INSTITUT FÜR INFORMATIK
DER LUDWIG–MAXIMILIANS–UNIVERSITÄT MÜNCHEN

Master's Thesis

Deriving and Distributing Static Compression Context for LPWANs via IKEv2

Sebastian Halder

INSTITUT FÜR INFORMATIK
DER LUDWIG–MAXIMILIANS–UNIVERSITÄT MÜNCHEN

Master's Thesis

Deriving and Distributing Static Compression Context for LPWANs via IKEv2

Sebastian Halder

Aufgabensteller:	Prof. Dr. Dieter Kranzlmüller
Betreuer:	Tobias Guggemos
Abgabetermin:	27. Juli 2020

Abstract

Header compression is a way to adapt existing network protocols for use in IoT (Internet of Things) scenarios. The motivation is the limited network bandwidth and energy budget of IoT devices. Specifically, default IPv6 requires the maximum transmission unit (MTU) to be at least 1280 bytes, which is not feasible for many of the networking devices used in IoT. Existing header compression protocols use prior knowledge (context) and/or approximations to encapsulate and compress header data of common network protocols. Static context header compression (SCHC) is one such protocol and uses exclusively prior knowledge (static context) to elide or compress header fields. A major shortcoming of SCHC is the lack of mechanics to distribute this static context between communication partners as it relies solely on manual configuration.

This work examines the possibility to combine SCHC with IPSec and use its key exchange protocol (IKEv2) to identify and distribute context information. The concept introduced in this work is able to leverage the power of header compression to reduce the overhead created by the protocols in the IP stack without relying on the manual configuration of SCHC. To this goal, we develop a method to derive static compression context for SCHC from the context information stored in the IPsec Security Associations. The proof of concept demonstrates some of the major obstacles that have to be overcome to successfully integrate SCHC processing into the IPsec suite.

Contents

1. Introduction **1**

2. Background **3**
- 2.1. Internet of Things . . . 3
- 2.2. LPWANs . . . 4
- 2.3. IPv6 . . . 5
- 2.4. Internet Protocol Security . . . 5
  - 2.4.1. ESP . . . 7
  - 2.4.2. IPsec Databases . . . 7
  - 2.4.3. IKEv2 . . . 9
- 2.5. RIOT OS . . . 10

3. Related work **11**
- 3.1. Static Context Header Compression . . . 11
  - 3.1.1. SCHC Rules . . . 12
  - 3.1.2. Matching Operators and C/D Actions . . . 13
- 3.2. ESP Header Compression (EHC) and Diet-ESP . . . 14
  - 3.2.1. ESP Processing . . . 14
  - 3.2.2. EHC Context . . . 15
  - 3.2.3. EHC Rules . . . 15
  - 3.2.4. Diet-ESP Strategy . . . 16
  - 3.2.5. IKE Extension . . . 17

4. Concept **19**
- 4.1. Design Goals . . . 19
- 4.2. Simplified Topology . . . 19
- 4.3. Protocol Stack . . . 20
- 4.4. IPsec Use Cases . . . 20
- 4.5. Packet Processing . . . 21
  - 4.5.1. Example . . . 21
  - 4.5.2. Generalized Processing . . . 23
- 4.6. Deriving SCHC Rules . . . 23
  - 4.6.1. Context Parameters . . . 23
  - 4.6.2. Rule Creation . . . 25
  - 4.6.3. Rule Downgrading . . . 28
  - 4.6.4. Padding Compression . . . 29
- 4.7. Operation . . . 29
  - 4.7.1. Modes of Operation . . . 30
  - 4.7.2. Initiating Compression . . . 31
  - 4.7.3. Identifying Network Roles . . . 31

4.7.4. End-to-end Encryption 31
4.7.5. Tunnel mode 32
4.7.6. Rule Creation Process 33
4.7.7. Differences between Oirignal and Decompressed Packet 34
4.8. IKEv2 Compression 34
4.8.1. Using SCHC 34
4.8.2. Using Common Algorithms 37

5. Implementation **39**
5.1. Testing Environment 39
5.1.1. Components 39
5.1.2. RIOT native 39
5.2. IKEv2 for RIOT OS 40
5.2.1. Initial Exchanges 40
5.2.2. Deriving Keys 42
5.3. ESP for RIOT OS 43
5.3.1. SA and Key Management 44
5.3.2. ESP Processing 45
5.4. SCHC for RIOT OS 46
5.4.1. Header Parser 46
5.4.2. Rule Manager 47
5.4.3. Bit-Buffer 47
5.4.4. Compressor 47
5.4.5. Decompressor 48
5.4.6. SCHC Processing 48
5.4.7. Compute CDA 49
5.5. Proof of Concept 50
5.5.1. SCHC Integration 50
5.5.2. SCHC Processing 52
5.5.3. Example 53

6. Evaluation **57**
6.1. Compression Efficiency 57
6.1.1. Strict-mode 58
6.1.2. Preset-mode 60
6.1.3. Tunnel Mode 60
6.1.4. Summary 61
6.1.5. IKE Overhead 61
6.2. End-to-end Setup 62
6.2.1. Compression Efficiency 62
6.2.2. Alternatives 63
6.3. Design Goals 63
6.4. Layer 2 Protocol Choices 64
6.5. Caveats and Limitations 64

7. Conclusion **67**

Appendices **69**

A. Sample Rule **71**

B. Rule Templates **73**
B.1. Strict-mode 73
B.2. Preset-mode 74

List of Figures **77**

List of Tables **79**

Listings **81**

Bibliography **83**

1. Introduction

The Internet of Things (IoT) sees technological advancements on a regular basis [GG16] [Ray20]. One of the more recent additions is the advent of Low-Power Wide Area (LPWA) technologies which extend the communication range for IoT devices to multiple kilometres [Sem19]. Using LPWA technologies, devices face even harsher restrictions on energy budget and network bandwidth than with some of the other IoT technologies [Far18]. To contend with these restrictions, developers and manufacturers have designed highly optimized low-level communication protocols (e.g. LoRa [Sem19], Sigfox [Sig]). In an ongoing effort to increase interoperability and compatibility between IoT devices, researchers strive to integrate the established IP protocol stack with those LPWA technologies. One of the main obstacles is the limited (radio) frame size for LPWA technologies, which can in some cases only carry tens of octets in data [Sem19]). As a solution, header compression algorithms such as Static Context Header Compression (SCHC) [MTG^{+}20] are employed to reduce the traffic overhead introduced by the communication protocols. However, while SCHC is able to provide compression for common protocols (e.g. IPv6, UDP, CoAP) it requires manual configuration to operate. The aim of this work is to identify options to reduce the configuration cost of SCHC for deployment in LPWA networks. In addition to the integration of the IP stack, ensuring data integrity and confidentiality in IoT traffic is another major goal in LPWA development. However, due to the additional protocols involved and the context information required to initiate encryption, this presents a considerable network overhead. Since this work combines Internet Protocol Security (IPsec) with SCHC, we aim to combine protocol security and header compression within a single concept. This way, we can take advantage of the context information required to initiate encryption by simultaneously using the information to reduce the traffic overhead for most of the protocol stack.

The main contribution of this work is the development of a concept to derive context information for Static Context Header Compression (SCHC) from IPsec Security Associations (SAs) and the development of a prototype to demonstrate the implementational requirements for integrating SCHC compression into the IPsec suite. Based on the findings by Migault et al. [MGBS19], we define parameters which describe context information in the IPsec Security Association and can be leveraged to compress the traffic flow associated with the SA. Using these parameters we are able to generate templates for a set of compression rules that can be quickly adapted to fit a specific IPsec context. Furthermore, we aim to identify additional methods to improve the efficiency of the compression and to reduce the overhead created by IPsec. With the proof of concept we assist in identifying and addressing the basic requirements for integrating SCHC compression into regular IPsec protected traffic. To assess the efficiency of the compression based on the context parameters derived from IPsec, we perform a best case/worst case analysis and match the findings to the known frame size restrictions for LoRa.

The remainder of this work is organized as follows: In Chapter 2 we introduce the basic underlying concepts of this thesis. Chapter 3 introduces related work and the building blocks (SCHC and EHC) that lay the foundation for the rest of this work. In Chapter 4 we establish our design goals and specify the characteristic of the target infrastructure. Subsequently, the specifics of our concept are developed and refined to fit these goals and characteristics. In Chapter 5 the prototype implementations for the required sub-modules (IKE, ESP and SCHC) are introduced and the specifications for the proof of concept are detailed. Chapter 6 evaluates the compression efficiency and design of the concept and recognizes its limitations.

2. Background

This chapter provides an overview of the fundamental architectures and protocols related to this work. In the first part, we introduce the Internet of Things (IoT) and one of the leading network infrastructures employed in this context. In the later parts, we introduce the Internet Protocol Security (IPsec) extension for IPv6 and present the primary operating system used in this work.

2.1. Internet of Things

The Internet of Things (IoT) has been described as a new concept that deals with pervasive things or objects which interact and cooperate with each other to reach common goals [AIM10]. From a more basic point of view, the IoT encompasses interacting devices that exist in our everyday life, many of which are not typically recognized as such and are therefore referred to as 'things or objects'. Examples for those objects are RFID tags, smart phones as well as different sensors and actuators. Those IoT devices find application in various domains ranging from smart home and infrastructure to mobile ticketing or health monitoring.

The IoT is often characterized by a highly distributed network of wireless devices that present a very distinct set of challenges.

Challenges

The challenges described in this section are only a small subset and represent the main problems that need to be accounted for in the scope of this work [BEK14].

- Power constraints: The development of the IoT has been enabled by technological advances that allow the reduction of size, weight and energy consumption of electronic devices [AIM10]. Nevertheless power consumption still plays a major role in the selection and practicability of IoT devices for their intended use. To reduce power consumption many IoT devices introduce sleep cycles during which the devices power down partially and usually cease network operation.

- Wireless communication: Since the IoT often consists of heterogeneous wireless networks, hardware and communication protocols need to facilitate the transmission of information between devices that originate not only from different manufacturers but also vary widely in their size and capability.

- Bandwidth constraints: Coinciding with the restraints imposed on power consumption the network bandwidth for many IoT devices is limited. This can originate from limited computational power and working memory, government regulated restrictions on duty cycles, high power consumption in wireless communication, limitations of wireless technologies or others.

- Security and privacy: Security remains one of the key issues when developing applications for the IoT [XYWV12]. Common solutions involve encryption and authentication of network traffic.

2.2. LPWANs

The term Low-Power Wide Area (LPWA) refers to wireless transmission technologies that provide low power, long range and low cost communication [MBCM19]. Prior to the inception of LPWA technologies, IoT applications were often limited to using cellular communications (eg. 2G, 4G) or resorted to multi-hop strategies using short range technologies such as Bluetooth [CVZZ16]. Due to their low energy cost devices operating in Low-Power Wide Area Networks (LPWANs) outperform cellular based communications when comparing the devices' battery lives, while still maintaining a range of up to 10-15km in rural areas and 1-5km in urban areas [Sem19]. This has primarily been achieved by increasing receiver sensitivity [CVZZ16]. While multi-hop short-range transmission technologies rival LPWANs in power consumption, the former are poorly suited for scenarios that require urban-wide coverage and have drastically higher network complexity.

Most LPWANs operate in unlicensed ISM bands which are, based on the region of operation, centred at 868/915 MHz, 433 MHz, and 169 MHz. To allow long-range communication LPWANs usually provide significantly lower bitrates compared to short-range technologies [CVZZ16]. This is however often outweighed by the low energy cost and comparably high range of LPWAN devices. On top of that, most IoT applications (e.g. sensors) don't require high network bandwidths but instead transmit data sporadically in small chunks over large amounts of time.

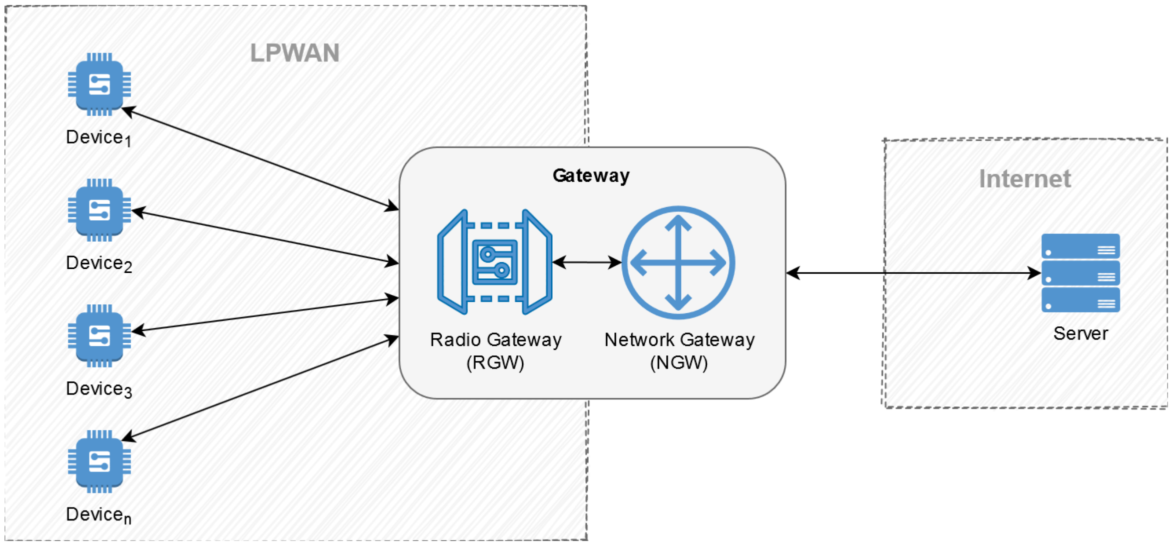

Figure 2.1.: Typical LPWA-Network arranged in a star topology. Multiple IoT devices(left) connect through a gateway(centre) to a central server(right).

In contrast to multi-hop based short-range networks (e.g. mesh), LPWANs are usually organized in a star topology as shown in Figure 2.1, consisting of several smaller IoT devices that establish a wireless connection to a central node called gateway [PW17]. The gateway provides bridging to other networks or the internet and could even be connected to the power grid and a wire-based network. This allows the operation of various small and distributed network devices (e.g. sensor nodes) in difficult terrain while maintaining control over the connection latency. The Gateway itself is comprised of two components: the Radio Gateway (RGW) and the Network Gateway (NGW). The RGW provides a network interface using a LPWA technology, while the NGW usually utilizes Ethernet to allow it to connect to a server over the internet. RGW and NGW are typically connected via a network bridge that relays traffic between the two interfaces.

LoRa

LoRa is a radio frequency (RF) modulation technology that uses a spread spectrum technique derived from existing Chirp Spread Spectrum(CSS) and is designed and patented by Semtech Corporation [Sel17]. The system operates in the unlicensed ISM Bands at 868 Mhz in Europe, 915 Mhz in North America and 433 Mhz in Asia [Sem19]. Using the six spreading factors SF7 through SF12 LoRa can achieve a trade-off between data rate and signal range. LoRa is purely a physical layer implementation and is designed to be used with the open standard LoRaWAN communication protocol.

In the context of this work, only the physical layer implementation (LoRa) is used. The reasoning for this is based on the lack of flexibility when combining LoRaWAN with compression and IP based encryption and is further elaborated upon in Section Section 6.4. LoRa, as opposed to competitors such as Sigfox or NB-IoT, has been chosen due to its good range, battery life and cost efficiency as stated by Mekki et al. [MBCM19].

2.3. IPv6

IPv6 is the successor to the established Internet Protocol (IPv4) standard. First and foremost, the new version allows for a much larger address space to combat the lack of public IP addresses. This is especially important in the context of IoT devices, whose inception has drastically increased their need. On top of that, IPv6 introduces extensions to support authentication, data integrity and data confidentiality for example in the form of the Internet Protocol Security (IPsec) suite which is described in detail in Section 2.4. It also simplifies auto-configuration of addresses and the header format itself shown in Figure 2.2. Most notably, the size of the IP address fields has been increased from 32 to 128 bits. This increases the minimum size of a IPv6 header from 160 bits to 320 bits compared to the IPv4 version. Additional information present in the IPv4 header like fragmentation data, checksums or routing information have either been moved to optional extension headers or have been omitted to be implemented by the communication endpoints on a higher network layer. Extension headers are separate optional headers following the IPv6 header. These can contain information about fragmentation, routing or authentication and allow the IPv6 standard to be easily extended in the future.

2.4. Internet Protocol Security

IP Security (IPsec) can be understood as a suite of protocols that serve as an extension to the Internet Protocol providing "confidentiality, data integrity, access control and data

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|Version| Traffic Class |           Flow Label                  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|         Payload Length        |  Next Header  |   Hop Limit   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+                                                               +
|                                                               |
+                         Source Address                        +
|                                                               |
+                                                               +
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+                                                               +
|                                                               |
+                      Destination Address                      +
|                                                               |
+                                                               +
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 2.2.: IPv6 Header as specified in RFC 8200 [DH17]

source authentication to IP datagrams" [KHN+14]. This is achieved by establishing a shared context between communication partners which includes cryptographic algorithms and keys to be used, as well as the information which services encryption has to be applied to. Note that although IPsec has been designed to be compatible with the IPv4 standard, in the context of this work, IPsec is only used in combination with IPv6 based networks. This is due to limitations imposed by the choice of operating system described in Section 2.5 as well as other considerations including future proving.

To efficiently establish and maintain a shared context between communication partners IPsec uses a separate protocol called Internet Key Exchange (IKE) and stores the information in several internal databases. The shared context is generally referred to as Security Association (SA) and is stored in the Security Policy Database (SPD) and the Security Association Database (SAD) which will be described in detail in Section 2.4.2. The IKE protocol has seen two iterations over its lifespan retroactively named IKEv1 and IKEv2. In the context of this work, the abbreviation IKE generally refers to the updated IKEv2 protocol since at the time of writing IKEv1 has been effectively made obsolete by the newer standard [FK11]. Confidentiality, data origin authentication and integrity are provided by the IP Encapsulating Security Payload (ESP) which is described in Section 2.4.1. The IPsec suite contains an additional security protocol, namely the Authentication Header (AH), which will not be covered in detail in this work, as it does not provide data confidentiality and will not be used.

