned membership duration.
The values of $K_1$ are generated in an intelligent and secure manner in order to simplify the
assignment to joining users, by providing only some root secret materials that can be used by
the member to further derive all $K_1$ values associated with all time slots he/she subscribed
for. $K_1$ are then used to handle all new join and “pre-determined” leave events.

On the other hand, $K_2$ is used to handle all “unpredictable” leave events. It is changed
and re-distributed by the KDC to all (and only) group members, in a scalable way, similarly to
other mechanisms already proposed in the literature.

Since the amount of operations and exchanged messages differ for managing of the subkeys
$K_1$ and $K_2$, the total amount of operations and exchanged messages is a function of the rate
of the “unpredictable” leave events over join and “pre-determined” leave events.

Details of how $K_1$ and $K_2$ are derived and managed are hereafter described.

Protocol details
The objective of the proposed key management protocol is to provide a group key that can be
securely shared by (and only by) all group members, taking into account and properly handling:

1. regular membership changes, that are due to new users that join the group and ac-
tive members that leave the group for “clean” membership expiration (“pre-determined”
leaves);

2. exceptional active member leaves, e.g., in the case of explicit membership revocation
(“unpredictable” leaves).

In order to take into account membership changes of type 1, the overall time span is considered
divided into a sequence of $N$ time slots $\Delta t_i$ with $i = 0, 1, 2, \ldots, N$ and in general, $\Delta t_i \neq \Delta t_j$
for $i \neq j$. In practice, however, it will be common to have $\Delta t_i = \Delta t_j = \Delta t \forall i, j$, with $\Delta t$ equal
to standard time units, such as a minute, a second, a month, etc. For each time interval $\Delta t_i$, in
the following referred as “slot” $i$, a different group key $K_i$ is determined. Consider now a user
member $x$ that will belong to the group from time $t_a$ to time $t_b + 1$, i.e. from time slot $\Delta t_a$ to time slot $\Delta t_b$: he/she will receive the subset of keys $S_X = \{K_i\}$ with $i = a, a + 1, a + 2, \ldots, b$. According to the above approach, as far as only membership changes of type 1 are considered, the KDC is requested to generate all keys $K_i$ and give to each new incoming member only the subset of keys corresponding to the time slots over which he/she will belong to the group. If the member will stay for a total of $m$ time slots, this will require the KDC to give to the new member $m$ different keys. In order to limit the total amount of cryptographic material that the KDC has to send to each new member, a proper distribution protocol is adopted.

However, regular membership changes (type 1) are not the only events that require the assignment and distribution of a new group key (i.e., a rekeying operation). In the case of an unpredictable leave event (type 2) in time slot $\Delta t_h$ of member $y$ that negotiated with KDC a membership from time slot $\Delta t_a$ to time slot $\Delta t_b$, at least all previously assigned keys (from $K_h + 1$ to $K_b$) must be re-assigned and distributed to all valid group members. This is needed in order to prevent $y$ to decrypt messages that are sent after time slot $\Delta t_h$ with valid keys that he/she received by the KDC in joining the group.

To handle both types of membership changes in a secure and flexible way, the following key derivation and distribution protocol is proposed.

Let us consider $N$ time slots, with $N = 2^D$. Each time slot $\Delta t_i$ is associated with a key $K_i$ defined as:

$$K_i = f(K_{1i}, K2) \quad i = 0, 1, 2, \ldots, N - 1$$

where $K_i$, $K_{1i}$, and $K2$ are fixed or variable-length bit strings, and $f(.)$ is a cryptographic one-way function that returns a bit string of length equal to or greater than $K_i$. If $f(.)$ returns a bit string of length greater than $K_i$, a truncation can be applied. A cryptographic hash function $H()$ (for example SHA-1 or MD5) can be used in place of $f(.)$ as follows:

$$K_i = f(K_{1i}, K2) = H(K_{1i} \parallel K2) \quad i = 0, 1, 2, \ldots, N - 1$$

The subkey $K_{1i}$ is defined as follows. Consider a binary tree with depth equal to $D + 1$, including the root node (level 0). At any level $h$, starting form 0, the binary tree has $2^h$ nodes. The last level is $D$, leading to $2^D = N$ leaves. Let’s indicate with $(h, j)$ the node $j$ of level $h$, with $0 \leq h \leq D$ and $0 \leq j \leq 2^h - 1$. Each node $(h, j)$ of the tree, excluding the last level $D$, has two child nodes that are respectively: left child $(h + 1, 2j)$ and right child $(h + 1, 2j + 1)$. Each node $(h, j)$ is associated to a value $x_{h,j}$ that is derived by the value of parent node as follows:

$$x_{h+1,2i} = f_0(x_h, i)$$
$$x_{h+1,2i+1} = f_1(x_h, i)$$

or equivalently:

$$x_{h,i} = \begin{cases} 
  f_0(x_{h-1,i/2}) & i = 0, 2, 4, \ldots, 2^h - 2 \\
  f_1(x_{h-1,(i-1)/2}) & i = 1, 3, 5, \ldots, 2^h - 1 
\end{cases}$$
where \( f_0() \) and \( f_1() \) are two different cryptographic one-way functions. They could be also defined based on the same function \( f() \) as follows:

\[
\begin{align*}
  f_0(x) &= f(x) \\
  f_1(x) &= f(x + 1)
\end{align*}
\]