2.4.1. ESP

As stated above, the Encapsulating Security Payload ensures data integrity and confidentiality. Therefore, the ESP header is located after the IP header in the protocol stack, which in our case will always be an IPv6 header without extensions. The preceding headers i.e. IPv6 will be referred to as 'outer' protocol headers. Figure 2.3 shows the format of the ESP protocol header. The Security Parameters Index (SPI) field is used to identify the corresponding Security Association (SA) in the IPSec Databases. For detailed information please refer to Section 2.4.2. The Sequence Number is a 32-bit counter that increases by one for each packet sent under the corresponding SA. This field serves as protection against replay attacks. The contents of the variable size Payload Data field depend on the mode ESP is currently operated in. The default mode of operation is 'transport mode', where the Payload Data field contains the original IP packet's payload described by the original packet's Next Header field (cf. Section 2.3), which will subsequently be protected by ESP. To provide confidentiality and integrity to the outer headers ESP can be operated in 'tunnel mode' between two security gateways i.e. VPN gateways. In this case the Payload Data field contains the original IP header and the subsequent payload while a new IPv6 header, addressed to the tunnel endpoint, is added before the ESP header. The original headers encapsulated by ESP will be referred to as 'inner' headers. Depending on the cypher used to encrypt the ESP packet, an Initialization Vector (IV) can be placed at the beginning of the Payload Data field. ESP Header fields following the Payload Data are referred to as ESP trailer fields. The Padding and Pad Length fields control optional padding which can be necessary when employing block ciphers and is used to align the Integrity Check Value (ICV) field on a 4-byte boundary. The Next Header field functions in the same way as the corresponding field in the IP header and its value is chosen from a set of IP Protocol Numbers indicating the type of header following ESP. The last field of the ESP header is the Integrity Check Value (ICV) which is computed over the ESP header, Payload and ESP trailer fields. The value and length of the ICV depend on selected options in the SA and the integrity check algorithm used.

2.4.2. IPsec Databases

Since understanding the internals of the major IPsec databases, namely the Security Policy Database (SPD) and Security Association Database (SAD), is essential to the process of deriving context information proposed in this work, the specifications of these databases as provided in [KK05] will be covered in some detail. The IPsec databases are used to represent the concept of Security Associations between communication partners and are stored locally. The IPsec specification provides a model for processing IP traffic in the context of IPsec but does not define the details of database management and operation. While an SA can be set up manually the usual process involves the dynamic negotiation of session parameters like ciphers, encryption keys and other values via the Internet Key Exchange Protocol (IKE) described in Section 2.4.3.

Security Policy Database

The Security Policy Database (SPD) controls which and how IP datagrams are processed by IPsec. The database is divided logically into three parts: SPD-I for inbound traffic, SPD-O for outbound traffic and SPD-S for traffic that is to be protected using IPsec. Each

```
0                   1                   2                   3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ ----
|               Security Parameters Index (SPI)                 | ^Int.
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ |Cov-
|                      Sequence Number                          | |ered
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ | ----
|                    Payload Data* (variable)                   | |   ^
|                                                               | |   |
|                                                               | |Conf.
+               +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ |Cov-
|               |     Padding (0-255 bytes)                     | |ered*
+-+-+-+-+-+-+-+-+               +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ |   |
|                               |  Pad Length   | Next Header   | v   v
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ ------
|         Integrity Check Value-ICV   (variable)                |
|                                                               |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 2.3.: ESP header format as specified in [Ken05]

part of the SPD contains an ordered list of selectors that specify the type of packet to be processed. These selectors include fields for remote and local ip addresses representing source and destination of corresponding connections as well as information about the next layer protocol (e.g. TCP, UDP). In the context of IKE these selectors will be referred to as 'Traffic Selectors'. Each SPD entry classifies packets as BYPASS, DISCARD or PROTECT. A packet classified as BYPASS will be passed down the IP stack just as if the IPsec module was absent. A packet classified as DISCARD will be discarded. A packet classified as PROTECT will be handed to the corresponding IPsec routine to be protected by either ESP or AH. SPD-I and SPD-O entries apply to traffic that is to be discarded or bypassed. SPD-S entries apply to outgoing traffic that is to be protected using IPsec via ESP or AH. This initiates the process that is responsible for the generation of SA session information in the form of SAD entries and usually involves the use of the IKE protocol. Incoming protected traffic will not be processed by SPD rules but instead triggers a lookup in the SAD.

Security Association Database

For each SA there needs to exist one entry in the Security Association Database (SAD). For outgoing traffic this SAD entry is pointed to by the corresponding SPD-S entry. If no SAD entry has been generated yet, the IKE protocol is tasked with populating the SAD entry with data gathered from the SPD selectors as well as session information negotiated with the communication partner. For inbound traffic the SAD entry is used to verify that header fields of the packet match the selector values negotiated for the SA. Each SAD entry contains a Security Parameter Index (SPI), that is transmitted in every ESP or AH packet and used to identify the associated SA (cf. Section 2.4.1). SAD entries also contain information pertaining to integrity and encryption algorithms like key, mode, IV and the algorithm itself. On top of that, each entry contains values that control anti-replay and other security features.

2.4.3. IKEv2

The Internet Key Exchange Protocol Version 2 (IKEv2), in this work referred to as IKE, is tasked with establishing a shared state between communication partners. This process begins by performing mutual authentication between the two parties and the establishment of an IKE SA that is subsequently used to establish SAs for ESP or AH traffic. Authentication can be realized via a shared secret, digital certificate or other means. IKE communication consists of request/response message pairs which are called 'exchanges'. The communication partners in an IKE exchange are referred to as Initiator and Responder, based on their role in the exchange. Note that the IKE protocol is designed in a way that any communication partner can initiate an IKE exchange regardless of their actual role in the network (e.g. server or client). The initial exchanges of IKE communication are called IKE_SA_INIT to establish an IKE SA and IKE_AUTH which transmits identities and sets up an SA for the first AH or ESP child SA. Additional exchanges can be CREATE_CHILD_SA to establish further SAs or INFORMATIONAL in the form of an IKE_NOTIFY to delete existing SAs or notify the communication partner of error conditions or for various other reasons. These additional exchanges can only be performed after the initial exchanges have been completed and an IKE SA has been established.

```
Initiator                         Responder

HDR, SAi1, KEi, Ni  -->
                    <--  HDR, SAr1, KEr, Nr
```

Figure 2.4.: IKE_SA_INIT exchange without optional payloads [KHN+14]

In the IKE_SA_INIT exchange a Diffie-Hellman key exchange is performed. Furthermore, security parameters for the IKE SA are negotiated. Figure 2.4 depicts the packet structure of an IKE_SA_INIT exchange by identifying the headers that are transmitted. Note that the sequence shown is the IKE_SA_INIT without the use of Certificates. A pre-shared key is used instead. HDR refers to the IKE Header, SAi1 and SAr1 refer to the Security Association Payload which carries the proposed IKE SA for the Initiator (SAi1) and Responder (SAr1) side respectively. The additional headers are used to transfer Diffie-Hellman and Nonce values from which keys for the IKE SA are derived. Following the first exchange, the IKE_AUTH exchange is initiated. Therein, the Initiator asserts its identity and proves knowledge of the shared secret. The IKE_AUTH exchange is covered in greater detail in Section 5.2.

SA proposals for both IKE and ESP/AH SAs are transmitted in the Security Association Payload as described above. While the SA payload is covered in greater detail in Section 5.2 it is important to know, that the payload can contain multiple SA proposals with different attributes. Therefore, when negotiating SAs during IKE exchanges, the Initiator can send multiple SA proposals from which the Responder selects none, one or more. If the Responder is not satisfied with any of the proposals, the negotiation fails and has to be reinitiated. In this case, any existing IKE SA remains intact. Rejection of proposals can occur if, for example, the Responder does not support any of the encryption algorithms proposed by the Initiator.

2.5. RIOT OS

RIOT OS is a modular open source operating system designed for devices with minimal resources like low-end IoT devices [BGH+18]. The operating system "provides multi-threading as well as real-time capabilities" [BHG+13] and can be programmed in standard C and C++. RIOT can be run on devices with limited memory in the order of 10kByte and adapts to different architectures in the range of 8 to 32 bits. On top of that, RIOT OS kernel modules allow for some abstraction in memory management e.g. dynamic memory allocation for devices which do not have their own memory management units (MMU). Since RIOT already offers support for many common IoT development boards, all device side development in this work will be based on RIOT OS (cf. Chapter 5).

Generic Network Stack

One of the most important subsystems in the context of this work is RIOT's default IP protocol stack named Generic Network Stack (GNRC) [BGH+18]. Note that GNRC does not support IPv4 and thus limits the development in this work to IPv6 (cf. Section 2.3). GNRC works by encapsulating each protocol in its own thread and uses message queues combined with RIOT's thread-targeted Inter Process Communication (IPC) [LKH+18]. Heart of the network stack is a central packet buffer ('pktbuf') that operates on fractions of packets called 'snips'. These are organised in linked lists and are used to distinguish between different headers and the payload. The central buffer allows GNRC to save memory and avoid packet duplication between different protocol implementations.

3. Related work

In general, header compression algorithms can be divided into stateful and stateless designs. Stateless compression like 6LoWPAN HC1 [MKHC07] uses encoding rules and common values to reduce header size. By not keeping track of a compression context for each flow, stateless algorithms require very little memory to operate but are also limited in their compression efficiency. Therefore, hybrid designs like LOWPAN_IPHC and LOWPAN_NHC [HT11] have been developed to compensate for shortcomings in stateless designs.

Stateful compression, as found in RObust Header Compression (ROHC) [SPJ10], operates based on a shared context between compressor and decompressor. In the case of ROHC this context is dynamic and built during operation. This allows ROHC to refine its compression context for each flow based on different profiles. While allowing good compression efficiency, designs using dynamic context can generate a considerable overhead, since packets are broadcast mostly uncompressed for at least the first transmission of each flow. To eliminate the overhead, compression can be performed on a static context. A static compression context is defined beforehand and does not change during operation. Static Context Header Compression (SCHC) [MTG^{+}20], the compression algorithm employed in this work, is based on this principle. Since SCHC forms part of the foundation of this work, the algorithm is described in more detail in Section 3.1. Note, that the operation of SCHC fragmentation is out of scope for this work and its details are thus omitted. Additional extensions for SCHC have been proposed by Abdelfadeel et al. in [ACP17] [ACP18] and could allow for more flexibility in rule creation.

The second foundational part for this work is formed by the ESP Header Compression and Diet-ESP (EHC) Algorithms [MGBS19]. These are designed to derive static compression context from IPsec SAs and extend compression to the ESP header. EHC is described in detail in Section 3.2. Additionally, compression of IKE communications has recently been proposed by V. Smyslov in [Smy20].

3.1. Static Context Header Compression

The Static Context Header Compression (SCHC) framework provides stateful header compression and an optional fragmentation mechanism [MTG^{+}20]. Since SCHC has been designed for use in LPWANs the individual network nodes are expected to be arranged in a star topology as described in Section 2.2. Compression is performed based on a common static context that is stored locally on both the LPWAN device and gateway. The compression context is represented by a set of rules which describe the actions that need to be performed for compression and decompression. By relying on a static context, SCHC exploits the fact that traffic flows from devices with built-in applications, as used in LPWANs (e.g. sensors, actuators, etc.), are generally known in advance. Unlike other compression strategies (e.g. ROHC [SPJ10], SCHC compression can be applied to Application Layer protocols as well. In the protocol stack, SCHC is situated between a lower (data link) layer and upper (network) layer. In practice this often means a LPWA technology (e.g. LoRa) and IPv6 (cf. Figure 3.1).