In this case we can write \( x_{h,i} \) (recursively) as:

\[
x_{h,i} = f(x_{h-1,\lfloor i/2 \rfloor} + (i \mod 2))
\]

By repeatedly applying the previous equations, starting from the value \( x_{h,i} \) of node \((h,i)\) it is possible to generate all values associated to the nodes of the subgraph that has \((h,i)\) as root. At the same time the value \( x_{h,i} \) of node \((h,i)\) can be obtained from the value associated to any node along the path from the \( x_{h,i} \) to the tree’s root \((0,0)\).

Given such a binary tree, we define the subkey \( K_1 \) equal to the value of the leaf \( i \), that is:

\[
K_1_i = x_{D,i}
\]

Then, \( K_1 \) can be obtained from the value associated to any node along the path from the leaf \( i \) to the tree’s root, or equivalently from any values from \( x_{D,i} \) to \( x_{0,0} \). Figure 20 shows the \( K_1 \) subkeys derivation process described above.

At the same time, starting from the value \( x_{h,i} \) of node \((h,i)\) it is possible to obtain all subkey values in the interval from \( 2^{D-h \cdot i} \) to \( 2^{D-h \cdot (i+1)} - 1 \) included, that is all subkeys from \( K_{1,2^{D-h \cdot i}} \) to \( K_{1,2^{D-h \cdot (i+1)-1}} \). Note that, as a special case, the value \( x_{0,0} \) can generate all subkeys from \( K_1_0 \) to \( K_1_{N-1} \). Figure 21 shows how to obtain backward and forward secrecy by distributing the minimum set of values \( x_{h,i} \) that cover the time period of a member’s subscription.

This property can be used by the KDC to distribute the \( K_1 \) subkeys to new members in a very efficient way, reducing from \( O(N) \) to \( O(\log(N)) \) the number of values that the KDC has to pass to a new member in order to set subkeys for all the temporal period that the new member will belong to the group.
The worst case occurs when the node joins the group from time slot 1 to time slot \(N-1\), included. In this case, \(2 \cdot (\log_2(N) - 1)\) keys need to be distributed: \(x_{D,1}, x_{D,N-1}, x_{D-1,1},\ldots\).

Let’s now consider the subkey \(K2\). The value \(K2\) is maintained constant as far as only regular membership changes happen. As soon as an unpredictable leave event occurs, all unexpired keys of the leaving member must be revoked and replaced by new ones. This objective is reached by replacing the \(K2\) that in turn will change all successive group keys \(K_i\) that are generated by the values of \(K1_i\) and \(K2\).

When a new \(K2\) value is generated, this has to be distributed by the KDC to all remaining valid group members. This operation is very similar to the one faced by current centralized key distribution protocols: for example LKH [67] can be used.

**Managing keys for unlimited time intervals**

The described protocol assumes that the number of time slots is fixed and equal to \(n = 2^D\). Therefore, it is inevitable that the key distribution protocol is doomed to come to an end eventually. In this section, we sketch a simple extension of the protocol to allow the KDC to handle time intervals that might last beyond the one covered by a single tree (i.e., more than \(n\) time slots). The basic idea is to instantiate as many trees as required in order to manage a time interval of arbitrary length. This extension makes it possible to manage time intervals of any length with just a slightly increased computational effort and memory consumption.

Time is split into intervals \(I_k\) of length \(\Delta T\) and periods \(\Pi_i\) of length \(n \cdot \Delta T\). Each period \(\Pi_i\) is associated with a given seed \(s_i\), which is equal to the value of the root \(x_{0,0}^{i}\) of a tree. Within a given period, the protocol works exactly as described above. If a subscription lasts beyond the end of a period, it is necessary to distribute keys from more than one tree. Each tree can be computed “on the fly” in the following way:

\[
x_{0,0}^{i} = g(s_{i}) = g(h(s_{i-1}))
\]

where \(s\) is a seed, \(h(\cdot)\) is a “one-way” function (i.e. hashing function), and \(g(\cdot)\) is a “blinding”
function (i.e. XOR function). For instance, possible choices for \( h(.) \) and \( g(.) \) are

\[
h(s_i) = H(s_{i-1}) = H^{i+1}(s)\\
g(s_i) = s \oplus h(s_i) = s \oplus H^{i+1}(iv)
\]

where \( H \) is a hashing function and \( iv \) is an initial vector. Therefore:

\[
s_0 = s \oplus H(iv)\\
s_1 = s \oplus H(H(iv)) = s \oplus H^2(iv)\\
\ldots\\
s_i = s \oplus H^{i+1}(iv)
\]

The key \( x_{h,i} \) can be calculated as

\[
x_{h,i} = x_{h,I}^P = f\left(\underbrace{f(\ldots f(}^h \text{ times} \underbrace{(g(s_P-1) + a_0) + a_1) \ldots + a_{h-1})}_{h \text{ bits}}\right)
\]

where \( P = \lfloor i/2^h \rfloor \) is the index of the period, \( I = i \mod 2^h \) is the index of the period’s interval, and \( a_0, a_1, \ldots, a_{h-1} \) are the \( h \) bits of the binary representation of \( I \).

This mechanism makes it possible to extend the functioning of the protocol to cover an unlimited time interval without extra memory requirements as keys can be computed on the fly with no particular computational effort since operations like hashing and XORing are very lightweight.

### 3.5.3. Comparison of Different Key Distribution Strategies

In this section, the performance of the proposed key distribution protocol is compared with current state-of-the-art solutions, considering different application scenarios. The performance of the protocol is evaluated in terms of the following metrics:

- amount of cryptographic material to be sent to a particular node or group members (K1 and K2 sub-keys)
- number of messages to be sent within the group, either from the KDC or relayed by group members

Table 8 shows the performances of the protocol when a node joins the group, a node leaves the group, or a node is evicted from a group of size \( n \). The table shows the number of involved receivers and the amount of cryptographic material (values of the binary tree nodes to derive any K1 in the group subscription interval and K2 sub-keys) that each node must receive.

<table>
<thead>
<tr>
<th>Event</th>
<th>receivers</th>
<th>( x_{h,i} ) values per node</th>
<th>K2 sub-keys per node</th>
<th>total messages</th>
</tr>
</thead>
<tbody>
<tr>
<td>JOIN</td>
<td>1</td>
<td>( \leq 2 \cdot (\log_2(N) - 1) )</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>LEAVE</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>EVICTION</td>
<td>( n-1 )</td>
<td>0</td>
<td>1</td>
<td>( M )</td>
</tr>
</tbody>
</table>

Table 8: Performance of the key distribution protocol evaluated in terms of number of keys to distribute and messages to be sent.

Note that, only in case of node eviction, the new K2 sub-key must be sent to all other group members. The total number of messages (\( M \)) needed to update K2 depends on the
distribution strategy. A naive approach could be to send a separate message to each of the
\(n - 1\) members still being part of the group, but this method is clearly inefficient and can be
considered a worst-case example. In order to optimize network traffic, we superimpose the
LKH “user-oriented rekeying” strategy, which offers the best performance on the client side,
but increases the computational effort of the KDC. We found this solution to best adapt to IoT
scenarios, where devices typically offer little computational power and must minimize energy
consumption. In this case, \(M = (d - 1)(h - 1)\), where \(d\) is the LKH tree degree, and \(h\) is the
length of the longest directed path of the LKH tree.

Tables 9, 10, and 11 compare the performances of the LKH and MARKS protocols with our
key distribution protocol, in case of join, leave, and eviction events, respectively. Note that,
the super-imposed LKH strategy is necessary only if nodes are evicted, since the K1 sub-key
alone guarantees forward secrecy when nodes leave the group gracefully.

\[
\begin{array}{|c|c|c|}
\hline
\text{Event} & \text{keys to distribute} & \text{messages} \\
\hline
\text{LKH} & h(h + 1)/2 - 1 & h \\
\text{MARKS} & \leq 2 \cdot (\log_2(N) - 1) & 1 \\
\text{ours} & \leq 2 \cdot (\log_2(N) - 1) & 1 \\
\hline
\end{array}
\]

Table 9: Comparison with LKH and MARKS in case of node joining group of size \(n\).

\[
\begin{array}{|c|c|c|}
\hline
\text{Event} & \text{keys to distribute} & \text{messages} \\
\hline
\text{LKH} & (d - 1)h(h - 1)/2 & (d - 1)(h - 1) \\
\text{MARKS} & 0 & 0 \\
\text{ ours} & 0 & 0 \\
\hline
\end{array}
\]

Table 10: Comparison with LKH and MARKS in case of node leaving group of size \(n\).

\[
\begin{array}{|c|c|c|}
\hline
\text{Event} & \text{keys to distribute} & \text{messages} \\
\hline
\text{LKH} & (d - 1)h(h - 1)/2 & (d - 1)(h - 1) \\
\text{MARKS} & \text{undefined} & \text{undefined} \\
\text{ours} & (d - 1)h(h - 1)/2 + 1 & (d - 1)(h - 1) \\
\hline
\end{array}
\]

Table 11: Comparison with LKH and MARKS in case of node being evicted from group of size
\(n\).

As already said in Section 3.5.1, MARKS only works if the leaving time of a member is set
when the member joins the group and therefore it does not apply to the case of unpredictable
leave events, such as evictions.

The comparison between the new protocol, MARKS, and LKH shows that the former
achieves optimal performance, in terms of the metrics taken into account, for join events,
predictable leave events, and unpredictable leave events. In the case of join and predictable
leave events, the new protocol requires the same number of keys to be distributed and messages
to be sent as MARKS.