```
            +----------------+
            |      IPv6      |
         +- +----------------+
         |  |   Compression  |
   SCHC <   +----------------+
         |  |  Fragmentation |
         +- +----------------+
            |LPWAN technology|
            +----------------+
```

Figure 3.1.: SCHC adaptation layer: Compression and Fragmentation sublayers are located between an upper layer (eg. IPv6) and a lower layer (eg. a LPWAN technology) [MTG+20]

A SCHC packet consists of a RuleID and Compression Residue which form the Compressed Header, which is followed by the payload of the original packet. The RuleID is used to identify the corresponding rule for decompression from the static context. Compression Residue refers to the remaining data that represents the original headers after compression. Note that both RuleID and Compression Residue do not necessarily have to be multiples of bytes in size. SCHC distinguishes between communication partners as Device side (e.g. LPWAN device) and Network Infrastructure side (e.g. gateway). Specifically, the Device side is denoted as 'Dev' and the Network Infrastructure side is denoted as 'App'. SCHC expects the implementation to be able to determine which side of the communication it is operating on and processes Rules in a way that allows them to be applicable to both sides.

3.1.1. SCHC Rules

The SCHC Compression/Decompression (SCHC C/D) context is stored as a set of Rules each referenced by a unique RuleID. Since the scope of the RuleID is limited to a single link between a device and gateway, RuleIDs may be reused by the gateway to store context information for separate devices. The manner in which devices are identified by the gateway is not specified by SCHC and is usually expected to be determinable from the lower layer protocol. "The main idea of the SCHC compression scheme is to transmit the RuleID [...] instead of sending known field values" [MTG+20].

A SCHC Rule consists of a list of Field Descriptors (FDs) in the order in which the fields appear in the packet header. Each Field Descriptor is identified by a Field Identifier (FID). Matching header fields to a specific FID (e.g. IPv6 Traffic Class) is performed by a separate protocol parser which is not part the SCHC specification. Field Length (FL) specifies the field length in bits in the original header, which can also be variable (please refer to [MTG+20] for further details). Field Position (FP) specifies the position of the field in case the original header contains multiple fields with the same FID. The default value for the FP is 1. This indicates that the FD represents the first occurrence of the header field. The Direction Indicator (DI) can either be 'Up' for packets travelling Uplink, 'Dw for packets travelling Downlink, or 'Bi' for both. The DI allows SCHC to perform asymmetric processing without the need for additional Rules. Target Value (TV), Matching Operator (MO) and Compression/Decompression Action (CDA) are what is used to perform the actual compression/decompression on the packet. The MO is used to match the field value from the original header to the TV while CDA describes the actions that need to be performed to apply compression/ and decompression respectively. Matching Operators and Compression/Decompression Actions are described in further detail in Section 3.1.2.

When processing packets, SCHC selects a matching SCHC Rule and performs the Compression/Decompression Actions. In the case of compression, SCHC examines the set of Rules to find a Rule that matches all fields in the packet with a Field Descriptor and performs the corresponding CDAs on all fields. The final Compression Residue is made up of the concatenation of residues from the individual fields after performing the CDA. In the case of decompression, SCHC identifies the corresponding Rule by the transmitted RuleID and uses the CDAs to reconstruct the original headers.

3.1.2. Matching Operators and C/D Actions

Matching Operators in SCHC are used to evaluate packets for compression by identifying applicable Rule(s) in the set. MOs are applied on the Field Value (FV), which represents the value of the header field in the original header, and the Target Value (TV) which is stored in the corresponding Field Descriptor. MOs can return either True or False. A matching Rule has been found if all MOs are True and the packet contains no additional fields not represented in the Rule. SCHC supports several different MOs as outlined in Table 3.1. MOs range from simple equality checks to ones that compare most significant bits or match to sets of values.

Table 3.1.: List of SCHC Matching Operators(MOs) [MTG+20]

Name	Operation
equal	True if Field Value (FV) matches Target Value (TV)
ignore	Always True
MSB(x)	True if x most significant bits match between FV and TV
match-mapping	True if FV matches one entry in TV list

CDAs are used to apply the actual compression/decompression to header fields and consist of pairs of actions. Table 3.2 shows the full list of SCHC Compression/Decompression Actions. These include sending the actual header value without compression and eliding the field completely since its value is already known. The CDAs DevIID and AppIID are targeted at IPv6 specifically and are applied to the 64-bit interface identifier of the address. As stated before, SCHC distinguishes explicitly between Device side and Network Infrastructure side and handles network addresses using these classifications instead of categorizing them as sender or receiver for each transmission. Furthermore, SCHC recomputes some Field Values at the receiving end. These include fields where the algorithm for computation is unambiguously defined by the relevant protocol specification (e.g. IPv6 length, UDP checksum). These computations are performed after all other CDAs have been applied. Note that the concrete implementations, while not specified by SCHC, need to take into

Table 3.2.: List of SCHC Compression/Decompression Actions(CDAs) [MTG^{+}20]

Action	Compression	Decompression
not-sent	elided	use TV stored in Rule
value-sent	send	use received value
mapping-sent	send index	retrieve value from TV list
LSB	send least significant bits (LSB)	concatenate TV and received value
compute-*	elided	recompute at decompressor
DevIID	elided	build IID from L2 Dev addr
AppIID	elided	build IID from L2 App addr

account the order in which compute-* actions are to be performed in case several of them exist in a single Rule.

3.2. ESP Header Compression (EHC) and Diet-ESP

EHC has been designed to enable the use of header compression in IPsec protected traffic, namely ESP [MGBS19]. Since ESP encrypts the packet's payload, including some header information, classic header compression algorithms like 6LowPAN or ROHC fail to compress those headers. Furthermore, EHC is able to leverage context information from IPsec Security Associations for use in header compression. While EHC is, at its core, based on a static compression context, it is, unlike SCHC, designed to share context information with communication partners prior to engaging compression. This initiation process uses the IKE protocol (cf. Section 2.4.3) to negotiate special IPsec modes called 'Compressed Transport' and 'Compressed Tunnel'. Diet-ESP is the name of a specific EHC Strategy also specified in [MGBS19].

3.2.1. ESP Processing

To successfully apply header compression to ESP traffic, EHC divides the processing of ESP packets into the following phases.

- **Pre-esp** processing applies to headers other than ESP that are encrypted in the final ESP packet. These include Network or Application Layer protocols that are part of the ESP payload. These actions have to be performed before ESP can apply its encryption algorithm.
- **Cleartext-esp** processing applies to the plaintext ESP packet to allow compression of the ESP trailer or encapsulated traffic in the case of tunnel-mode ESP (cf. 'inner headers' Section 2.4.1).
- **Post-esp** processing is performed on the encrypted ESP packet and allows compression of the ESP header itself.

While pre-esp and post-esp processing can be integrated into the protocol stack without interfering with the ESP implementation, processing in the cleartext-esp phase needs to be integrated into ESP.

3.2.2. EHC Context

A EHC Context exists for each IPsec SA and can be defined for any protocol encapsulated by ESP and for ESP itself. The EHC Context Parameters specify for each individual header which fields can be obtained from the IPsec SA and which need to be transmitted or negotiated beforehand. Table 3.3 shows the EHC Context Parameters for inner IPv6 as specified by the Diet-ESP strategy (cf. Section 3.2.4)[MGBS19]. 'In SA' indicates whether values for this header field can be derived from the SA. In this case, the IPv6 addresses can be obtained from the selector in the SPD which has been negotiated as part of the Traffic Selector payload during the IKE_AUTH or CREATE_CHILD_SA exchange (cf. IKE Section 2.4.3 and SPD Section 2.4.2). The attributes ip6_tcfl_comp and ip6_hl_comp specify whether values for IPv6 Traffic Class and Hop Limit can be derived from the outer ipv6 header, are part of the negotiated EHC Context or need to be transmitted without compression.

Table 3.3.: List of ECH Context Parameters for the inner IPv6 header [MGBS19]

Context Attribute	In SA	Possible Values
ip6_tcfl_comp	No	"Outer", "Value", "UnComp"
ip6_tc	No	IPv6 Traffic Class
ip6_fl	No	IPv6 Flow Label
ip6_hl_comp	No	"Outer", "Value", "UnComp"
ip6_hl	No	Hop Limit Value
ip6_src	Yes	IPv6 Source Address
ip6_dst	Yes	IPv6 Destination Address

Additionally, EHC provides Context Parameters for ESP, inner IPv4, UDP, UDP-Lite and TCP which will not be covered in detail in this section and can be found in the RFC draft [MGBS19].

3.2.3. EHC Rules

Similar to SCHC described in Section 3.1, EHC uses Rules to apply compression to each header field separately. A EHC Rule consists of a Rule Name, Field Identifier, Action and Parameters. Action is the analogue to the SCHC CDA and can take the following values:

- **send-value** means no compression is applied and the actual field value is sent.
- **elided** means no value is sent and the field value as a whole can be reconstructed from the EHC Context.
- **lsb(_lsb_size)** specifies, that only the indicated number of least significant bytes need to be sent, the remainder of the field value can be reconstructed from the EHC Context.
- **lower** indicates, that the field value can be retrieved from the lower layer protocol (e.g. outer IPv6) and therefore is not transmitted.
- **checksum** specifies, that the field value is not sent and can be computed by the corresponding algorithm on the receiver's side (e.g. TCP checksum). This is possible

because the Integrity Check Value (ICV) in the ESP header already guarantees the integrity of the packet.

- **padding(_align)** is specific to ESP and allows computation of the padding length.

EHC Rules are only activated if the corresponding Field exists and Parameters are defined as a 'Single' value (i.e. Port Number, IP Address, etc.). If any of the Parameters are undefined or ambiguous, no compression can be performed and the packet is discarded.

Table 3.4 shows the EHC Rules used to process the ESP header. The SPI and Sequence Number field are compressed/decompressed by the 'lsb' Action during the post-esp phase. If the ESP_NH Rule is activated, the ESP Next Header field is compressed/decompressed during the cleartext-esp phase using the 'elided' Action. Its value can be derived from the IPsec Mode (IP if tunnel mode) or if the IPsec Traffic Selector specifies a single protocol. [MGBS19] contains additional EHC Rules for inner IPv4, inner IPv6, UDP, UDP-Lite and TCP.

Table 3.4.: EHC Rules for the ESP header [MGBS19]

EHC Rule	Field	Action	Parameters
ESP_SPI	SPI	lsb	esp_spi_lsb, esp_spi
ESP_SN	Sequence Number	lsb	esp_sn_lsb, esp_sn_gen, esp_sn
ESP_NH	Next Header	elided	l4_proto, ipsec_mode
ESP_PAD	Pad Lenght, Padding	padding	esp_align, esp_encr

3.2.4. Diet-ESP Strategy

Diet-ESP is an EHC Strategy designed as a compromise between compression efficiency and ease of configuration. The strategy contains rules for the ESP, IPv4, IPv6, UDP, UDP-Lite and TCP protocols. However, some Rules are designed to only activate if specific assumptions apply. These assumptions include that the algorithm responsible for ESP Sequence Number generation is set to 'Incremental' and no IPv4 options are enabled. Table 3.5 and Table 3.6 show the Diet-ESP Rules for the ESP header, as well as the origin and possible values for the individual EHC Parameters.

Of the indicated Parameters, only 'esp_sn_lsb', 'esp_spi_lsb' and 'esp_align' need to be agreed on during negotiation. The first two Parameters indicate the number of least significant bits transmitted for Sequence Number and SPI respectively. The latter defines the alignment that has to be observed to guarantee compatibility with the particular operating systems.

The additional Diet-ESP Rules indicated above (IPv4, IPv6, ...) are implemented in a similar fashion and are not presented in detail in this work. The strategy leaves EHC with a total of seven Context Parameters that are not accounted for in the SA and need to be negotiated. Negotiation entails use of the IKE protocol to initiate compression and exchange values for the missing Context Parameters. The process is described in the following section.

Table 3.5.: Diet-ESP Context Parameters for ESP [MGBS19]

Context Attribute	In SA	Possible Values
ipsec_mode	Yes	"Tunnel", "Transport"
outer_version	Yes	"IPv4", "IPv6"
esp_spi	Yes	ESP SPI
esp_spi_lsb	No	0, 1, 2, 3, 4
esp_sn	Yes	ESP Sequence Number
esp_sn_lsb	No	0, 1, 2, 3, 4
esp_sn_gen	No	"Time", "Incremental"
esp_align	No	8, 16, 24, 32
esp_encr	Yes	ESP Encryption Algorithm

Table 3.6.: Diet-ESP Rules for the ESP header [MGBS19]

EHC Rule	Activated if	Parameter	Value
ESP_SPI	Diet-ESP	esp_spi_lsb	Negotiated
		esp_spi	In SA
ESP_SN	Diet-ESP	esp_sn_lsb	Negotiated
		esp_sn_gen	Negotiated
		esp_sn	In SA
ESP_NH	Diet-ESP	ipsec_mode	In SA
		l4_proto	In SA
ESP_PAD	Diet-ESP	esp_align	Negotiated
		esp_encr	In SA

3.2.5. IKE Extension

Negotiation of the EHC Context Parameters and EHC Strategy is achieved using IKE Notify Payloads as specified in the 'Internet Key Exchange version 2 (IKEv2) extension for the ESP Header Compression (EHC) Strategy' [MGS18]. For this goal, three additional Notify Payloads are proposed. The payloads are as follows:

- USE_COMPRESSED_MODE: This payload is used during the IKE Initial Exchanges to establish the use of EHC compression for the negotiated SA. The Initiator appends a USE_COMPRESSED_MODE Notify Payload to the IKE_AUTH exchange to indicate the use of compression for the SA. The payload can further be used to specify the compression mode to be used. Supported compression modes are 'Compressed Transport Mode' and 'Compressed Tunnel Mode', the former being the default value. The Responder may answer the request for compression by responding with a USE_COMPRESSED_MODE payload to indicate it accepts the request and is able to implement the requested compression mode.
- EHC_STRATEGY_SUPPORTED: This payload is also transmitted by the Initiator during the IKE_AUTH exchange and indicates that the negotiation of EHC Strategies

is supported. Furthermore, the payload may specify ranges of values for Context Parameters and the EHC Strategy. The Responder may acknowledge by returning a EHC_STRATEGY_SUPPORTED payload with the specific parameters agreed upon.

- EHC_STRATEGY_UNACCEPTABLE_PARAMETER: This payload is used to indicate that the parameters proposed with the EHC_STRATEGY_SUPPORTED payload are not acceptable. This can happen if the Responder's supported values for the negotiated parameters do no match the Range Values provided by the Initiator. In this case negotiation of a regular ESP SA or the deletion of the SA is suggested.

The IKE extensions further specifies a compressed format to exchange parameter values using the Notify Payload. The specifics will not be covered in detail.

4. Concept

In this chapter, we introduce a method to derive static compression context from IPsec Security Associations. In the first part, we designate our design goals and define the general parameters including network topology, protocol stack and the IPsec use cases considered in this work. Subsequently, we determine the requirements for SCHC processing in encrypted traffic and draft a strategy for practical implementation. In the following part, we adapt some of the Context Parameters introduced by Migault et al. [MGBS19] for use in combination with SCHC and develop the concept of Rule Templates to allow further specification. In the later parts of the chapter, we manage several special cases and introduce an extension to the concept that is designed to reduce the network overhead introduced by IPsec.

4.1. Design Goals

Based on the challenges for IoT introduced in Section 2.1 we can determine that a header compression scheme for constrained devices not only has to provide good compression efficiency, but also has to respect their limited capabilities and strict energy budget. Additionally, any traffic overhead generated by normal IPsec operation has be to considered and potentially reduced. Furthermore, to facilitate the integration process, it would seem advantageous to limit the amount of changes required for SCHC and IPsec. From the restrictions examined above, we define the following design goals:

(I.) Apply SCHC based header compression on ESP protected traffic, that

- a) reduces SCHC configuration costs by deriving compression context from IPsec SAs;
- b) is able to compress encrypted headers in the ESP payload;
- c) achieves reasonable compression without requiring substantial changes to SCHC or IPsec;

(II.) Extend SCHC and/or IPsec to

- a) increase compression efficiency;
- b) reduce traffic overhead generated by IPsec;

(III.) Explore the limits of header compression for ESP protected traffic using SCHC.

4.2. Simplified Topology

Section 2.2 introduces the star topology commonly used in networks employing LPWA technology. The remainder of this work is based on the simplified topology shown in Figure 4.1. A single constrained device, referred to as Device, represents the IoT side of the network.

The Device is connected via an LPWA technology to a single entity that performs the role of both radio and network gateway and is referred to as Gateway. The Gateway itself connects to a Server over the internet. It acts as a bridge to allow the Device access to the Server and vice versa.

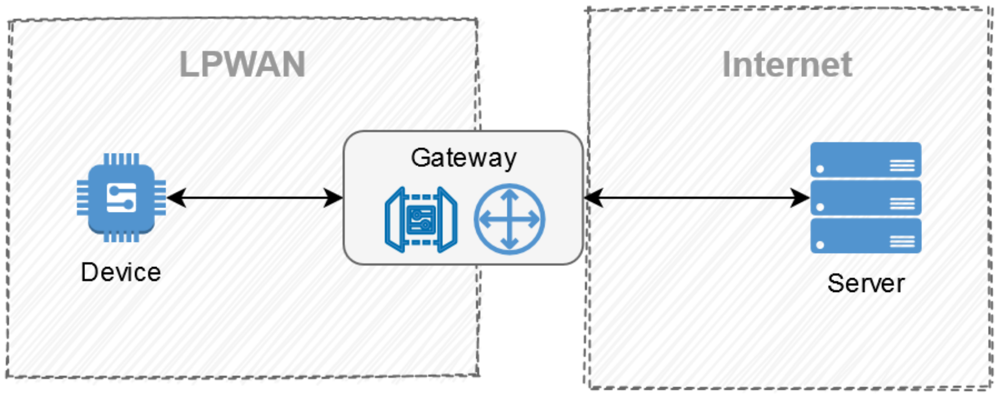

Figure 4.1.: Simplified network topology showing a single Device connected to a Server via a Gateway. The Device is connected to the Gateway via a LPWA technology. The Gateway connects to the Server over the Internet.

4.3. Protocol Stack

While SCHC is designed to be usable with a multitude of protocols from different layers, in the scope of this work the protocol stack is limited to the one shown in Figure 4.2. For this chapter, the Data Link Layer (l2) protocol is unspecified and represents any LPWA technology for the connection between Device and Gateway and any generic Layer 2 protocol for the connection between Gateway and Server. The Network Layer protocol is represented by IPv6 with integrated IPsec as specified in [KK05]. The Transport Layer protocol is represented by UDP. The protocol stack does not contain any upper Layer protocols (i.e. Application Layer like CoAP). The stack is intended to represent a generic network stack used in various fields of application for IoT.

4.4. IPsec Use Cases

Since the header compression proposed in this work is based on the application of SCHC compression on IPsec protected traffic, we need to consider common use cases for IPsec encryption. This is especially important since some areas of application may present additional obstacles for header compression. To find and organize use cases we adopt the network topology shown in Figure 4.1. Additionally, we group the areas of application by the network segment(s) that are to be protected.

- Wireless Traffic: Since wireless traffic is inherently susceptible to interception and attacks the connection between Device and Gateway is most likely to be the focus

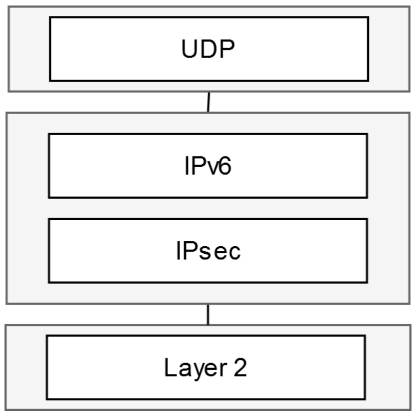

Figure 4.2.: Protocol stack with integrated IPsec implementation.

of encryption. Assuming the Gateway can be trusted, it is sufficient to protect the link between Device and Gateway using IPsec. In this case, an IPsec SA using either 'transport' or 'tunnel mode' is established between Device and Gateway.

- Internet Traffic: The intention when protecting traffic between Gateway and Server can be either to protect the traffic contents or to hide the address(es) of the device(s) that are located behind the Gateway. To achieve the former, using 'transport mode' between Gateway and Server is sufficient. When trying to hide endpoint addresses 'tunnel mode' must be used. In both cases, the Device itself is entirely unaffected. To benefit from compression proposed in this work an additional IPsec SA would have to be established between Device and Gateway.
- End-to-End: Combining some of the use cases introduced above, some areas of application may require end-to-end encryption. This could be the case if the location of the Gateway has limited physical security (e.g. located outside) and cannot be trusted to the point that traffic is allowed to be processed in unencrypted form. In this instance, an IPsec SA has to be established between the Device and Server using either 'tunnel mode' or 'transport mode'. Because a shared context is only established between Device and Server but the Gateway might still need to decompress headers to allow forwarding (e.g. outer IPv6 header), this configuration poses additional challenges (cf. Section 4.7.4).

4.5. Packet Processing

This section describes the processing steps required to apply compression/decompression to ESP protected traffic. In Section 4.5.1 we take a look at a processing example to establish the basic requirements when combining compression with encryption. In Section 4.5.2 a generalized model for SCHC integration into ESP is provided.

4.5.1. Example

To demonstrate the requirements and functionality of header compression in ESP protected traffic we examine a network where encryption is established between device and gateway only(cf. Figure Figure 4.1) . Furthermore, ESP traffic is expected to employ "transport mode" as described in Section 2.4.

```
                     BEFORE APPLYING ESP
            ---------------------------------------
      IPv6  |             | ext hdrs |     |      |
            | orig IP hdr |if present| UDP | Data |
            ---------------------------------------