The new approach considers also the case of node eviction, in order to provide a thorough
description of all possible events that might occur in a group’s lifecycle. Node eviction is not
addressed by MARKS, while the proposed protocol manages this kind of events with the same
performance as LKH both for the number of keys to be distributed and the number of messages
to be sent.
Communication between KDC and group members is required only when nodes are evicted, thus communication overhead is kept to a minimum, thus ensuring optimal consumption of processing and network resources.
3.6. Distributed key verification and management

Two critical issues in Wireless Sensor Networks (WSNs) are key distribution and verification. They are mandatory to establish confidentiality in the network. Over the last decade, symmetric cryptography was, by far, the most popular primitive used for key distribution and certification [81, 82] to name a few since asymmetric cryptography was considered to be computationally too heavy to handle for low-power sensor nodes. However, at the end of the last decade, Elliptic Curve Cryptography (ECC) started to be deployed on sensor nodes [83], which showed that asymmetric cryptography is feasible in WSNs. Moreover, the IETF gave endorsement to ECC in several drafts [84].

If the cost of asymmetric cryptography is now no more an issue, there is still the problem of certifying the public keys of nodes. We cannot simply pre-install the public key of the Certificate Authority (CA) and the digital certificates that guarantee the authenticity of the public keys of nodes: any nodes that are certified by the same CA would potentially be able to communicate with each other, even if they are not supposed to be in the same network (e.g. your nodes would start communicating with your neighbor nodes, because they are produced and certified by the same manufacturer). A solution to this problem would be to allow the owner of the network to act as the CA and pre-install within each of the nodes the necessary cryptographic material. However, this solution would not scale up for a large number of nodes. Moreover, if the only way to install the cryptographic material within nodes is through the wireless medium, because they are located in places that are inaccessible (e.g. embedded inside walls), an attacker may initiate a Man-in-the-Middle (MitM) attack and swap the public key and the certificates of the owner with its own keys. Our contribution is to solve this problem by proposing a distributed key certification protocol based on one-way accumulators that is fully autonomous and does not require the use of CAs and certificates.

Accumulators are space/time efficient data structures used to test if an element belongs to a predefined set. Several accumulator designs are available ranging from RSA-based accumulators [85] to Secure Bloom filters [86]. We have studied and evaluated the different designs by implementing them on the Sun SPOT platform to determine which one is the most practical for our protocol in terms of processing speed, memory footprint, data transmission, and energy efficiency. We observe that our protocol has the best overall performance with ECC-based accumulators.

All the hypothesis for sensor nodes and the adversary are defined in Section 3.6.1. The accumulators, as well as our protocol for distributed key verification, are described in Section 3.6.2. A brief security analysis of our protocol is presented in Section 3.6.3. The performance evaluation and the impact of the choice of accumulators on performance are discussed in Section 3.6.4. Section 3.6.5 reviews the related work, Finally, Section 3.6.6 concludes our work and proposes future research directions.

3.6.1. Assumptions

In this work, we focus on securing large multi-hop networks that may be deployed alongside other unrelated networks. Wireless sensor nodes are low-power embedded devices. A gateway is used to collect data and/or relay traffic from nodes that may be placed at any arbitrary position and even may be mobile. A primary concern for a node manufacturer, the network administrator as well as for the user of applications running on such networks is twofold: first, to be able to distinguish between different legitimate networks to prevent unwanted information leakage or unwillingly transiting external traffic (i.e. that may waste energy), and second, to prevent attackers from joining or tempering with a network in which they are unauthorized.
Node Model

We assume that nodes are interface-less: no physical interaction with a node is possible and the only way to communicate with them is through the wireless medium. This assumption is realistic considering the cases of the Intelligent Building and the Internet of Things, in which nodes might be, for example, embedded inside walls. Manufacturers also tend to integrate various sensors into a single chip (SoC) so that they are easier to miniaturize, mass-produce, and deploy in a non-invasive way. We assume that nodes are manufactured and distributed in batches by the manufacturer. This is a reasonable assumption considering that manufacturers would probably not sell cheap, mass-produced nodes individually, and that at least a few nodes are needed to build a practical multi-hop WSN. We also consider that customizing nodes with specific cryptographic material at the production time is not much of a limitation. This process is already quite common for certain chip manufacturing in which unique identifiers or calibration data are loaded after the production.

Attacker Model

Our protocol is designed to deal with three particular threats: Node injection, Node capture and Denial-of-Service (DoS).

Node injection – The adversary must not be able to inject its own node in the network.

Node capture – The adversary can gain full control of a node since we assume that the nodes are not tamper-resistant. This implies that all the node secrets are exposed to the adversary. To measure the resistance of a protocol, the number of communications that can be eavesdropped by the adversary, after the node capture, is considered [87]. It does not include the communications of the compromised node.

Denial-of-Service (DoS) – The adversary may attempt to exhaust the resources of nodes by sending bogus messages or requests. The protocol must protect all the computationally intensive operations: the adversary must not be able to trigger them easily.

We have considered that two attacks were outside the scope of our work. The Relay attack is the most basic form of a MitM attack. It can be used to mount more sophisticated attacks such as Wormhole attacks or DoS. Gollakota et al. considered this class of attacks [88]. In Replication attacks, the adversary injects nodes in the network. The nodes are copies of a legitimate node. Specific solutions to this problem can be found elsewhere [89].

In the next section, we detail how accumulators can be used for key distribution and verification in WSNs.

3.6.2. Accumulator-based Protocol

After reviewing the principles of different one-way accumulators, we describe how to use them to verify keys in the context of sensor networks.

One-Way Accumulators

One-way accumulators are authenticated data structures similar to Bloom filters [90]. They were introduced by Benaloh and de Mare [85] under the name Accumulated Hashing. The purpose of an accumulator is to allow a user to learn if an item belongs or not to a given set. This probabilistic data structure is known to be efficient in time and space at the cost of a probability of false positives.

Let us define two sets $\mathcal{A}$ and $\mathcal{B}$. $\mathcal{A}$ is the set of the accumulator and $\mathcal{B}$ is the set of the items to be accumulated. An accumulator for set $\mathcal{S} \subseteq \mathcal{B}$ of $i$ items $v_i$ is denoted as $z^i$. Accumulators are based on commutative functions.
Definition A function $F : A \times B \rightarrow A$ is said to be commutative (or quasi-commutative) if: $F(F(a, b), c) = F(F(a, c), b), \forall a \in A$ and $b, c \in B$.

All the accumulators implement the following components:

Key-Gen($s, K_{priv}, f$) – Given a security parameter, $s$, master key, $K_{priv}$, and key derivation algorithm $f$, it generates all the key materials needed.

Build-Acc($z^0, v_1, \ldots, v_m$) – From a commutative function, a seed $z^0 \in A$ and $m$ items $v_i \in B$, value $z^m$ is computed recursively:

$z^i = F(z^{i-1}, v_i)$, $i$ in $1, \ldots, m$.

Gen-Wit – The witness of membership $w_i$ is a value associated with each value $v_i$ accumulated in $z$. The witness is used during the verification of $v_i$.

Authenticate($z^m, v_i$) – It verifies if item $v_i$ belongs to accumulator $z^m$. When a witness is needed, Authenticate verifies if the following equality is satisfied: $z^m = F(w_i, v_i)$.

Throughout this section, notation $z$ is used for simplicity instead of $z^i$.

Asymmetric Accumulators

In the original paper $[85]$, Benaloh et al. suggested to use the modular exponentiation as a quasi-commutative function to create a one-way accumulator: $F(a, b) = a^b \mod n$, with $a \in A$ and $b \in B$. For an appropriate choice of $n$, this function is one-way as long as the RSA assumption holds. The accumulator of Benaloh et al. works as follows:

Gen - This function generates all the values needed by function $F$. Let us define $n = pq$ as the product of two safe primes $p, q$, of approximately the same size. Further details on the security issues concerning the choice of $n$ can be found in the original paper. We only review its main ideas in this section.

Build-Acc - seed $z^0$ is also needed to bootstrap the accumulator. Therefore, value $z^1$ is computed recursively: $z^1 = F(z^{0-1}, v_1)$.

Gen-Wit - we associate witness $w_i$ with each value $v_i$ belonging to accumulator $z$. Witness $w_i$, corresponding to item $v_i$, is a partial accumulator that includes all the values in $z$ except $v_i$.

Authenticate - value $v_i$ that belongs to $z$ verifies the following equation:

$F(w_i, v_i) = w_i^{v_i} \mod n,$

$= (z^0)^{\prod_{j=1}^{m} v_j} \mod n,$

$= z.$

Other quasi-commutative functions have been proposed $[91, 92]$. The scalar-point product over elliptic curves is a natural choice. Let us denote scalar $a$ and $P$, $Q$ as two points of an elliptic curve such that: $Q = aP$. An ECC-based accumulator works similarly to the original accumulator proposed by Benaloh et al. The Elliptic Curve Discrete Logarithmic Problem guarantees that it is computationally infeasible to deduce $a$ from the knowledge of $P$ and $Q$ $[93]$.

Symmetric Accumulators

A symmetric-key equivalent of Benaloh accumulators appeared at FSE 1996. Based on cryptographic hash functions, Nyberg accumulators $[94]$ have a much simpler Authenticate function, because it does not require partial accumulators. The connection between Nyberg accumulators and Bloom filters was later made by Yum et al. $[95]$ who also demonstrated that Bloom filters are better cryptographic accumulators than Nyberg accumulators in terms of the minimal false positive rate. In parallel and independently of the Nyberg’s results, Cryptographic/Secure Bloom filters were proposed $[86]$ with applications to private information retrieval in databases.
A symmetric accumulator is a $\ell$-bit vector, $z = (z_1, \cdots, z_\ell)$ with $z_i \in \mathbb{F}_2$. The support of an $\ell$-bit vector denoted $	ext{supp}(z)$, is the set of the non-zero coordinate indexes: $	ext{supp}(z) = \{ i \in [1, \ell], z_i \neq 0 \}$.