                    AFTER APPLYING ESP
            ---------------------------------------------------------
      IPv6  | orig |hop-by-hop,dest*,|   |dest|   |    | ESP   | ESP|
            |IP hdr|routing,fragment.|ESP|opt*|UDP|Data|Trailer| ICV|
            ---------------------------------------------------------
                                         |<--- encryption ---->|
                                     |<------ integrity ------>|
```

Figure 4.3.: ESP header encapsulation in IPv6 [Ken05]

Section 3.1 describes how to apply a set of SCHC rules on existing network traffic between a device and gateway. To extend this concept to ESP protected traffic, certain specifics of the encrypted packet have to be considered. Figure 4.3 shows the encapsulated headers of an IPv6/UDP packet as seen inside ESP traffic. As described in Section 2.4.1 the second part of the ESP header as well as the payload are not only covered by ESP's integrity protection, but are also encrypted. To allow successful compression and decompression of encrypted headers the compression algorithm has to be integrated in such a way, that it can process headers before and after passing them to ESP for decryption. On top of that, compression can be applied to the ESP header itself as proposed in the Diet-ESP EHC Strategy (cf. Section 3.2.4).

To process outgoing traffic, we start with a protocol stack as described in Figure 4.2. The packet is first handed to IPsec to construct an ESP header and encapsulate the UDP packet as its payload. This needs to be performed first, since in the case of 'tunnel mode' the original IPv6 header is also encapsulated. Before ESP applies encryption, any Field Descriptors regarding the encapsulated payload and the ESP trailer have to be applied. Following that, the packet is handed back to ESP for encryption and integrity protection. Once completed, compression can be applied to the ESP and outer IPv6 header.

When processing inbound traffic, the outer IPv6 and ESP header fields have to be decompressed before any other packet processing can occur. In the next step ESP is tasked with performing integrity checks and decrypting the packet. Finally, the inner headers and ESP trailer can be decompressed.

The number of steps that need to be performed to apply compression to ESP protected traffic suggests that SCHC Rules have to be applied in multiple stages. In the following Section we propose processing phases that differ slightly from those described in Section 3.2.

4.5.2. Generalized Processing

Section 3.2.1 describes how EHC divides ESP packet processing into several phases. In the previous section we demonstrate their necessity as we need to be able to apply compression/decompression before and after encryption, as well as during the ESP routine. Instead of separating phases into before, during and after ESP, we propose phases that are characterized by the state of the packet. This eliminates the need to distinguish between inbound and outbound traffic when describing actions performed during said phases and improves readability. The phases are defined as follows:

- **plaintext phase:** In the plaintext phase the packet resides in memory in encapsulated form without encryption. During this phase, compression/decompression is applied to the inner headers and the ESP trailer.
- **ciphertext phase:** In the ciphertext phase the packet resides in memory in encrypted form. During this phase, compression/decompression is applied to the outer and ESP header(s).

Note that the order in which the phases appear in packet processing now depends on traffic direction. The order for outbound traffic is plaintext phase followed by ciphertext phase. For ingoing traffic processing is applied in reverse order. The integration of SCHC into the protocol stack is described in further detail in Section 5.5.1.

4.6. Deriving SCHC Rules

As has been established by Migault et al. ([MGBS19])(cf. Section 3.2) some static context for use in header compression can be derived from IPsec SAs. For this to work, an IPsec SA has to already be established between communication partners using IKE. Since Migault et al. were able to derive rules for their ESP header compression from IPsec context, this section focuses on converting these rules into compression context usable by SCHC. Note that an EHC Rule is a policy that is applied to a single header field whereas a SCHC Rule includes all header fields possibly consisting of multiple headers. In SCHC the equivalent of an EHC Rule is called a Field Descriptor (cf. Section 3.1). Since we use SCHC to apply compression/decompression, SCHC terminology is used.

4.6.1. Context Parameters

In the first step, we define Context Parameters similar to those introduced in [MGBS19](cf. Section 3.2). Each Context Parameter holds information to perform compression/decompression on a single or multiple header fields. For some fields multiple parameters are required, while others do not require any parameters. Fields without parameters can be reconstructed by other means and are detailed in the sections describing rule generation for their respective headers (cf. Section 4.6.2). For each Context Parameter we specify the source from which it can be obtained. Possible sources are 'In SA' and 'Negotiated'. 'In SA' indicates that the information represented by this Context Parameter can be obtained from the IPsec SA. 'Negotiated' indicates that the value of this parameter has to be agreed on by some other means (cf. Section 4.7.1). Note that some Context Parameters marked with 'In SA' might not actually be present in the IPsec databases since they depend on the actual IPsec configuration. Other parameters could hold multiple or ambiguous values. Section 4.6.3 introduces a method how viable SCHC Rules can be generated nevertheless.

ESP Parameters

The Context Parameters relevant to process ESP headers are shown in Table 4.1. 'esp_spi_lsb' specifies the number of least significant bits which are transmitted instead of the full SPI value. The parameter can not be obtained from the SA and has to be negotiated. Possible values range from 0 to 32, indicating the transmission of: no SPI, the complete SPI or anything in between. 'esp_spi' represents the Security Parameter Index (SPI) which is located in the SAD for the corresponding SA. 'esp_sn_lsb' specifies the number of least significant bits which are transmitted instead of the full Sequence Number. The parameter can not be obtained from the SA and has to be negotiated. Possible values range from 1 to 31. This range prohibits the complete omission of the Sequence Number field in compressed traffic to retain some means of reordering ESP traffic. 'esp_sn' represents the 32-bit Sequence Number (SN) and can be derived from the 64-bit Sequence Number Counter entry in the SAD. 'l4_proto' holds the protocol number of the next header following IP. This value can be retrieved from the Next Layer Protocol value in the selector assigned to the SA in the SPD. 'ipsec_mode' specifies the mode of operation for IPsec (tunnel or transport). The value can be obtained from the IPsec protocol mode value in the SAD.

Table 4.1.: Context Parameters for the ESP header.

Context Parameter	Source	Possible Values
esp_spi_lsb	Negotiated	0...32
esp_spi	In SA	SPI
esp_sn_lsb	Negotiated	1...31
esp_sn	In SA	Sequence Number
l4_proto	In SA	Next Header
ipsec_mode	In SA	"Tunnel", "Transport"

IPv6 Parameters

The parameters relevant to process IPv6 headers are shown in Table 4.2. 'ip_version' indicates the version of the IP protocol in use. This value can be derived from the Remote IP Address field(s) of the selector in the SPD. Note that IPsec supports inner and outer IP header to be of different versions. However, Network Layer protocols other than IPv6 are out of scope for this work. 'ip_tc' holds the Traffic Class value for the IPv6 header and has to be negotiated. 'ip_fl' holds the Flow Label value for the IPv6 header and has to be negotiated. 'ip_hl' holds the Hop Limit value for the IPv6 header and has to be negotiated. 'ip6_dev_prefix' holds the 64-bit network prefix for the IPv6 address of the Device. 'ip6_dev_iid' holds the 64-bit interface identifier for the IPv6 address of the Device. 'ip6_app_prefix' holds the network prefix for the IPv6 address on the Network Infrastructure side (Gateway or Server). 'ip6_app_iid' holds the interface identifier for the IPv6 address on the Network Infrastructure side. 'ip6_tun_prefix' is only used if 'tunnel mode' is selected and holds the network prefix for the IPv6 address of the tunnel endpoint. This corresponds to the source/destination IP for the outer IPv6 header. 'ip6_tun_iid' is only used if 'tunnel mode' is selected and holds the interface identifier for the tunnel endpoint. This corresponds to the source/destination IP for the outer IPv6 header. All address values can be obtained from the Local/Remote IP Address fields of the selectors in the SPD.

Table 4.2.: Context Parameters for the IPv6 header.

Context Parameter	Source	Possible Values
ip_version	In SA	IP Version
ip_tc	Negotiated	Traffic Class
ip_fl	Negotiated	Flow Label
ip_hl	Negotiated	Hop Limit
ip6_dev_prefix	In SA	Device IPv6 Prefix
ip6_dev_iid	In SA	Device IPv6 IID
ip6_app_prefix	In SA	App IPv6 Prefix
ip6_app_iid	In SA	App IPv6 IID
ip6_tun_prefix	In SA	Tunnel IPv6 Prefix
ip6_tun_iid	In SA	Tunnel IPv6 IID

UDP Parameters

The Parameters relevant to process UDP headers are shown in Table 4.3. 'udp_dev_port' holds the port value for the Device side. 'udp_app_port' holds the port value for the Network infrastructure side. Both values can be obtained from the port selector of the next layer protocol in the SPD.

Table 4.3.: Context Parameters for the UDP header.

Context Parameter	Source	Possible Values
udp_dev_port	In SA	Device Port
udp_app_port	In SA	App Port

4.6.2. Rule Creation

Based on the Context Parameters defined previously, this section focuses on the creation of viable SCHC Rules. The process is designed to take place before transmission of compressed traffic between the two endpoints, but after IPsec SAs have been negotiated. To generate a Rule, we have to define a single Field Descriptor for every header field present. In contrast to EHC, a non-matching Field Descriptor is not skipped, but instead causes the entire Rule to fail. Therefore, Rule generation needs to represent the expected contents of the headers as closely as possible without restricting any FDs to a point that results in failure to match when applying its MOs. Another notable difference to EHC is the fact that we want to apply compression to both the outer and inner headers. Therefore, we can not use information gained from the outer headers to facilitate compression of the inner headers.

As stated in Section 4.6.1, Context Parameters derived from SAs can not be guaranteed to hold unambiguous values or any value at all. For this reason the Rule creation process needs to be able to adapt when Context Parameters are absent or ambiguous. The adaptation process is called Rule Downgrading and is detailed in Section 4.6.3. It allows the Rule creation process to decrease the specificity of single FDs based on the granularity of the SA. On that account, this section demonstrates how to obtain 'best case' FDs when all Context Parameters are viable. The Rules presented in this section are referred to as Rule Templates from which actual SCHC Rules can be generated by replacing the Target Value (TV) with the value of the corresponding Context Parameter. In some cases where multiple Context Parameters are indicated, one of the parameters is considered a parameter for the CDA as specified in the corresponding section. Note that the parameter for the MSB MO can be derived from the parameter of the LSB CDA using the following formula, where msb_p is the parameter for the MSB MO, lsb_p is the parameter for the LSB CDA and FL is the Field Length (FL): $msb_p = FL - lsb_p$

ESP Rule Template

The Rule Template for ESP is shown in Table 4.4. In the ESP header only the SPI, Sequence Number and Next Header fields are viable candidates for compression. Compression of the Payload Data field has to be skipped in this Rule, since it either contains header(s) specified in a different Rule or no compressible headers at all. The Padding and Pad Length fields need to be intact for compression but could possibly be compressed afterwards. Refer to Section 4.6.4 for considerations. The ICV field is responsible for the integrity protection of the entire packet and therefore cannot be compressed. For these reasons, the aforementioned header fields have been assigned the MO 'ignore' and CDA 'not-sent' indicating that no compression/decompression is performed. The ESP SPI field can be reconstructed from the 'esp_spi' Context Parameter by performing the LSB CDA with 'esp_spi_lsb' as parameter. The ESP Sequence Number field can be reconstructed from the 'esp_sn' Context Parameter by performing the LSB CDA with 'esp_sn_lsb' as parameter. Similar to the EHC Diet-ESP strategy, we expect the use of incremental sequence number generation as introduced in [KHN+14]. If sequence numbers are generated by a different algorithm, the ESP Sequence Number field can not be compressed and the 'value-sent' CDA has to be used instead. The ESP Next Header field can be reconstructed from the 'l4_proto' and 'ipsec_mode' Context Parameters and can be eliminated. More specifically, if 'ipsec_mode' indicates the use of 'tunnel mode' the next header is known to be IPv6, otherwise, the next header protocol is specified by 'l4_proto'.

IPv6 Rule Template

The Rule Template for IPv6 is shown in Table 4.5. In the IPv6 header all fields are viable for compression. Note that the use of IPv6 Extension Headers is not supported. The IPv6 Version field can be reconstructed from the 'ip_version' Context Parameter and is not sent. The IPv6 Traffic Class field can be reconstructed from the 'ip_tc' Context Parameter and is not sent. The IPv6 Flow Labeld field can be reconstructed from the 'ip_fl' Context

Table 4.4.: Rule Template for the ESP header.

FID	FL	FP	Di	Target Value	MO	CDA
ESP SPI	32	1	Bi	esp_spi_lsb, esp_spi	MSB	LSB
ESP Sequence Number	32	1	Bi	esp_sn_lsb, esp_sn	MSB	LSB
ESP Payload Data	variable	1	Bi	-	ignore	value-sent
ESP Padding	variable	1	Bi	-	ignore	value-sent
ESP Pad Length	8	1	Bi	-	ignore	value-sent
ESP Next Header	8	1	Bi	l4_proto, ipsec_mode	equal	not-sent
ESP ICV	32	1	Bi	-	ignore	value-sent

Parameter and is not sent. The IPv6 Payload Length field can be computed on the receiver's side and is not sent. The IPv6 Next Header field can be reconstructed from the 'l4_proto' and 'ipsec_mode' Context Parameters and is not sent. More specifically, if 'ipsec_mode' indicates the use of 'tunnel mode' the next header is known to be ESP, otherwise, the next header protocol is specified by 'ip_l4proto'. The IPv6 Hop Limit field can be reconstructed from the 'ip_hl' Context Parameter and is not sent. The IPv6 address corresponding to the Device can be reconstructed from the 'ip6_dev_prefix' and 'ip6_dev_iid' Context Parameters. For 'transport mode' the address of the communication partner can be reconstructed from the 'ip6_app_prefix' and 'ip6_app_iid' Context Parameters. In the case of 'tunnel mode' these have to be replaced with the 'ip6_tun_prefix' and 'ip6_tun_iid' Context Parameters when processing the outer IPv6 header. The specific mapping of Dev and App addresses to the Source and Destination Address fields is performed by SCHC. For details on how to identify the network roles of each endpoint from its own perspective refer to Section 4.7.3.

Table 4.5.: Rule Template for the IPv6 header.

FID	FL	FP	DI	Target Value	MO	CDA
IPv6 Version	4	1	Bi	ip_version	equal	not-sent
IPv6 Traffic Class	8	1	Bi	ip_tc	ignore	not-sent
IPv6 Flow Label	20	1	Bi	ip_fl	ignore	not-sent
IPv6 Payload Length	16	1	Bi	-	ignore	compute-*
IPv6 Next Header	8	1	Bi	l4_proto, ipsec_mode	equal	not-sent
IPv6 Hop Limit	8	1	Bi	ip_hl	ignore	not-sent
IPv6 DevPrefix	64	1	Bi	ip6_dev_prefix	equal	not-sent
IPv6 DevIID	64	1	Bi	ip6_dev_iid	equal	not-sent
IPv6 AppPrefix	64	1	Bi	ip6_app_prefix	equal	not-sent
IPv6 AppIID	64	1	Bi	ip6_app_iid	equal	not-sent

UDP Rule Template

In the UDP header all fields are viable for compression. The Rule Template for UDP is shown in Table 4.6. The UDP Dev Port and UDP App Port fields represent the Source and Destination Port fields of the UDP header. The precise mapping is determined by SCHC (cf. Section 4.7.3). The fields can be reconstructed form the 'udp_dev_port' and 'udp_app_port' Context Parameters and are not sent. The UDP Length field can be computed at the receiver's side and is not sent. The UDP Checksum field can be computed at the receiver's side an is not sent. Note that since IPsec already provides integrity protection by use of the ICV, the UDP Checksum can safely be omitted.

Table 4.6.: Rule Template for the UDP header.

FID	FL	FP	DI	Target Value	MO	CDA
UDP Dev Port	16	1	Bi	udp_dev_port	equal	not-sent
UDP App Port	16	1	Bi	udp_app_port	equal	not-sent
UDP Length	16	1	Bi	-	ignore	compute-*
UDP Checksum	16	1	Bi	-	ignore	compute-*

4.6.3. Rule Downgrading

As stated before, Context Parameters can not be guaranteed to specify unambiguous values. This is due to the variable granularity allowed for in the IPsec configuration. More specifically, IPsec selectors in the SPD can not only hold single values, but are allowed to hold ranges of values as well as the special values ANY and OPAQUE. Ranges of values are often used to specify address or port ranges when a single SA is configured to handle multiple services or multiple endpoints. The special values ANY and OPAQUE are used if the precise value is either of no concern or unavailable [KK05]. For SCHC this poses the problem that Rules generated from the Rule Templates introduced in the previous section may hold Target Values (TVs) that are ambiguous or hold no usable values at all. This would cause the respective MOs to fail and lead to the rejection of the Rule. Therefore, we have to account for variable granularity in the IPsec configuration during Rule generation. To achieve this, we propose a concept called Rule Downgrading. In this process, the Rule Templates introduced in the previous section are adapted to fit the actual specificity of the IPsec SA. The process distinguishes between ranges of values and the special values ANY and OPAQUE which are treated as 'no value'. If a Context Parameter holds a range of values the parameter is defined as 'ambiguous'. If a Context Parameter holds no value or the corresponding SPD selector holds the value ANY or OPAQUE, the parameter is defined as 'unspecified'.

No Value

If a Context Parameter is 'unspecified' the corresponding header field can not be compressed. Therefore, the FD specified in the Rule has to be modified in the following way:

- set the MO to 'ignore'

- set the CDA to 'value-sent'

Value Ranges

If a Concept Parameter is 'ambiguous' and the MO/CDA pair for the FD in question is set to 'equal'/'not-sent' in the Rule Template, it can be processed as follows:

- determine the minimum number of least significant bits that represent the range(s) of values lsb_p
- set the TV to any value within the range
- set the MO to 'MSB($FL - lsb_p$)'
- set the CDA to 'LSB(lsb_p)'

In any other case, or if the processing above fails at any step, the Context Parameter is processed as 'unspecified'.

Note that the IPsec configuration allows for the use of multiple ranges of values in a single entry. Therefore, when determining the minimum number of least significant bits all ranges have to be taken into account. The specific implementation of this algorithm is out of the scope of this work.

4.6.4. Padding Compression

Unlike in the Diet-ESP strategy introduced by Migault et al. ([MGBS19]) we do not suggest the compression of the ESP Pad Length field. This has several reasons. First of all, we apply compression not only to the inner headers, but also to the outer IPv6 header. This could lead to a situation where the compute-* CDA implementation has trouble to reconstruct the ESP Pad Length field. Secondly, although the encryption algorithm and corresponding block size has to be known to the ESP implementation we suspect that it may not be as easily accessible as the more standardized SA information stored in the SAD and SPD. In order to reduce the cost of integrating SCHC into existing ESP implementations we have decided to forgo the compression of the ESP Pad Length field.

As an alternative, we suggest the use of encryption algorithms that do not require fixed block sizes to operate. One example is the use of Counter Mode (e.g. AES-CTR [Hou04]). In that case, the size of the ESP Padding field would be zero and the ESP Pad Length field could be safely eliminated by the compression.

4.7. Operation

This section introduces the basic operation principles of the concept introduced in this work. This includes adaptations to different scenarios and modular improvements to compression efficiency. Additionally, the complete process of rule creation is outlined.

4.7.1. Modes of Operation

In Section 4.6.1 we introduce the Context Parameters used to generate SCHC Rules and specify their origin as either 'In SA' or 'Negotiated'. Parameters marked as 'Negotiated' are not part of the IPSec SA and have to be agreed upon by some other means. In this section we propose three modes of operation that differ mainly in the way they implement the 'negotiation' of these Context Parameters. The modes are defined as follows:

- **strict-mode:** In strict-mode only context information gathered from the IPsec SAs is used for compression. This means that all Context Parameters marked as 'Negotiated' are unspecified and corresponding Rules are subject to Rule Downgrading (cf. Section 4.6.3).
- **preset-mode:** In preset-mode Context Parameters marked as 'Negotiated' are set to default values that must be identical between all endpoints. These Context Parameters are set globally and are equivalent for all SAs in operation. In practice, preset values could be supplied via a default configuration allowing for some flexibility.
- **sync-mode:** In sync-mode Context Parameters marked as 'Negotiated' are synchronized during the IKE Initial Exchanges. Therefore, IKE has to be adapted to allow the IKE_NOTIFY payload to be used to negotiate the values of these parameters. Context Parameters negotiated in sync-mode are only valid for the IPsec SA they were negotiated in.

Preset-mode Parameters

For use in combination with preset-mode we propose the following default values (s. Table 4.7) for the Context Parameters introduced in Section 4.6.1. 'esp_spi_lsb' should be set to '4' limiting the number of bits sent in the ESP SPI field to four. 'esp_sn_lsb' should be set to '4' limiting the number of bits sent in the ESP Sequence Number field to four. This means both SPI and Sequence Number are transmitted partially and still retain some of their operational value. 'ip_tc' represents the value of the IPv6 Traffic Class field should be set to '0'. This indicates that no Explicit Congestion Notification (ECN) is available ([RFB01]) and the default Per-hop Behaviour (PHB) is used ([NBBB98]). 'ip_fl' represents the value of the IPv6 Flow Label field and should be set to '0'. This indicates that the packet has not been labelled as part of any flow as specified in the IPv6 Flow Label Specification ([ACJR11]). 'ip_hl' represents the value of the IPv6 Hop Limit field and should be set to '255'.

Table 4.7.: Default values for the Context Parameters in preset-mode.

Context Parameter	Preset Value
esp_spi_lsb	4
esp_sn_lsb	4
ip_tc	0
ip_fl	0
ip_hl	255

4.7.2. Initiating Compression

In the context of this work, we propose that the use of compression is negotiated during the IKE Initial Exchanges in the same manner as specified by Migault et al. (cf. Section 3.2.5). Therefore, IKE Notify Payloads are used during the Initial Exchanges to indicate the use of compression in the negotiated SA. For the purpose of negotiation, the strategy proposed in this work can be considered an EHC Strategy.

The USE_COMPRESSED_MODE Notify payload is extended to support the additional modes strict-mode, preset-mode and sync-mode. These must be used instead of the modes 'Compressed Tunnel Mode' and 'Compressed Transport Mode' when negotiating this strategy. As described in Section 4.7.1 strict-mode and preset-mode do not require negotiation of Context Parameters. To enable the use of sync-mode the EHC_STRATEGY_SUPPORTED Notify Payload has to be extended to include the Context Parameters marked as 'Negotiated' (cf. Section 4.6.1).

4.7.3. Identifying Network Roles

As stated in Section 3.1 a SCHC implementation needs to be able to distinguish between Device and Network Infrastructure side. This information allows the use of the pseudo header fields labelled Dev* and App* (eg. DevIID, AppIID) instead of source and destination fields, thereby facilitating the use of single Rules for inbound and outbound traffic. To allow correct identification we propose the following options:

- Implementation: Since the SCHC and IPsec implementations used on the Device side are most likely different from those used on the Network Infrastructure side, Device identification can be provided during the implementation when integrating SCHC and IPsec. This means that the implementation on the Device side is aware of its role in the network.
- IPsec configuration: IPsec implementations may distinguish between communication endpoints by referring to them as 'left' and 'right' (e.g. libreswan, strongswan). For the purpose of this work we suggest the 'left' side be allocated to the Device. Therefore, device identification can be performed when scanning the IPsec configuration files.

4.7.4. End-to-end Encryption

To enable end-to-end encryption between two communication partners while maintaining header compression we need to consider additional restrictions. Figure 4.4 depicts a network similar to the one in previous sections with the addition, that we attempt to initiate end-to-end encryption between Device and Server. In this case all IKE traffic is encrypted between both endpoints and is merely forwarded by the Gateway. Since the latter is not part of the IKE exchange and has no knowledge about any SAs established between Device and Server, the Gateway is unable to derive any compression context. When subsequently attempting to transmit compressed ESP traffic between Device and Server, the Gateway is unable to decompress the outer IPv6 header and cannot forward the packet.

To solve this issue, we propose the following two methods:

- Skip compression of the outer IPv6 header. Using this method, no data preceding the ESP header is compressed allowing the Gateway to correctly process and forward the packet.

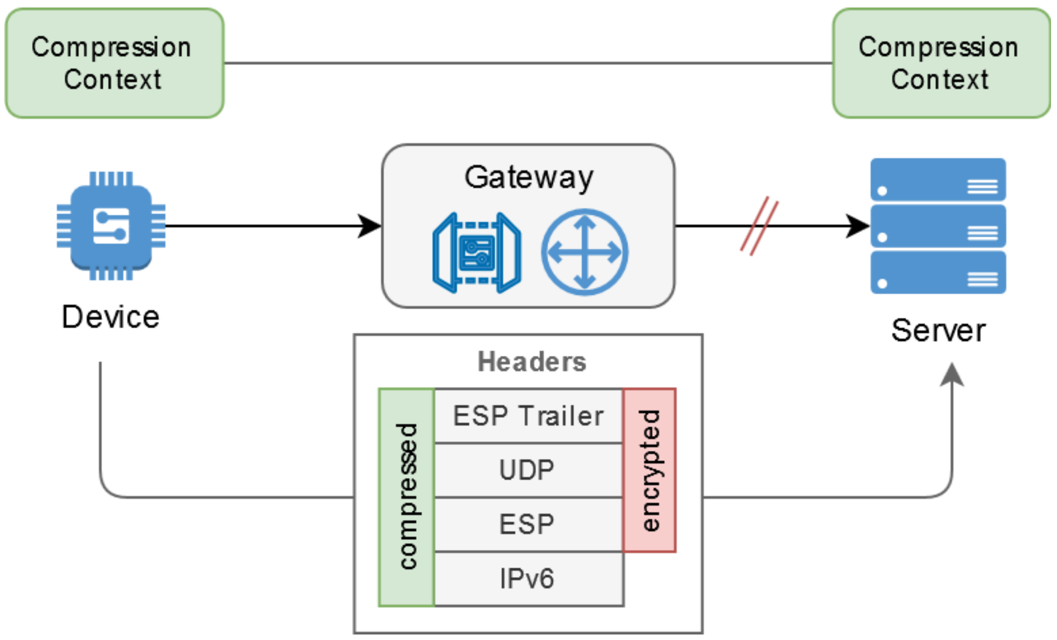

Figure 4.4.: Topology for end-to-end encryption: Device and Server share a common compression context established through IKE. When attempting to transmit a compressed packet, the Gateway is unable to decompress the outer IPv6 header and forward the packet.

- Inform the Gateway about the compression context used for the outer IPv6 header. In this method the Gateway has to be informed about the existing compression context by one of the communication partners.

The downside of the first approach is the loss of compression efficiency because the outer header(s) of the packet are transmitted without compression. However, the approach is considerably more flexible if, for example, end-to-end encryption between the device and a client several hops behind the server is desired. For additional considerations refer to Section 6.2.

4.7.5. Tunnel mode

When generating a Rule for a 'tunnel mode' SA there have to exist separate FDs for the inner and outer IPv6 headers. However, different use cases for 'tunnel mode' SAs require further considerations. The following use cases have to be considered:

- Tunnel mode is used instead of transport mode, but the destination of the inner IPv6 packet is identical to the destination of the outer IPv6 packet. This type of ESP encapsulation is referred to as self-encapsulation. As tunnel mode is considered the standard mode of operation for IPsec, this can be expect to be the case frequently.

- Tunnel mode is used to establish a VPN and therefore the destination of the inner IPv6 packet may not be identical to the destination of the outer IPv6 packet.

In some cases, two separate Rules are required because the indicated use of tunnel mode may not limit the SA to either of the instances described above. This is for example the case, if the selector of the SA specifies a range of addresses that includes the tunnel endpoint itself. To allow for this kind of flexibility, rule generation needs to provide two Rules if the SA is not explicitly limited to a single one of the cases outlined above. The self-encapsulation Rule has identical FDs for the inner and outer IPv6 header except for the 'IPv6 Next Header' FD. For this descriptor, the combined Context Parameters 'l4_proto' and 'ipsec_mode' indicate diverging Target Values, more specifically ESP(50) for the outer header and UDP(17) for the inner header. In the VPN Rule, the 'IPv6 AppPrefix' and 'IPv6 AppIID' FDs are also different for the outer header and need to have their Target Values replaced with the 'ip6_tun_prefix' and 'ip6_tun_iid' Context Parameters. Note that the actual values for the IPv6 Payload Length field differ between inner and outer IPv6 header because the payload length from the outer header's perspective includes the ESP header and encapsulated payload. However, this does not require any changes to the actual FDs since the implementation of the compute-* CDA is expected to generate correct values from either header's perspective.

4.7.6. Rule Creation Process

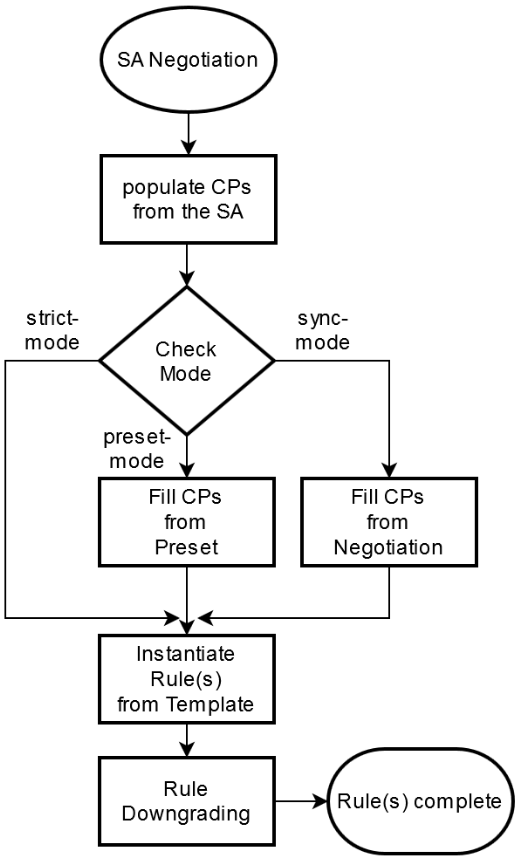

Figure 4.5.: Overview of the Rule creation process.

In this section, the rule creation process established in this chapter is summarized. Note that the automation of the process is out of the scope of this work. Rule creation starts with the SA negotiation using IKEv2. From the context information in the SA, Context

Parameters (CPs) are populated as described in Section 4.6.1. In the next step, Context Parameters marked as 'Negotiated' are filled depending on the mode of operation. The mode of operation is negotiated during the IKE exchanges as described in Section 4.7.2. In preset-mode, the parameters are filled from the presets introduced in Section 4.7.1. In sync-mode, the parameters are filled from the additional context negotiated using IKE. For strict-mode, no additional parameters are used. Using all Context Parameters available, one or more Rules are instantiated from the Rule Templates introduced in Section 4.6.2. Finally, any FDs that have either unspecified or ambiguous Context Parameters are subject to Rule Downgrading as proposed in Section 4.6.3.

4.7.7. Differences between Oirignal and Decompressed Packet

During decompression, some of the header fields may not be reconstructed with their original values. This is intended and due to nature of the Rule Templates introduced in Section 4.6.2. For the IPv6 header fields Traffic Class, Flow Label and Hop Limit, the Matching Operator 'ignore' is indicated in the Rule Template in combination with the 'not-sent' CDA. This leads to the reconstruction of the fields from their local Target Values without considering their original value in the packet. This is possible because we believe that these fields can be assumed to carry their default values without impairing IPv6 operation in an IoT context and is intended to improve compression efficiency. Note that this process has no impact on the validity of the ICV when considering the inner headers. This is true because the ICV value is calculated after compression. For further considerations and the specific default values please refer to the relevant documents ([RFB01], [ACJR11], [NBBB98]).

4.8. IKEv2 Compression

So far, only traffic following the installation of an ESP SA has been considered for compression. However, when employing header compression to ESP traffic the initial IKE communication, which establishes the context we use to derive compression context from, can be expected to present a considerable traffic overhead. Therefore, in this section, we discuss possibilities to reduce the size of the IKE Initial Exchanges. The following section focuses on methods that provide compression for the IKE Initial Exchanges using SCHC. Section 4.8.2 introduces a different approach as proposed by V. Smyslov in [Smy20].

4.8.1. Using SCHC

As described in section Section 2.4.3, the IKE Initial Exchanges consist of several headers and payloads in a predetermined order that are transmitted to initiate new IPsec Security Associations. Based on the predictive nature of IKE exchanges, we can make several assumptions about the structure of the exchange based on the exchange type. Furthermore, if no IKE SA has been established yet, we can expect the first IKE exchange to be of the type IKE_SA_INIT.

Since SCHC works on the basis of a static compression context the first step is to identify which information about the Initial Exchanges can be expected for both endpoints before beginning communication. Table 4.8 elaborates compression of the outermost IKE header. In an IOT context, field length for initiator and responder SPIs could be reduced on a case by case basis and are marked for a LSB CDA. Since the order of the headers in the initial exchanges are predetermined and the Exchange Type field identifies is sufficient to identify the next payload, the Next Payload field can be omitted. The IKE version fields are set to their fixed values for IKEv2 and are not transmitted. Since only four Exchange Types exists, the field can be eliminated if we generate a separate Rule for every type. The Flags field can be omitted since Initiator and Responder roles are fixed. The Message ID field is not compressed since it hardens the protocol against message replay attacks [KHN+14].

Table 4.8.: Candidates for SCHC compression in the IKE header fields during *IKE_SA_INIT*.

Header Field	Compression	Mode
IKE SA Initiator's SPI	Yes	LSB
IKE SA Responder's SPI	Yes	LSB
Next Payload	Yes	not-sent
MjVer	Yes	not-sent
MnVer	Yes	not-sent
Exchange Type	Yes	not-sent
Flags	Yes	not-sent
Message ID	No	value-sent
Length	No	value-sent

Similarly, the Next Payload and several reserved header fields in the SA, Key Exchange, Nonce, Auth, and Traffic Selector Payload can be omitted. However, these payloads do not offer significant amounts of static context that we can leverage for compression.

Context Parameters

The Context Parameters for the IKE header are shown in Table 4.9. Unlike the Context Parameters defined in previous sections, negotiation of these parameters can, for obvious reasons, not be performed during the IKE Initial Exchanges. Instead, parameters marked as 'Negotiated' have to be provided by some form of configuration or have to be negotiated by some other means. Parameters marked as 'Fixed' can not be altered through configuration or negotiation.

'ike_spi_i' indicates the Initiator SPI of the IKE SA which is generated before the Initial Exchanges and subsequently stored in the IKE SA. 'ike_spi_i_lsb' specifies the number of least significant bits that are transmitted instead of the Initiator SPI. Any compression that is applied to the SPI this way, effectively shortens to relevant SPI value to the number of bits indicated. 'ike_spi_r' indicates the Responder SPI of the IKE SA which is generated by the Responder before the Initial Exchanges and is stored in the IKE SA after the IKE_SA_INIT. 'ike_spi_r_lsb' specifies the number of least significant bits that are transmitted instead of the Responder SPI. 'ike_version' specifies the version of the IKE protocol that is used. This value is fixed to '2.0' indicating the use of IKEv2. The Context Parameters and Rule Templates introduced in this section are only applicable to this version. 'ike_role' indicates the role in the IKE exchange and can be either Initiator or Responder.

Table 4.9.: Context Parameters for the IKE Header

Context Parameter	Source	Possible Values
ike_spi_i	In SA	IKE SA Initiator's SPI
ike_spi_i_lsb	Negotiated	1..32
ike_spi_r	In SA	IKE SA Responder's SPI
ike_spi_r_lsb	Negotiated	1..32
ike_version	Fixed	2.0
ike_type	Negotiated	any Exchange Type
ike_role	Negotiated	Initiator or Responder

Rule Template

The Rule Template for the IKE header is shown in Table 4.10. The Initiator end Responder SPI fields can be reconstructed from the 'ike_spi_i', 'ike_spi_i_lsb' and 'ike_spi_r', 'ike_spi_r_lsb' Context Parameters. The 'lsb' parameters effectively shorten the SPIs to the indicated length and reconstruction makes sure they match the expected values in the SA. The Next Payload field can be reconstructed from the 'ike_type' field which indicates the IKE Exchange Type. Possible values represent the four available exchange types which are the Initial Exchanges IKE_SA_INIT and IKE_AUTH as well as the CREATE_CHILD_SA and INFORMATIONAL exchanges. From the template, separate Rules for each exchange type have to be generated to allow compression of the field. The version fields MJVer and MnVer are reconstructed from the 'ike_type' Context Parameter. The Exchange Type field can be reconstructed from the 'ike_type' Context Parameter and is subject to the same restrictions as the Next Payload field. The Flags field can be reconstructed from the 'ike_role' Context Parameter. The Message ID and Length fields are not compressed.

Table 4.10.: Rule Template for the IKE Header.

FID	FL	FP	DI	Target Value	MO	CDA
Initiator's SPI	64	1	Bi	ike_spi_i, ike_spi_i_lsb	MSB	LSB
Responder's SPI	64	1	Bi	ike_spi_r, ike_spi_r_lsb	MSB	LSB
Next Payload	8	1	Bi	ike_type	equal	value-sent
MjVer	4	1	Bi	ike_version	equal	not-sent
MnVer	4	1	Bi	ike_version	equal	not-sent
Exchange Type	8	1	Bi	ike_type	equal	not-sent
Flags	8	1	Bi	ike_role	not-sent	
Message ID	32	1	Bi	-	ignore	value-sent
Length	32	1	Bi	-	ignore	value-sent

Due to a lack of candidates for compression in the other IKE headers, we have decided not to pursue the option of applying SCHC compression to the IKE exchanges any further. For details and alternatives refer to the following section as well as Section 6.1.5.

4.8.2. Using Common Algorithms

In [Smy20] V. Smyslov proposes the use of standard lossless compression algorithms to reduce the size of IKE traffic. For this purpose the Compressed payload shown in Figure 4.6 is introduced. The Next Payload field identifies the payload type of the next payload in the message. The First Payload field identifies the payload type for the first payload in the Compressed Payloads field. The Algorithm field specifies the compression algorithm used to compress the payloads contained in the Compressed Payload field. Applicable algorithms are identical to those used in IPComp ([SMPT01]) and include deflate, LZS and LZJH.