**Build-Acc**$(z^0, v_1, \cdots, v_m)$ – The creation of the accumulator is done for $m$ items (keys) using the following recursion: $z^i = z^{i-1} \lor g(v_i)$, $i$ in $1, \cdots, m$ with $\lor$ the bitwise inclusive-or operator and $g$ is a function from $B$ to $A$. We will not give more details on this function since it varies according to the accumulator design.

**Authenticate**$(z, v_i)$ – We can observe from the previous equation that

\[
\text{supp}(z) = \text{supp}(z^{m-1}) \cup \text{supp}(g(v_i)).
\]

Therefore, item $v_i \in B$ belongs to $z$ if: $\text{supp}(g(v_i)) \subset \text{supp}(z)$.

For Bloom filters, the **Key-Gen** function must generate $k$ secret keys $K_i$ from secret $s$. If $H$ is a cryptographic hash function that can be keyed, function $g$ is defined by

\[
\text{supp}(g(v_i)) = \{ H(K_i, v_i), i \in 1, \cdots, k \}.
\]

The differences between Bloom filters and Secure Bloom filters are the following: a) the use of HMAC-SHA-1 as a hash function and b) the use of implementation trade-offs to reduce the number of computed hashes [96]. It is worth mentioning that many libraries use cryptographic hash functions for Bloom filters instead of universal hash functions.

**False Positive Probability**

Since accumulators are probabilistic data structures, they have to be carefully designed to control their false positive probability. Bari and Pfitzmann have shown [91] how to reduce the false positive problem to the strong RSA assumption under certain conditions. Benaloh et al. have shown that the false positive probability is negligible for $|n| \geq 1024$ bits. The same can be said about accumulators based on the 160-bit prime field ECC. For Bloom filters, false positive probability $p$ is [90]:

\[
\left( 1 - \left[ 1 - \frac{1}{\ell} \right]^{km} \right)^k.
\]

**Key Verification Protocol**

The protocol described in this section is based on three functions: **initialization**, **ownership transfer**, and **node-to-node key verification**.

**Initialization** – The manufacturer assigns to each node $N_i$ a pair of public/private keys $(PK_{N_i}, SK_{N_i})$. All the node public keys $PK_{N_i}$ belonging to the network are all accumulated in $z$. In addition, the manufacturer generates a public/private key pair for the gateway and includes the public key of the gateway in accumulator $z$ (i.e. the gateway is considered as any other node). The manufacturer stores $z$ in every node. In addition, every node is set up with all the values needed for the accumulator including witness $w_i$.

**Ownership transfer** – The final user of the network uses the gateway to manage and monitor the network. To transfer the ownership, the manufacturer provides the gateway public/private key pair to the final user. We assume that at a given time, there is only one owner. The manufacturer needs to solve two issues to transfer the ownership to the final user: a) the gateway must be able to distinguish the nodes that belong to the network and b) the nodes must recognize the gateway public key. The first issue is solved by transferring $z$ and all the necessary parameters from the manufacturer to the gateway. The gateway can then execute the Authenticate function as any node. The second problem is solved by including the gateway public key in $z$. 
Node-to-node key verification – As shown in Figure 22, nodes exchange their public keys $PK_{N_i}$ with corresponding witnesses $w_i$. They then execute the Verify function described in Algorithm 8. When a node determines that a public key belongs to the accumulator, it stores it as an entry in table $T$. The Lookup$(T, PK_{N_i})$ function verifies if the node has already verified the public key of node $N_i$. The Put$(T, PK_{N_i})$ function adds the public key of the node to the table. Once the nodes have mutually verified their public keys with Verify, they can establish a symmetric key through the Elliptic Curve Diffie-Hellman (ECDH) key agreement protocol [97].

**Algorithm 8 Verify$(z, PK_{N_i}, w_i)$ function ($N_1$ by $N_2$).**

```plaintext
if Lookup$(T, PK_{N_i}) = false$ then
  if Authenticate$(z, PK_{N_i}, w_i) = true$ then
    Put$(T, PK_{N_i})$
  end if
else
  Do nothing (key already verified)
end if
```

### 3.6.3. Security Analysis

Our protocol inherits the major security properties of one-way accumulators: “one-way-ness” and resistance to forgery [85] [91]:

**Node injection** – The first attempt of the adversary might be to send its own public key and witness. In this case, the security of our protocol is reduced to the security of the accumulator.

**Node capture** – Due to the “one-way-ness” of accumulators, the capture of a node does not compromise the communications of other nodes: only the keys of the captured node are compromised.

**Denial-of-Service (DoS)** – The goal of the adversary is to make the nodes waste precious resources (e.g. energy). To achieve this goal, computationally expensive operations are triggered. We assume that the most expensive operation is ECDH and our protocol prevents an adversary to trigger useless ECDH computations. Lookup table $T$ prevents the replay of correct messages that cause the exhaustion of the node resources.