```
                    1                   2                   3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Next Payload  |C|  RESERVED   |         Payload Length        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| First Payload |   Algorithm   |                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+                               |
|                      Compressed Payloads                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 4.6.: IKE Compressed payload header as proposed in [Smy20]

In the IKE_SA_INIT exchange, the Compressed payload is used to encapsulate the other IKE payloads in the way specified in Figure 4.7. This excludes the Nonce Payload and several optional payloads. For the sake of brevity and to be consistent with the exchange detailed in Section 2.4.3 the optional headers have been omitted. The encapsulation is identical for both Initiator and Responder.

```
Initiator                                   Responder

HDR, C!{SA, KE}, Ni  -->
                          <--  HDR, C!{SA, KE}, Nr
```

Figure 4.7.: Compressed IKE_SA_INIT exchange without optional payloads as proposed in [Smy20]

For the subsequent exchanges inlcuding IKE_AUTH, Smyslov proposes that the Compressed payload can be used in a similar way. However, compression has to be applied before encryption since the random nature of encrypted data stifles the efficiency of compression.

5. Implementation

As part of this work and in collaboration with other students at the Ludwig-Maximillians-Universität München, implementations of SCHC, IKEv2 and ESP for RIOT OS have been developed. IKEv2 for RIOT OS has been primarily developed by Tobias Heider [Hei17][Hei19] and Marinus Enzinger [Enz19]. ESP for RIOT OS has been primarily developed by Maximilian-Mathias Malkus [Mal19]. SCHC for RIOT OS has been developed by the author of this work.

This chapter introduces the parts needed for the proof-of-concept implementation as well as the hardware and setup it was designed for. RIOT OS has been introduced in Section 2.5 and is used as operating system for all Device side implementations. As a result of the modular nature of RIOT OS, all parts could bee implemented as modules and are designed to work independently.

5.1. Testing Environment

In this section we introduce the testing environment used to verify the operation of the individual modules as well as the strategies proposed in this work. While some of the implementations were targeted to the hardware described in Section 5.1.1, the integration process has not been completed for all modules at the time of writing. Therefore, the RIOT native hardware virtualizer has been used as primary test-bed.

5.1.1. Components

The target hardware for the various network components is listed in the following:

- Device: Nucleo F411RE
- LPWA technology: Semtech sx1272 LoRa module
- Gateway: Rapsberry Pi Model A Revision 2.0
- Server: generic Linux virtual machine (Linux Mint)
- non-IoT IPsec implementation: Libreswan

5.1.2. RIOT native

RIOT native is a basic hardware virtualizer which allows RIOT applications to be compiled and executed on a host operating system [BGH+18]. The virtualizer can emulate typical IoT devices consisting of a board, CPU, network interface and more. Communication between multiple RIOT native instances can be achieved through virtual Ethernet Interfaces (e.g. TAP interfaces).