Please note that since the manufacturer produces each batch of nodes with distinct pre-installed accumulators, the problem of “promiscuous” connections between certified nodes that should not belong in the same network, as well as the MiTM attack during the wireless pre-installation of cryptographic material that were mentioned at the beginning become irrelevant.
3.6.4. Performance Evaluation

We have implemented the one-way accumulators of Benaloh et al. (we will call them RSA and ECC-based accumulators) and Secure Bloom filter (hereafter simply referred to as Bloom filter) in Java ME on Sun SPOTs. Each Sun SPOT includes a 180 MHz 32-bit ARM920T processor, 512 kB RAM, 4 MB flash memory, and a 3.7V rechargeable 750 mAh lithium-ion battery. It has an integrated TI CC2420 radio chip operating in the 2.4 GHz band and compliant with the IEEE 802.15.4 standard. Sun SPOTs also provide a readily available security API. The elements accumulated in our implementation are 160-bit Elliptic Curve (EC) public keys using the secp160r1 curve domain parameters. We have used SHA-1 as the hashing algorithm that generates 160-bit hash codes.

Once we have implemented all different accumulators, we have measured the time the various computations and transmissions within our protocol take to evaluate the total time required for a node-to-node key verification with Sun SPOTs. The entire node-to-node key verification between two Sun SPOTs (let us call them SPOT 1 and SPOT 2) using asymmetric accumulators (RSA-based and ECC-based accumulators) can be divided into four steps:

1. SPOT 1 sends its public key $PK_1$ and its witness $w_1$ to SPOT 2.
2. SPOT 2 executes $\text{Verify}(z, PK_1, w_1)$.
3. SPOT 2 sends its public key $PK_2$ and its witness $w_2$ to SPOT 1.
4. SPOT 1 executes $\text{Verify}(z, PK_2, w_2)$.

The node-to-node key verification using Bloom filters would only differ by the lack of witnesses in each step. We can see that steps 3 and 4 are simply a repetition of steps 1 and 2. Therefore, we have implemented steps 1 and 2 to be run between two SPOTs and we have measured the duration of each step. The results obtained for each step with the implementations based on asymmetric accumulators and Bloom filters are shown in Table [13a]. The time to perform a node-to-node key verification between two SPOTs is obtained by multiplying the time of two steps by two (e.g. 1.4 x 2 = 2.8 seconds for the key verification using Bloom filters). For each measurement, one thousand 160-bit EC public keys have been accumulated. The parameter values used for different accumulator implementations are presented in Table [12]. They ensure that all the implementations have an equivalent level of security.

Table 12: Parameter values used for the accumulators.

<table>
<thead>
<tr>
<th>Accumulator</th>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>RSA-based</td>
<td>Length of $n$</td>
<td>1024 bit</td>
</tr>
<tr>
<td>ECC-based</td>
<td>Curve domain parameters</td>
<td>secp160r1</td>
</tr>
<tr>
<td>Bloom</td>
<td>Value of $p$</td>
<td>$2^{-80}$</td>
</tr>
</tbody>
</table>

The time that step 2 takes mostly consists of the execution time of the Authenticate function in Verify (see Algorithm [5]). We should also note that, although the time that step 2 takes when using Bloom filters is slightly less than when using ECC-based accumulators, it becomes equivalent or longer once $p$ is set to $2^{-90}$ or less.
Table 13: Duration and battery consumption of the node-to-node key verification.

(a) Duration of the first two steps in the node-to-node key verification.

<table>
<thead>
<tr>
<th>Step</th>
<th>RSA</th>
<th>ECC</th>
<th>Bloom</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.1</td>
<td>1.1</td>
<td>1.0</td>
</tr>
<tr>
<td>2</td>
<td>4.4</td>
<td>0.5</td>
<td>0.4</td>
</tr>
<tr>
<td>TOTAL</td>
<td>5.5</td>
<td>1.6</td>
<td>1.4</td>
</tr>
</tbody>
</table>

(b) Energy consumption during the node-to-node key verification.

<table>
<thead>
<tr>
<th>Step</th>
<th>RSA</th>
<th>ECC</th>
<th>Bloom</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.029</td>
<td>0.028</td>
<td>0.027</td>
</tr>
<tr>
<td>2</td>
<td>0.033</td>
<td>0.007</td>
<td>0.006</td>
</tr>
<tr>
<td>3</td>
<td>0.015</td>
<td>0.013</td>
<td>0.013</td>
</tr>
<tr>
<td>4</td>
<td>0.099</td>
<td>0.011</td>
<td>0.009</td>
</tr>
<tr>
<td>TOTAL</td>
<td>0.176</td>
<td>0.059</td>
<td>0.055</td>
</tr>
</tbody>
</table>

We have also estimated the energy consumption on the Sun SPOT based on the duration of each steps for the node-to-node key verification (cf. Table 13b). The most “energy-efficient” key verifications are the ones using Bloom filters and ECC-based accumulators. The extra energy that the key verification using ECC-based accumulators requires from step 1 to 3 compared with the ones using Bloom filters is due to the additional exchange of witnesses. However, the amount of the consumed energy in step 4 of the key verifications using Bloom filters would grow as the execution time of the Verify function increases due to a lower $p$.

Table 14: Memory footprint of the node-to-node key verification.

(a) Size of the variables employed during the node-to-node key verification.

<table>
<thead>
<tr>
<th>Instance</th>
<th>Size (bytes)</th>
<th>RSA</th>
<th>ECC</th>
<th>Bloom</th>
</tr>
</thead>
<tbody>
<tr>
<td>PK</td>
<td></td>
<td>1338</td>
<td>1338</td>
<td>1338</td>
</tr>
<tr>
<td>SK</td>
<td></td>
<td>1261</td>
<td>1261</td>
<td>1261</td>
</tr>
<tr>
<td>$z$</td>
<td></td>
<td>128</td>
<td>48</td>
<td>1427</td>
</tr>
<tr>
<td>$w$</td>
<td></td>
<td>128</td>
<td>48</td>
<td>N/A</td>
</tr>
<tr>
<td>TOTAL</td>
<td></td>
<td>2855</td>
<td>2695</td>
<td>17026</td>
</tr>
</tbody>
</table>

(b) Memory usage during the node-to-node key verification.

<table>
<thead>
<tr>
<th>Step</th>
<th>Size (kB)</th>
<th>RSA</th>
<th>ECC</th>
<th>Bloom</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>79</td>
<td>78</td>
<td>78</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>53</td>
<td>43</td>
<td>36</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>3.1</td>
<td>3.6</td>
<td>1.6</td>
</tr>
<tr>
<td>TOTAL</td>
<td></td>
<td>135</td>
<td>125</td>
<td>116</td>
</tr>
</tbody>
</table>

Table 14a shows the size of the public/private keys, accumulators, and witnesses used in the node-to-node key verification. We can obviously notice that much less memory is required to store asymmetric accumulators than Bloom filters. The advantage of using an ECC-based accumulator is apparent here, since it is only 1/150 of the size of a Bloom filter. Additionally, the size of Bloom filters rapidly increases as more elements are accumulated since they need to increase the number of bits per element to maintain their false positive probability. Since the size of asymmetric accumulators is constant regardless of the number of accumulated elements, we can conclude that asymmetric accumulators have much better scalability in terms of the accumulator size compared to Bloom filters. From the memory usage presented in Table 14b, we can see that two most “memory-efficient” key verification implementations are the ones using Bloom filters and ECC-based accumulators. However, the memory required in step 4 of the key verification using Bloom filters would increase if the number of accumulated keys increases or if $p$ is set to a lower value.
3.6.5. Related Work

Key establishment and distribution are critical problems in WSNs that were already deeply investigated by the community. Two approaches are in competition: the symmetric and asymmetric cryptography. Our work naturally belongs to the latter. Below, we only briefly review and compare the work based on the asymmetric cryptography. We encourage readers who are interested in further details on the work based on the symmetric cryptography to consult the literature [98].

The asymmetric cryptography was, for a while, considered infeasible for sensor nodes especially for RSA-based cryptosystems. TinyPK [99] was the first attempt to implement a public-key infrastructure in WSNs. It uses the classical Diffie-Hellman protocol and a CA to certify the keys. The performance results were relatively modest and not promising for a practical application. Since then, TinyECC [83] and NanoECC [100] libraries proved the feasibility and the efficiency of the Elliptic Curve Cryptography on sensor nodes. Low-cost hardware extensions have also been demonstrated [101]. The Elliptic Curve Diffie-Hellmann protocol has been put into practice in WSNs [102][103]. Meulenaer et al. [102] proposed to use a trusted third party to certify the keys in a way similar to Kerberos. Sun et al. [103] used a “self-certified” ECDH protocol along with a polynomial-based weak authentication scheme to thwart DoS attacks initiated by bogus ECDH requests. Our approach is a natural extension of these efforts: accumulators eliminate the need for CAs and certificates during key establishment.

3.6.6. Conclusion and Future Work

We have proposed a key verification protocol that does not require CAs and certificates. Under the protocol, nodes can autonomously verify the authenticity of public keys they exchange using one-way accumulators. The performance results obtained for our implementations of the key verification schemes show that the exchange and verification of public keys between two wireless sensor nodes can be performed within few seconds while consuming little energy. Our analysis of the existing accumulators lets us conclude that ECC-based accumulators are the best suited for our protocol since they perform as well as Bloom filters while having a memory footprint that does not increase with a greater number of accumulated elements.

Our next immediate objective is to study how our protocol can fit into the current wireless sensor network protocol stacks (802.15.4, RPL etc.). Although many nodes produced nowadays are as powerful as Sun SPOTs (e.g. STM32W107) albeit without a Java Virtual Machine), our protocol should also be tested on nodes with a less processing power. Another issue to address is that our protocol cannot safely add and remove nodes from the network. Using dynamic accumulators [104] that allows adding and removing elements may be a solution to this problem, but security flaws have been found in these designs [105].
4. Conclusions

This deliverable presents the results obtained in Task 5.3 and shows the relation between the technical work achieved and the CALIPSO application scenarios. The presented contributions are parts of the whole CALIPSO protocol stack, whose integration and validation will be delivered in D6.61 at month M26. A subset of the contribution modules presented in this document will be selected for evaluation in the project field trials (D6.62 at month M30).

References