In the context of this work, RIOT native has been used to emulate the hardware representing the Device. The connection between Device and Gateway has been provided through a tap network interface.

5.2. IKEv2 for RIOT OS

The IKEv2 implementation provided for this work was originally designed as 'g-ikev2' (group-ike) to perform group key management for Group Security Associations (GSAs) as proposed in [SW20]. The implementation is based on the prototype provided by Tobias Heider [Hei17] and was extended by Marinus Enzinger [Enz19] and the author. In the context of this work, the prototype is intended to provide the means to establish IKE SAs for ESP to allow testing of the SCHC integration within IPsec. For this purpose, the prototype represents a minimal implementation of IKEv2 which allows the negotiation of various preset SAs. The prototype has access to only a very limited selection of cryptographic algorithms including AES and HMAC_SHA1.

5.2.1. Initial Exchanges

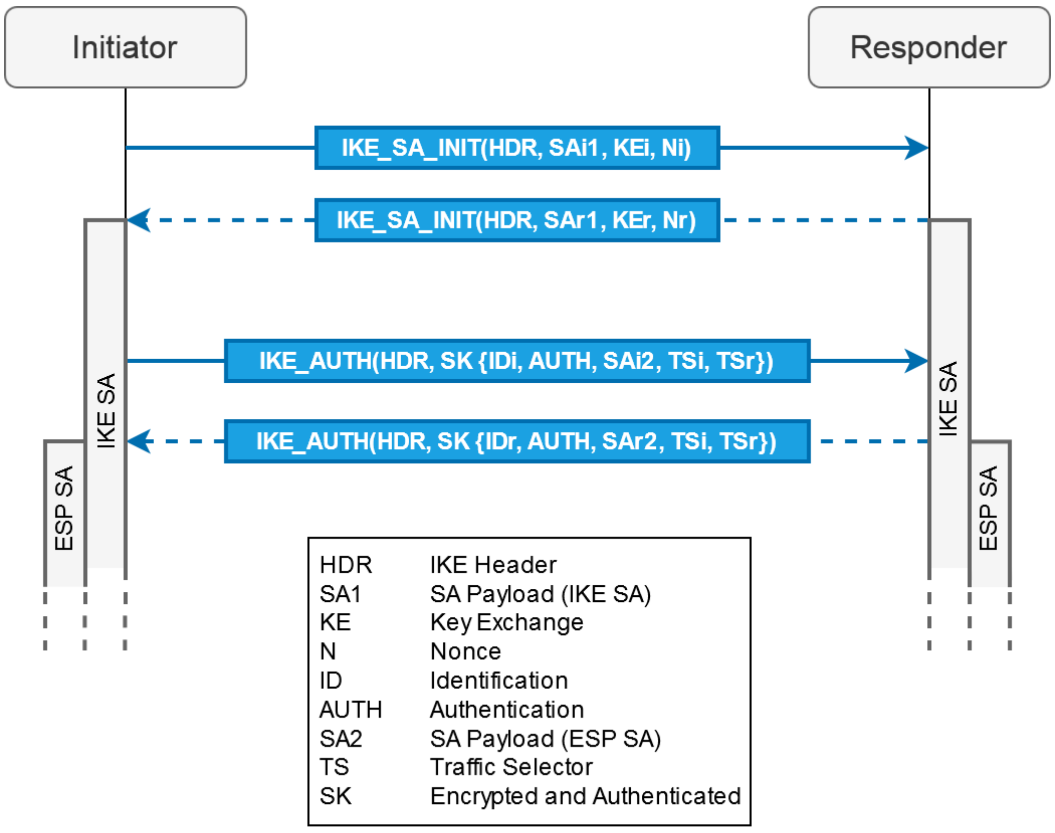
Initiator
Responder
IKE_SA_INIT(HDR, SAi1, KEi, Ni)
IKE_SA_INIT(HDR, SAr1, KEr, Nr)
IKE_AUTH(HDR, SK {IDi, AUTH, SAi2, TSi, TSr})
IKE_AUTH(HDR, SK {IDr, AUTH, SAr2, TSi, TSr})
IKE SA
ESP SA
IKE SA
ESP SA
HDR IKE Header
SA1 SA Payload (IKE SA)
KE Key Exchange
N Nonce
ID Identification
AUTH Authentication
SA2 SA Payload (ESP SA)
TS Traffic Selector
SK Encrypted and Authenticated

Figure 5.1.: Sequence diagram for the IKE Initial Exchanges.

The IKEv2 initial exchanges consist of the IKE_SA_INIT and IKE_AUTH exchange as shown in Figure 5.1. The IKE_SA_INIT exchange has been described in some detail in Section 2.4.3. The initial exchange for g-ikev2 is identical to the standard IKE_SA_INIT [SW20][KHN$^+$14]. The IKE_AUTH exchange is different from the GSA_AUTH exchange used by g-ikev2. During IKE_AUTH the Initator and Responder exchange information about identification (IDi, IDr), authentication (AUTH), Security Associations(SAi2, SAr2) and Traffic Selectors (TSi, TSr). Each information is contained in its own header format specified in [KHN+14]. The exchange is encrypted and integrity protected by the IKE SA established during IKE_SA_INIT. Section 5.2.2 details the process of key generation.

```
Initiator:
AUTH = prf( prf(Shared Secret (SK_pi), "Key Pad for IKEv2"),
<InitiatorSignedOctets>)

Responder:
AUTH = prf( prf(Shared Secret (SK_pr), "Key Pad for IKEv2"),
<ResponderSignedOctets>)
```

Figure 5.2.: Generating authentication data from a Pre-shared key in IKEv2 [KHN+14]

In the Identification Payloads the Initiator an Responder provide identification information. IKEv2 supports multiple different types of identification including IP addresses and fully-qualified domain names (FQDN). For the purpose of this prototype the ID_KEY_ID type is used which allows the use of an opaque octet stream. This means any string can be used for identification which allows for complete flexibility when choosing addresses in a test scenario. To allow successful identification the Responder side has to be provided with the expected ID of its communication partner during configuration. In the Authentication Payloads the Initiator and Responder prove knowledge of the secret associated with the corresponding ID. The prototype uses a prey-shared key (PSK) for authentication. The PSK is transmitted in the format described in Figure 5.2. 'prf' is the pseudo-random-function negotiated in the IKE_SA_INIT and the "Key Pad for IKEv2" consists of 17 ASCII characters. The last part of the key is padding which is used to allow the use of a password as shared secret without having to store the password in cleartext [KHN+14].

The Security Association Payloads are used to transmit the SA proposals. SA proposals consist of multiple Transform Substructures that indicate the protocol they are intended for (IKE, AH or ESP) and the cryptographic algorithms they intend to use. For this prototype the algorithms include an encryption algorithm (AES-CBC) and a separate integrity algorithm (AUTH_HMAC_SHA1_96). The specific transforms for this preset ESP SA proposal are shown in Listing 5.1. The size of the Transform Payload indicated by the 'length' fields are computed during operation. The 'trans_type' variable holds the Transform Type value for encryption algorithm (ENCR), integrity algorithm (INTEG) and extended sequence numbers (ESN) respectively. The 'trans_id' variable is used to store the Transform IDs which indicate the algorithms specified in their respective transforms. The number of Attributes in the Transform is indicated by 'attr_num'. In this case, Attributes are only used to specify the key length for the indicated encryption algorithm which is set to 128 bits. The ESN transform with a Transform ID of zero indicates that normal (non-extend) sequence numbers are used.

In the Traffic Selector Payloads, Initiator and Responder negotiate on the specifics of the packet flow that needs to be protected by IPsec. Each Traffic Selector (TS) may specify a port range and address range. The Traffic Selector transmitted by the Responder can be a subset of the Initiator's TS. In that case, the Responder increases the granularity of the negotiated SA. The use of transport mode is negotiated by an additional USE_TRANSPORT_MODE notification from the Initiator.

Listing 5.1: Transform configuration for the ESP proposal in ike_config.h

```
transform_t transforms_esp[] = {
  {
    .trans_type = TR_ENCR,
    .trans_id = ENCR_AES_CBC_ID,
    .attr_num = 1,
    .length = 0,
    .attr_list = attributes,
  },
  {
    .trans_type = TR_INTEG,
    .trans_id = AUTH_HMAC_SHA1_96_ID,
    .attr_num = 0,
    .length = 0,
  },
  {
    .trans_type = TR_ESN,
    .trans_id = 0,
    .attr_num = 0,
    .length = 0,
  },
};
```

5.2.2. Deriving Keys

```
prf+ (K,S) = T1 | T2 | T3 | T4 | ...

where:
T1 = prf (K, S | 0x01)
T2 = prf (K, T1 | S | 0x02)
T3 = prf (K, T2 | S | 0x03)
T4 = prf (K, T3 | S | 0x04)
...
```

Figure 5.3.: Generating keying material for IPsec as introduced in [KHN+14]

IKE derives all keys from a quantity call SKEYSEED which is available to both communication partners after the IKE_SA_INIT exchange. Keying material is produced by the negotiated pseudo random function (PRF) algorithm. SKEYSEED itself is derived from the concatenation of nonces that were transmitted during the exchange. From SKEYSEED, the following seven additional secrets are derived: SK_d, SK_ai, SK_ar, SK_ei, SK_er, SK_pi, SK_pr. SK_d is used to generate additional keying material for use in Child SAs. The other secrets are used for integrity protection and encryption/decryption of the subsequent exchanges. SK_pi and SK_pr are required to generate the AUTH payload. Figure 5.3 shows the process of deriving keys using the PRF. Prf+ (prf-plus) indicates the iterative use of the

PRF algorithm to generate keying material larger than the size of the output of the PRF. Prf+ generates keying material as an octet string of infinite length. K and S are specified based on the keying material that is to be generated. To generate keys, output matching the length of the generated key is taken from prf+. Subsequent key generation resumes exactly where the previous key ended.

Listing 5.2: Generating keying material for the IKE SA and ESP SA in ikev2.c

```
1  /* Get Keys for IKE SA*/
2  prf_plus_ctxt ctxt;
3  prf_plus_init(&ctxt, sa->keymat.sk_seed, nonces_spis,
4  sa->len_nonce_i+sa->len_nonce_r + 2*sizeof(uint64_t));
5
6  prf_plus_read(&ctxt, sa->keymat.sk_d , 20);
7  prf_plus_read(&ctxt, sa->keymat.sk_ai, AUTH_KEYSIZE);
8  prf_plus_read(&ctxt, sa->keymat.sk_ar, AUTH_KEYSIZE);
9  prf_plus_read(&ctxt, sa->keymat.sk_ei, 16);
10 prf_plus_read(&ctxt, sa->keymat.sk_er, 16);
11 prf_plus_read(&ctxt, sa->keymat.sk_pi, 20);
12 prf_plus_read(&ctxt, sa->keymat.sk_pr, 20);
13
14 /* Get Keys for CHILD SA */
15 prf_plus_ctxt ctxt2;
16 prf_plus_init(&ctxt2, sa->keymat.sk_d,
17 nonces_spis, sa->len_nonce_i+sa->len_nonce_r);
18
19 prf_plus_read(&ctxt2, sa->aes_key , 16);
20 prf_plus_read(&ctxt2, sa->hmac_key, 20);
```

The additional secrets (SK_d, etc.) are derived from SKEYSEED through iterative use of: $prf+(SKEYSEED, Ni|Nr|SPIi|SPIr)$ (| indicates concatenation). Listing 5.2 shows the generation of keying material in the prototype. Lines 2 through 12 show the use of prf+ to generate the secrets needed in the IKE SA. As stated in Section 5.2.1 the prototype negotiates for the use of AES-CBC and HMAC-SCHA1-96 as encryption and integrity algorithms for the Child (ESP) SA. Lines 15 through 20 show the derivation of keys for the ESP SA from SK_d. The following keys are generated: a 128-bit key for AES-CBC and a 160-bit key for HMAC-SCHA1-96.

5.3. ESP for RIOT OS

The ESP prototype for RIOT OS is based on the implementation by Maximilian-Mathias Malkus [Mal19]. Unlike the IKE implementation, the ESP prototype has been integrated directly into RIOT's IP protocol stack. The IP stack used in RIOT OS, named GNRC, has been covered briefly in Section 2.5. The ESP implementation is integrated inside of the demultiplexing routine that handles IPV6 extension headers. Listing 5.3 shows the code segment where ESP traffic handling is introduced. The packet is transferred to the ESP subroutine in line 3. To allow integration of ESP packet handling into GNRC several other adjustments had to be made. For details please refer to [Mal19].

Listing 5.3: Integration of ESP processing into the GNRC network stack in gnrc_ipv6_ext.c

```
1          case PROTNUM_IPV6_EXT_ESP:
2  #ifdef MODULE_GNRC_IPV6_IPSEC
3          pkt = esp_header_process(pkt, protnum);
4          if( pkt == NULL) {
5                  DEBUG("ipv6_ext: Rx esp header processing
6                  failed or pkt was consumed
7                  by ext handling\n");
8                  gnrc_pktbuf_release(pkt);
9                  return NULL;
10         }
11         break;
12 #endif /* MODULE_GNRC_IPV6_IPSEC */
```

5.3.1. SA and Key Management

As described in Section 2.4 ESP packet processing is reliant on an internal data-structure representing the IPSec SA referred to as SPD and SAD. In most IPsec implementations security policies are handled by the kernel. To access information stored in the IPsec SAs from user-space, there has to exist a way to communicate with the IPsec kernel module. This is ordinarily handled through some Application Programming Interface (API). On Unix based systems, common APIs are Netlink xfrm and pf_key. These APIs interact with the kernel, usually using some form of IPC (e.g. Netlink), to store and retrieve information from the IPsec databases. Xfrm in particular uses virtual interfaces that allow it to transform packets to for example apply encryption to the payload. The RIOT ESP prototype was originally designed to include a pf_key implementation. However, because of the serious development overhead and limited standardization of the API, pf_key has not been implemented for the ESP prototype. At the time of writing, no standardized access to the IPsec databases in the form of either pf_key or xfrm exists in the prototype. Instead, the SAD and SPD can be accessed through a custom tool named 'dbfrm'. Dbfrm has been introduced by Maximilian-Mathias Malkus [Mal19] and has been extended for use in this prototype.

To install an SA, dbfrm can be provided the following parameters:

- action: Can be either PROTECT, BYPASS or DISCARD to indicate the processing choice for the SPD.
- id: A unique numerical identifier assigned to the SPD.
- spi: The SPI value assigned to the SPD.
- dst: The IPv6 destination address for the SA.
- src: The IPv6 source addres for the SA.
- proto: The protocol value for the next header following IPv6.
- port_dst: The destination port for the Layer 4 protocol.
- port_src: The source port for the Layer 4 protocol.
- mode: The IPsec mode of operation. Can be either 'tunnel' or 'transport'.

- c_mode: The cipher mode used in the SA. Can be either 'auth', 'authenc', or 'comb', indicating the use of an authentication, authentication and encryption or combined algorithm respectively.
- auth: The authentication algorithm to be used in the SA. Can be either 'none' or 'sha'.
- hash_key: The key used by the authentication algorithm. Note that due to the nature of static memory allocation the maximum supported key length is 32 bytes.
- enc: The encryption algorithm to be used in the SA. Can be either 'none', 'aes', 'chacha', or 'mockup'. Mockup indicates the use of a NULL encryption algorithm that does not actually perform any encryption. However the implementation of the mockup cipher is not entirely faithful to the specifications in [GK98]. Note that only the AES and mockup ciphers have been implemented at the time of writing.
- enc_key: The key used by the encryption algorithm. Note that due to the nature of static memory allocation the maximum supported key length is 32 bytes.
- iv: The Initialization Vector (IV) for the encryption algorithm.
- t_dst: The IPv6 destination address for the ESP tunnel, if tunnel mode is used.
- t_src: The IPv6 source address for the ESP tunnel, if tunnel mode is used.

An example for SA generation using dbfrm is given in Section 5.5.3. Note that dbfrm processes data for the SPD and SAD simultaneously. At the time of writing, dbrfm does not support the use of address or port ranges when creating SAs. This is due to limitations of the internal data structure holding the IPsec databases that could not be resolved within the scope of this work.

As part of the prototype implementation and to allow quick configuration of IPsec SAs, several preset SPD entries are hard-coded. The entries specify the processing of loopback and ICMP traffic, as well as other traffic that is not specified to be PROTECTed. The default processing for ICMP and loopback traffic is set to BYPASS. The default processing for other traffic is set to DISCARD.

5.3.2. ESP Processing

The actual processing of ESP packets is handled in two separate methods used on outgoing and ingoing traffic respectively. The declarations for both methods are shown in Listing 5.4. For outgoing traffic, 'esp_header_build' is invoked. Processing differentiates between tunnel and transport mode. In tunnel mode the packet is encapsulated and a new outer IPv6 header is generated. In transport mode, only the Layer 4 header and payload are encapsulated. In the next step the ESP header fields are populated from the SA. Encryption takes place after encapsulation and is performed on the ESP payload including padding and the ESP trailer. The alignment for padding depends on the block size of the encryption algorithm. In the case of AES-128, the block size is 16 bytes. Finally, the integrity protection algorithm (HMAC-SHA1) generates the ICV and the packet is handed back to the GNRC stack.

For inbound traffic, 'esp_header_process' is invoked. In the first step the SA corresponding to the SPI from the received ESP packet is identified. Subsequently, the packet is authenticated using the ICV and decrypted if authentication was successful. Finally any encapsulation is removed. For tunnel mode, the original IPv6 packet is relayed via *gnrc_netapi_dispatch_receive* and processing ends. For transport mode, the layer 4 header and payload are passed on to the GNRC stack.

The AES-128 algorithm and Cipher-Block-Chaining (CBC) mode are provided by the

Listing 5.4: Declaration of the ESP processing methods for ingoing and outgoing traffic in esp.h

```
/**
* @brief   Build ESP header for sending
*
* @param[in] pkt   head after IPv6 header build
* @param[in] sa_entry   Database pointer to according
*             Security Association (SA) entry
* @param[in] ts   Pointer to Traffic Selector (TS) of pkt
*
* @return  pktsnip at IPv6 with ESP
*/
gnrc_pktsnip_t *esp_header_build(gnrc_pktsnip_t *pkt,
        const ipsec_sa_t *sa_entry,
        ipsec_ts_t *ts);

/**
* @brief   Marks, Decrypts and returns pkt at next header.
If the ipsec rules dictate tunnel mode,
packet is consumed and processed.
*
* @param[in] pktsnip at ESP EXT header
*
* @return  processed ESP pkt at next header poisition
* @return  NULL on tunnel mode
*/
gnrc_pktsnip_t *esp_header_process(gnrc_pktsnip_t *pkt,
        uint8_t protnum);
```

RIOT OS *Crypto* module. The HMAC-SHA1 algorithm is part of the RIOT OS *Hashes* module. The decision to limit the prototype to a single encryption algorithm can be attributed to the *Crypto* module which supports only the ciphers AES-128, 3DES and NULL.

5.4. SCHC for RIOT OS

This section describes the implementation of the SCHC module for RIOT OS. The implementation can be divided into the following submodules: header parser, rule manager, decompressor, compressor and bit-buffer. Each submodule is covered in detail in the following sections.

5.4.1. Header Parser

The header parser is responsible for translating header data into a memory format that allows for easy identification of the respective header fields. The parser takes uncompressed header data as input. The parser is also able to generate preset IPv6 and UDP headers for testing purposes. Furthermore, the parser must detect faulty packets or packets that do not meet the requirements. In this case, further processing is rejected and the SCHC compression is aborted. Requirements for successful packet parsing are the limitation of Transport Layer protocols to only UDP.

The memory model for parsed headers allows the addressing of each header field by its name. This is achieved through the use of an 'enum' that holds values for all supported header fields.

5.4.2. Rule Manager

The rule manager is responsible for managing and generating SCHC Rules. The Matching Operators described in Section 3.1.2 are implemented here. Additionally, the rule manager can generate rules for testing purposes. The main task of the rule manager is to identify applicable rules given a set of successfully parsed headers. As per SCHC specification the rule selection goes through the following steps for each Rule in the Rule-set:

- For each Field Descriptor (FD) check if a corresponding header field is present in the parsed header data. If no Rule matching the exact layout of the parsed header fields is found the process fails. This includes cases where the parsed header data contains header fields for which no FD exists.
- For each FID check if there exists a FD witch matching Direction Indicator (DI).
- For each FD check if the position of the header field in the parsed header data corresponds to the Field Position (FP). For the purpose of this implementation FPs are only allowed to hold the value '1'.
- For each FD the corresponding MO is applied to the Field Value (FV) and Target Value (TV). If all MOs return as 'true' a matching Rule has been found.

If multiple Rules match the parsed header data, the first Rule is selected. More specifically the rule selection process terminates once the first match has been found.

5.4.3. Bit-Buffer

Since SCHC does not require the Compression Residue to consist of multiples of one byte, a custom data structure able to read and write data from/to a stream of bits is required. The bit-buffer supports two operations: *append* and *get*. *Append* is used to add Compression Residue produced by a single FD to the buffer. The Compression Residue can have arbitrary length. Compression Residues are directly concatenated without any padding or byte alignment required. The *get* operation is used to retrieve the Compression Residue of a single FD from the buffer.

5.4.4. Compressor

The compressor sub-module implements all Compression Actions (with some limitations) from the set of CDAs described in Section 3.1.2. Given parsed header data and a matching SCHC Rule, the compressor generates and concatenates Compression Residues for all FDs in the Rule in a bit-buffer. Any payload following the compressed headers is appended to the Compression Residue without padding.

5.4.5. Decompressor

The decompressor sub-module implements all Decompression Actions (with some limitations) from the set of CDAs described in Section 3.1.2. Given a bit-buffer holding Compression Residue and the matching SCHC Rule, the decompressor is able to reconstruct the original header data from the residue.

5.4.6. SCHC Processing

This section describes the specifics to processing of SCHC compressed traffic. Therefore, we have to distinguish between the compression and decompression sequences. The compression sequence is shown in Figure 5.4. The method takes an uncompressed network packet and the traffic direction as input. The traffic direction is represented by a SCHC Direction Indicator (DI) and can indicate either uplink (up) or downlink (dw). The direction is considered to be observed from the Device's perspective. An uplink DI, for example, means that the packet is travelling from the Device to the Gateway. The first processing step is to parse the headers into the internal data format needed to perform further processing. In the second step, a viable Rule from the existing Ruleset is identified. If any of the previous steps fail, SCHC processing is aborted. More specifically for this implementation the packet is most likely transmitted without compression. In the next step, a new packet is initialized containing the original Data Link Layer (l2) header and the RuleID needed for decompression. Subsequently, Compression Actions are applied to all header fields specified in the matching Rule and the Compression Residue is added to the packet. In the final step, the original (and uncompressed) payload is appended to the packet.

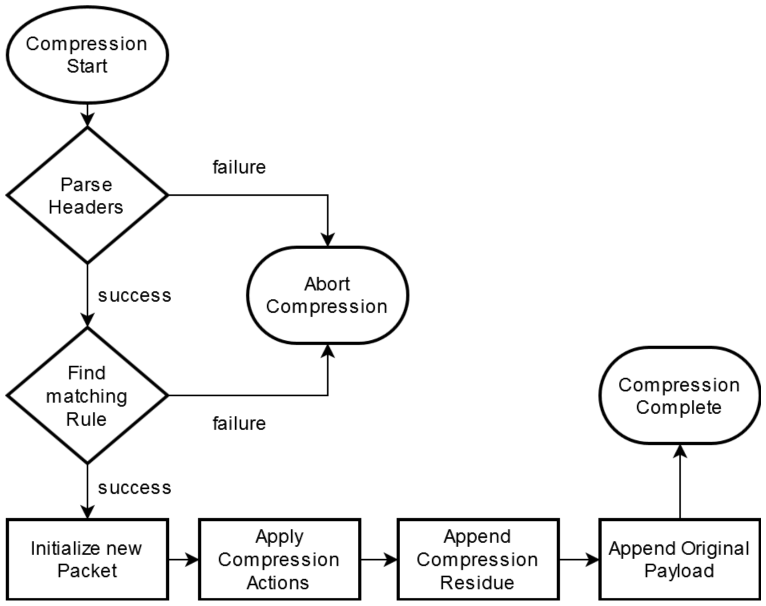

Figure 5.4.: SCHC Compression Sequence

The decompression sequence is shown in Figure 5.5. The method takes a compressed network packet and the traffic direction as input. The traffic direction is represented by a SCHC Direction Indicator (DI) as described above. The first processing step is to find a

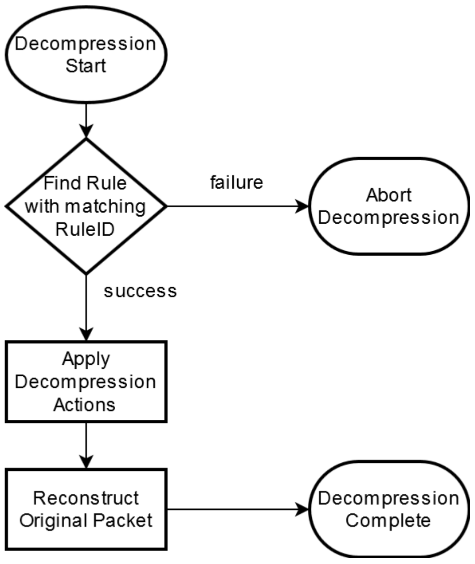
Decompression Start
Find Rule with matching RuleID
failure
Abort Decompression
success
Apply Decompression Actions
Reconstruct Original Packet
Decompression Complete

Figure 5.5.: SCHC Decompression Sequence

Rule witch matching RuleID in the local Rule-set. If no Rule with the corresponding RuleID is found, processing is aborted. In this case, the packet will be dropped. This is reasonable because a packet that was sent without compression would still need to carry a valid RuleID which indicates that no compression has been performed. In the second step, Decompression Actions are applied to the Compression Residue. Finally, the original packet is reconstructed by appending the decompressed headers and payload to the l2-header. This step may also remove the SCHC RuleID from the header stack.

5.4.7. Compute CDA

When processing multiple FDs with the compute-* CDA, a specific processing order may need to be observed. As an example, the UDP checksum is generated over a pseudo header that includes the IPv6 Payload Length field. Since this field is usually also compressed using the compute-* CDA as proposed in [MTG+20], computation of the Payload Length field has to occur first. Additionally, the compute-* CDA has to be postponed until all other CDAs from all FDs included in the Rule have been processed. This includes FDs that target entirely different headers. As the IPv6 header precedes the UDP header, processing in the order of occurrence leads to the desired result. However, we believe that this cannot be taken for granted.

To alleviate this problem, some process to allow the reordering of compute-* CDAs is required. Since the algorithms eligible for this CDA have to be known in advance, no synchronization between compression and decompression is required. The ordering of the compute-* actions can be determined solely from the implementations of the compute-* actions themselves. To provide the ability to process compute-* actions in a pre-determined order, the prototype generates a linked list containing all compute-* actions when processing the FDs. Additionally, a priority value is assigned to each action indicating the relative ordering of the different computation algorithms. When processing of the FDs, with the exception of the compute-* actions, is complete, the list is reordered according to the priority values. Subsequently, the compute-* CDAs are processed in the order indicated by the linked list. This ensures that the processing of compute-* actions is performed in the order that is required for them to perform their desired functions.

5.5. Proof of Concept

The Proof of Concept for this work specifies the interaction between the modules detailed in the previous sections. As described in Chapter 4 the integration of SCHC compression and decompression in ESP protected traffic requires that several IPsec modules interact with SCHC.

5.5.1. SCHC Integration

In normal SCHC operation, compression/decompression would only need to occur between the processing of Data Link and Network Layer (L2 and L3). However, since we apply compression to encrypted traffic, SCHC processing has to be performed in multiple phases. Figure 5.6 shows an overview of packet processing with ESP and SCHC compression enabled. The graphic can be read left to right as the processing of a single packet that is transmitted from the Device to a SCHC enabled endpoint. As stated in Chapter 4, the endpoint is expected to represent either a Gateway or a Server. The Network and Transport Layer (L3 and L4) protocols are represented by IPv6 and UDP respectively. In the figure, the complete protocol stack for both endpoints in tunnel mode is displayed. Any 'optional' Transport Layer payload would be placed at the top of the stack but has been omitted to reduce clutter. To represent transport mode processing, simply remove the processing step indicated as 'tunnel mode only'. The Data Link Layer protocol (L2) is not specified because the prototype uses a custom protocol. For Layer 2 protocol options refer to Section 6.4. Physical Layer (L1) communication is expected to include some form of LPWA technology (e.g. LoRa).

ESP processing is split into encapsulation, encryption and unwrapping, decryption respectively. ESP encapsulation represents the processing step where the ESP header is generated and the protected packet is encapsulated in the ESP Payload Data field by the sender. Analogously, ESP unwrapping represents the reversal of the process by the receiver. ESP encryption and decryption represent the processing steps where the encryption and integrity protection algorithms are implemented. SCHC processing is performed in two phases as proposed in Section 4.5.2. Plaintext phase processing has to occur between ESP encapsulation end ESP encryption for outbound traffic and between ESP decryption and ESP unwrapping for inbound traffic. Ciphertext phase processing has to be performed between Data Link and Network Layer processing. The placement is identical to normal SCHC operation. The centre of the figure shows the state of the packet as a concatenation of headers before or after specific processing steps. ESPH and ESPT denote the ESP header and ESP trailer as introduced in Section 2.4.1. IP and UDP denote the IPv6 and UDP header respectively. The symbol C indicates that compression has been performed on all header fields specified in the brackets. RID indicates the SCHC RuleID that is transmitted with the compressed packet.

In the following, the implementation specifics for SCHC integration are shown. Each SCHC processing phase (plaintext and ciphertext) has to be integrated into the protocol stack separately. For the Device side the RIOT OS modules introduced in the previous

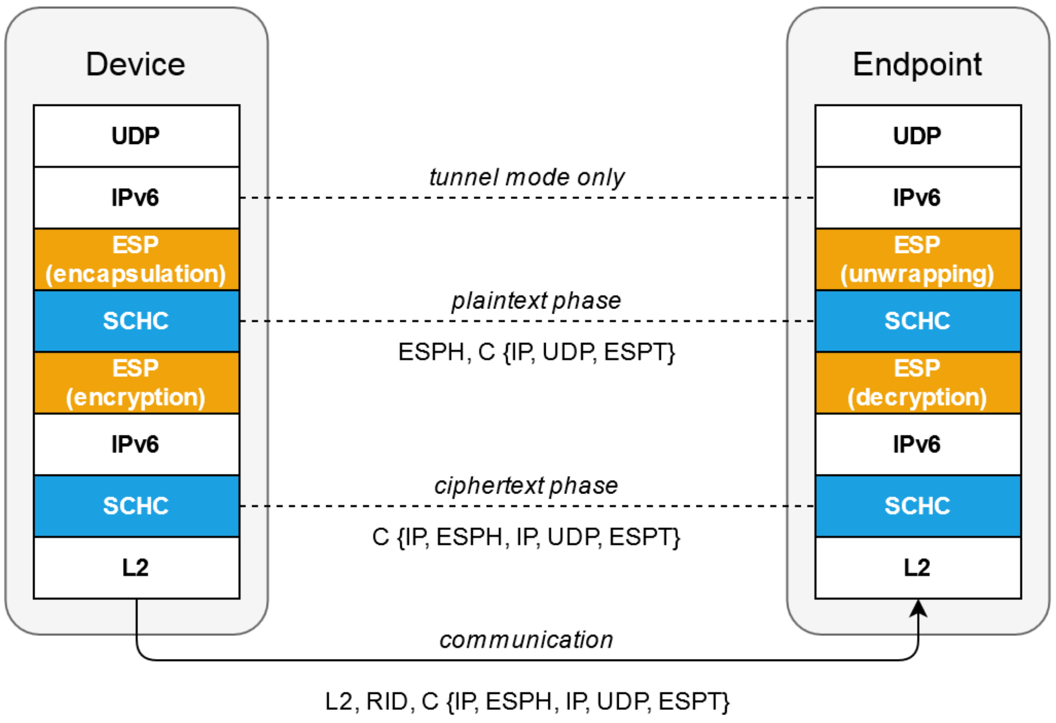

Figure 5.6.: SCHC integration into the IPsec stack: SCHC processing is integrated in two phases to allow compression of encrypted traffic.

sections are used. For the endpoint side, we use a Linux system with the libreswan IPsec implementation as described in Section 5.1.

Device

On the device, plaintext phase integration is implemented in the 'esp_header_build' and 'esp_header_process' functions introduced in Section 5.3.2. As discussed in the current section, the integration has to be implemented after encapsulation but before encryption for outbound traffic in 'esp_header_build', and after decryption but before unwrapping for inbound traffic in 'esp_header_process'.

Due to the use of a custom Layer 2 protocol in combination with LoRa, the ciphertext phase integration has been implemented in a test module controlling the operation of the LoRa driver. In practice, ciphertext phase integration could be implemented at any point between Layer 2 and Layer 3 processing. This includes integration into the IP stack or as extensions to the L2 or L3 (IPv6) processing. Note however that the fact that we have to insert a RuleID between the L2 and L3 header could lead to erroneous packet handling if any L3 processing is allowed to be performed before the id is removed by SCHC.

Endpoint

At the time of writing, SCHC integration on the endpoint has not been completed. However, points of integration for both processing phases have been identified and are described in the following. The plaintext phase has to be integrated into the ESP implementation. When using libreswan we have the option to use the Linux built-in Netkey/xfrm stack or the libreswan IPsec stack called KLIPS, which is intended for use in embedded systems. When using KLIPS, the ESP module can be found in the *./linux/net/ipsec/* subdirectory of the libreswan source code. For inbound traffic, SCHC integration can be performed in the 'ipsec_rcv_esp_post_decrypt' function in *ipsec_esp.c* which handles decapsulation of the ESP packet after decryption. For outgoing traffic, the SCHC integration can be implemented into the 'ipsec_xmit_esp' fucntion in *ipsec_xmit.c*, where ESP encapsulation and encryption are performed. For xfrm, the most likely candidates for plaintext phase integration are 'xfrm_input' in *xfrm_input.c* for inbound traffic and 'xfrm_output_one' in *xfrm_output.c* for outgoing traffic.

The ciphertext phase integration on an endpoint running Linux has to be implemented in a manner similar to the Device side implementation. Assuming the use of separate network interfaces for LPWAN and internet traffic, processing can be performed by installing a socket to listen for inbound/outgoing traffic on the LPWAN interface and processing traffic before returning it to the network stack. This can be achieved by using an AF_PACKET socket with type SOCK_RAW or SOCK_DGRAM. Integration into the L2 protocol implementation is also possible. For a discussion on L2 protocol selection refer to Section 6.4.

5.5.2. SCHC Processing

Up to this point, SCHC compression context has been represented by a single Rule that contains FDs for all fields in the header stack. Considering however that the actual SCHC processing is split into two phases that are integrated into the protocol stack separately, it becomes clear that storing compression context in a single SCHC Rule is not a practical option. Moreover, plaintext phase processing may not have access to the transmitted RuleID. This could be the case if the ciphertext phase processing has removed the RuleID from the packet, or if the plaintext phase implementation has access to only the encapsulated headers.

To alleviate the first problem, we split each SCHC Rule into the parts that correspond to the particular processing phase. This means that from a single Rule with FDs for all header fields present in the packet including the innner, ESP and outer headers, we create the following separate Rules:

- A plaintext phase Rule, that is comprised of the FDs for the inner headers and the ESP trailer.
- A ciphertext phase Rule, that is comprised of the FDs for the outer and ESP header(s).

Note that this requires some flexibility from the header parser in the SCHC implementation.

To accommodate the second problem, we need to be able to identify the corresponding decompression Rule without access to the transmitted RuleID. One obvious choice is to make use of the SPI in the SA. Because each Rule is targeted to a specific SA, we can identify the corresponding Rule by the SPI of the SA instead of the RuleID while in the plaintext processing phase. However, this limits each SA to a single Rule per phase. Therefore, we suggest that the implementation keep track of a global SCHC processing state for each SA that specifies which Rule needs to be applied for each phase. Access to this shared state from both the plaintext and ciphertext processing phase could be implemented in the form of shared memory or other synchronization methods. Also note that one could argue to omit the process of finding a matching Rule in the SCHC compression sequence (cf. Figure 5.4). This is possible because each Rule is expected to be specifically designed to match a single SA.

5.5.3. Example

In this section we provide an example for the complete Proof of Concept process. In the first step, we have to configure IKE to establish an SA according to our parameters. Listing 5.5 shows a complete configuration file for IPsec used in the prototype. Lines 2 and 3 specify the IPv6 addresses of both endpoints. As suggested in Section 4.7.3 we use left to represent the Device and right to represent the endpoint on the Network Infrastructure side, in this case the Gateway. Line 3 specifies the use of a shared secret which is stored in a separate file and has to be known by both communication partners. Line 7 specifies the identifier of the Device using the identification type ID_KEY_ID as described in Section 5.2.1. Lines 7 and 8 are part of the Traffic Selector and specify the use of UDP as Transport Layer protocol and restrict the SA to traffic coming from and going to port 12345. In line 11 we instruct IKE to accept the encryption and integrity algorithms implemented in the IKEv2 for RIOT implementation. Since we do not specifically request the use of 'transport mode', the SA defaults to the use of 'tunnel mode'

Listing 5.5: IPsec configuration for Libreswan as part of the proof of concept implementation.

```
1  conn test
2          left=cafe::102
3          right=cafe::2
4          authby=secret
5          auto=add
6          fragmentation=no
7          leftid=@[abcdefgh]
8          leftprotoport=udp/12345
9          rightprotoport=udp/12345
10
11         ike=aes-sha1
```

We provide the same configuration to the Device. Since the IKE module introduced in Section 5.2 lacks an external means of configuration at the time of writing, this requires minor changes to some of the header files. Note that in practice, the configuration on the side of the Gateway could be specified to handle connections from multiple Devices. However, as long as the configuration for the Device is identical to the one presented, the resulting SA would be functionally the same since both endpoints have to agree on the parameters during SA negotiation.

Listing 5.6: Dbfrm command to transfer SA information from the IKE module to the ESP module as used in the proof of concept.

```

dbfrm protect '1' '3186194719' 'cafe::2' 'cafe::102'
        17 12345 12345 tunnel authenc sha 'INTEG_KEY'
        aes_cbc 'ENCR_KEY' '4242424242' cafe::2 cafe::102
```

In the second step, IKE is tasked with negotiating the SA. This process is described in some detail in Section 5.2.1 and Section 2.4.3. During the IKE Initial Exchanges, we also establish the use of SCHC compression in ESP protected traffic for the SA by using an IKE Notify payload as described in Section 4.7.2. Additionally, the mode of operation is set to preset-mode with the parameters proposed in Section 4.7.1. Assuming the SA negotiation was successful we have established an IPsec SA in self-encapsulated tunnel mode between Device and Gateway with the parameters specified above. To transfer the parameters negotiated during the IKE exchanges to the ESP module we use the 'dbfrm' tool. The command used to transfer the specific parameters used in this example is shown in Listing 5.6. For details on how to interpret the parameters please refer to Section 5.3.1. The values 'INTEG_KEY' and 'ENCR_KEY' are placeholders for the actual keys that are provided by the IKE module after the initial exchanges. In the next step, the process of generating SCHC Rules for the SA commences.

Rule Generation

The Rule generation process follows the steps specified in Section 4.7.6. From the SA negotiated in this example, we can derive values for the Context Parameters as shown in Table 5.1. All remaining Context Parameters are provided by the preset-values proposed in Section 4.7.1.

Table 5.1.: Context Parameters derived from the sample SA.

Context Parameter	Value
esp_spi	3186194719
esp_sn	0
l4_proto	UDP(17)
ipsec_mode	"Tunnel"
ip_version	6
ip6_dev_prefix	cafe::0
ip6_dev_idd	0::102
ip6_app_prefix	cafe::0
ip6_app_iid	0::2
ip6_tun_prefix	cafe::0
ip6_tun_iid	0::2
udp_dev_port	12345
udp_app_port	12345

Subsequently, we generate a SCHC Rule using the Context Parameters determined in the previous step. Since we can identify that the SA specifies a self-encapsulated tunnel by comparing the 'ip6_app' and 'ip6_tun' Context Parameters, a single Rule is sufficient (cf. Section 4.7.5). The finalized Rule is shown in Appendix A, Table A.1.

In the table, FDs for the different headers are shown in the order of appearance in the overall encapsulated packet. This is not required for the implementation but has been done to improve readability. In the implementation the FDs would be split into separate Rules corresponding to their processing phase as described in Section 5.5.2. Furthermore, not all FDs for the inner IPv6 are shown in Table A.1. All FDs that are not displayed are identical to their counterpart in the outer IPv6 header.

Compression/Decompression

In the next step, we attempt to send a UDP packet from port 12345 of the Device to port 12345 on the Gateway. As payload we use a string of characters ("PAYLOAD"). Note that this is not a Null-terminated string but simply the concatenation of characters. Since an ESP SA is specified to protect traffic transmitted between these specific ports from the Device to the Gateway, IPsec starts processing the packet. After ESP encapsulation, SCHC plaintext phase processing identifies the Rule associated with the SPI of the ESP SA and parses the headers. The subsequent processing is described in detail in Section 5.5.2. The Rule used for processing can be derived from the Rule shown in Table A.1 by splitting the Rule at the first FD of the inner IPv6 header (IPv6 Version). For plaintext phase processing the second part of the Rule is used. In the ciphertext processing phase the first part is used.

Table 5.2.: Compression Residues for a test transmission in the sample SA.

Compression Residue	Length	Value
L2 Header	8	1
RuleID	8	1
ESP SPI	4	15
ESP Sequence Number	4	1
Payload	56	"PAYLOAD"
ESP Padding	64	-
ESP Pad Length	8	64
ESP ICV	32	-
Total	176	-

Table 5.2 shows the Compression Residues in the compressed packet. 'L2 Header' represents the entire L2 header. The prototype uses a custom L2 protocol which only carries the ID of the device as an 8-bit value. 'RuleID' represents the SCHC RuleID that needs to be transmitted for decompression. Since we only have one Rule, the Rule has been assigned the ID of '1'. In the prototype, we have decided to use a 8-bit value for the RuleID. In practice, a value of any length could be used as long as all Rules can be unambiguously identified by both communication partners. The process of selecting a viable length for the RuleID field is out of the scope of this work. Note that even though we actually use two Rules, since we split the Rule between the processing phases, the same RuleID can be used for both. 'ESP SPI' represents the Compression Residue of the ESP SPI field. Its value consists of the four least significant bits of the actual SPI value. 'ESP Sequence Number' represents the Compression Residue of the ESP Sequence Number field. Its value consists of the four least significant bits of the actual Sequence Number value. The Sequence Number field requires some special treatment during compression and decompression. This is due to the volatile nature of the field, as it is the only one that changes between transmissions. For this reason, the Target Value for the ESP Sequence Number FD has to be updated every time ciphertext phase processing is performed. As stated in Section 4.6.2 we expected the sequence numbers to be generated by incrementation. Therefore, in each ciphertext processing phase on each of

the endpoints, the Target Value of the ESP Sequence Number has to be incremented by one. [Ken05] specifies, that the Sequence Number is initialized with a value of '0', but the first packet sent using the SA contains a Sequence Number value of '1'. Using the Rule shown in Table A.1 and incrementing the Sequence Number before applying compression yields the Comrpession Residue for the Sequence Number shown in Table 5.2. 'Payload' represents the original Layer 4 payload which has not been compressed. 'ESP Padding' represents the padding added by ESP to perform encryption. Encryption is applied to the entire encapsulated payload and the ESP trailer fields consisting of the ESP Pad Length and the ESP Next Header field. The Compression Residue for the encapsulated payload is entirely comprised of the Layer 4 payload because all header fields have been eliminated by the compression. The encryption algorithm used in the prototype is AES-128 in CBC mode. From the AES blocksize (128 bits), the size of the compressed Payload Data field (56 bits = 7 bytes) and the size of the compressed ESP trailer fields (8 bits) the size of the ESP Padding field is calculated. After compression, the packet including payload has a total size of 176 bits (22 bytes). For decompression, the entire process is reversed on the Gateway side. Since at the time of writing SCHC integration on the Gateway could not be completed, the Compression Residue was validated by hand. For this purpose, the traffic was captured and decrypted using the key provided by the IKE module.

6. Evaluation

In this chapter, the concept introduced in Chapter 4 is evaluated. To measure the performance of the compression based on the context information derived from IPsec, we define best and worst case scenarios for the different modes of operation and evaluate the results against the frame size limits of LoRa. Based on our observations during the development of the proof of concept, we also propose improvements and alternatives to some of the components and recognize the limitations of the concept.

6.1. Compression Efficiency

To judge the benefits of the SCHC Rules introduced in this work, we first examine the compression efficiency in different scenarios. The metric used in this section represents the fraction of a header that could be eliminated through compression and is named space savings (SV). The metric is calculated from the size of the compressed header (S_c) and the original size of the uncompressed header (S_o) by using the following formula:$SV = \frac{S_o - S_c}{S_o}$.

Additionally, we divide the scenarios into two groups: 'best case' and 'worst case'. In the best case scenarios, all Context Parameters obtainable from the SA are assumed to be specified. For this to be the case an SA has to be configured in the following way:

- 'left' is set to a single IPv6 address.
- 'right' is set to a single IPv6 address.
- 'leftprotoport' is set to a single protocol and port number.
- 'rightprotoport' is set to a single protocol and port number.
- 'leftsubnet/rightsubnet' are not set.

An SA configured this way is only valid for a connection from a single device using the service specified by the combination of L4 protocol and port number.

In the worst case scenarios many of the Context Parameters obtainable from the SA are assumed to be unspecified. Table 6.1 shows all Context Parameters in question as well as their assumed status. Even though we consider this the worst case, some Context Parameters can still be obtained from the SA. These parameters are basic requirements for IPsec operation and include the SPI, sequence number and mode of operation as well as the selected Layer 3 protocol (IPv6). Additionally, IPsec always requires some amount of knowledge about the communication partners. For this reason, we assume that at least the 64-bit network prefix for all communication partners is known to the SA. All FDs that contain unspecified Context Parameters are subject to Rule Downgrading as specified in Section 4.6.3 and have their CDA adjusted to 'value-sent'.

In the following, we examine best and worst case scenarios for the different modes of operation introduced in Section 4.7.1. In the first two sections IPsec is assumed to operate

Table 6.1.: Status of the individual Context Parameters in the worst case scenarios.

Context Parameter	Status	Rule Downgrade
esp_spi_lsb	specified	-
esp_sn_lsb	specified	-
l4_proto	unspecified	value-sent
ipsec_mode	specified	-
ip_version	specified	-
ip6_dev_prefix	specified	-
ip6_dev_iid	unspecified	value-sent
ip6_app_prefix	specified	-
ip6_app_iid	unspecified	value-sent
ip6_tun_prefix	unspecified	value-sent
ip6_tun_iid	unspecified	value-sent
udp_dev_port	unspecified	value-sent
udp_app_port	unspecified	value-sent

in transport mode. Compression efficiency for scenarios in tunnel mode are discussed in Section 6.1.3. For reference, the uncompressed header sizes for all protocols are shown in Table 6.2. Note that the size of the ESP header is calculated without any payload or padding. Sync-mode is not represented in the evaluation since the Context Parameters and in turn the size of the Compression Residue depend largely on the actual values of said parameters which are negotiated during the IKE Initial Exchanges. It is also noteworthy that the values for Compression Residues and space savings are calculated without any Layer 4 payload or padding present. Due to the interaction between the ESP Payload Data field and the required padding, the actual packet size can be larger than indicated. As proposed in Section 4.6.4 the use of Counter Mode for encryption algorithms (e.g. AES-CTR) could help eliminate any overhead introduced by padding.

Table 6.2.: Uncompressed header size for IPv6, ESP and UDP

Header	Uncompressed Size (in bits)
IPv6	320
ESP	112
UDP	64
Total	496

6.1.1. Strict-mode

In strict-mode, only context information obtained from the IPsec SA is used to perform compression. Consequently, all Context Parameters designated as 'Negotiated' are considered as 'unspecified' and the corresponding FDs are subject to Rule Downgrading. This affects the following header fields, corresponding Context Parameters are shown in parenthesis:

- IPv6 Traffic Class (ip_tc)
- IPv6 Flow Label (ip_fl)
- IPv6 Hop Limit (ip_hl)
- ESP SPI (esp_spi_lsb)
- ESP Sequence Number (esp_sn_lsb)

Appendix B.1 shows the resulting Rule Templates in transport mode. The Compression Residues for each Rule can be calculated by adding the Field Length values for all FDs that indicate the 'value-sent' CDA and adding the LSB parameters for all FDs that indicate the LSB CDA (if any).

Table 6.3.: Compression Residues for the best case scenario in strict-mode.

Header	Compression Residue (in bits)	Space Savings (in %)
IPv6	36	88.75
ESP	104	7.14
UDP	0	100
Total	140	71.77

Table 6.4.: Compression Residues for the worst case scenario in strict-mode.

Header	Compression Residue (in bits)	Space Savings (in %)
IPv6	164	48.75
ESP	112	0
UDP	32	50
Total	308	37.90

Table 6.3 and Table 6.4 show the resulting Compression Residues for the best and worst case scenario respectively. In the best case scenario, the total Compression Residue could be reduced to below 18 bytes (140 bits). Here, the ESP SPI and ESP Sequence Number fields have the biggest impact on the size of the compressed headers. In the worst case scenario, the total Compression Residue is almost 39 bytes (308 bits) in size. In addition to the aforementioned ESP header fields, half of the address fields in the IPv6 header are not compressed.

6.1.2. Preset-mode

In preset-mode Context Parameters which can not be obtained from the SA are drawn from a set of default values proposed in Section 4.7.1. The resulting Rule Templates for transport mode are shown in Appendix B.2.

Table 6.5.: Compression Residues for the best case scenario in preset-mode.

Header	Compression Residue (in bits)	Space Savings (in %)
IPv6	0	100
ESP	48	57.14
UDP	0	100
Total	48	90.32

Table 6.6.: Compression Residues for the worst case scenario in preset-mode.

Header	Compression Residue (in bits)	Space Savings (in %)
IPv6	128	60
ESP	56	50
UDP	32	50
Total	216	56.45

The Compression Residues for each Rule can be calculated as described above. Table 6.5 and Table 6.6 show the resulting Compression Residues for the best and worst case scenario respectively. In the best case scenario, both IPV6 and UDP header can be completely eliminated by the compression. The largest part of the Compression Residue (more than 60%) is occupied by the ESP ICV field. In the worst case scenario, the uncompressed IPv6 Device Identifiers have once again the largest impact on compression efficiency. As expected, the resulting Compression Residues for preset-mode operation are overall smaller than those generated in strict-mode.

6.1.3. Tunnel Mode

For tunnel mode traffic we have to differentiate between self-encapsulated and VPN traffic. In self-encapsulated traffic, the inner IPv6 header is practically identical to the outer IPv6 header (cf. Section 4.7.5). As a result, the additional Compression Residue has exactly the same size as the Compression Residue of the outer IPv6 header in the scenarios above. In the best case preset-mode scenario the outer IPv6 header is also eliminated completely. Table 6.7 shows the total Compression Residues for the different scenarios in self-encapsulated tunnel mode. For reference, the first row of the table shows the new total size of the uncompressed headers. Because the IPv6 header can be compressed with relatively high efficiency in all scenarios we notice overall increased space savings. However, the total size of the packets increases in all cases except for the preset-mode best case, where both IPv6 headers can be eliminated entirely.

Table 6.7.: Total Compression Residues for different scenarios in self-encapsulated tunnel mode.

Mode	Total Compression Residue (in bits)	Space Savings (in %)
Uncompressed	816	-
strict-mode best	176	78.43
strict-mode worst	472	42.15
preset-mode best	48	94.12
preset-mode worst	344	57.84

For VPN traffic, the inner IPv6 header has the actual recipient of the packet as destination while the outer IPv6 header is addressed to the encryption endpoint. In this case, the additional size of the total Compression Residue depends mainly on the granularity of the SA and could be up to 128 bits larger, which corresponds to the size of the address of the recipient.

6.1.4. Summary

One of the main reasons to use header compression is the limited size for payloads in LPWA technologies. For LoRa, the maximum payload size ranges from 59 to 250 bytes on the lowest layer [PMP$^+$17] and depends on the spreading factor (SF). The spreading factor also influences transmission range, time on air and the maximum achievable bitrate ([Sem19]). When performing header compression our main goal is to be able to transmit as much actual payload data as possible and avoid fragmentation of the packet. This is especially important due to the existing duty cycle restrictions which limit the time on air for LPWAN devices.

Examining the total Compression Residues in the scenarios introduced above, we can observe, that all Compression Residues with the exception of only a few (worst case tunnel mode) reduce the total header sizes from more than 62 bytes to below 59 bytes which corresponds to the lowest available payload size for LoRa. In five of the eight scenarios the resulting header sizes are below the 59 byte threshold by more than 30 bytes and therefore leave a considerable amount of room for actual payload data.

6.1.5. IKE Overhead

Since use of the compression introduced in this work requires the existence of an IPsec SA we have to analyse the overhead introduced by the creation of said SA. The protocol responsible for SA creation is IKEv2 introduced in Section 2.4.3. From limited testing we have found that IKE exchanges in our testing environment (cf. Section 5.1) vary from 200 to 600 bytes per message in size. One of the largest components of the IKE_SA_INIT would seem to be the SA payload which can, in normal operation, hold multiple proposals for encryption and integrity algorithms. Although the scientific estimation of the actual sizes and composition of IKE exchanges in different application scenarios is not within the scope of this work, we can still assume that a single message in an IKE exchange is most likely too large for some LPWA technologies.

For this reason, we introduced two techniques to apply header compression to the IKE exchanges (cf. Section 4.8). In the first technique we proposed the use of SCHC to compress the IKE headers in a similar way as previously proposed in combination with the IPv6, ESP and UDP headers. However we could only identify enough static context for compression for the outermost IKE Header. We can estimate the compression to eliminate at most 19 bytes from the header fields in the IKE Header. For the subsequent headers, we do not expect to save a significantly larger amount of space. This translates to a space savings of less than 10% depending on the size of the IKE message. For this reason, we cannot recommend the use of SCHC for compression of the IKE headers as proposed in Section 4.8.1. This is not entirely unexpected because SCHC relies on static context information which is known to both communication partners before the transmission. Since IKE is specifically designed to exchange and negotiate context information which is considered its payload, it does not represent a particularly likely candidate for SCHC compression.

As an alternative, Section 4.8.2 introduces a different approach to IKE compression as proposed by V. Smyslov [Smy20]. The basic idea is to employ the loss-less compression algorithms, deflate, LZS and LZJH, used in IP Payload Compression (IPComp) [SMPT01] to reduce the size of IKE messages. [FM98] estimates the compression ratio for LZS to be within 1.43 and 1.58 for datagrams with a size of 256 to 512 bytes, which corresponds to a space savings of 30 to 37 percent. This is already an improvement over the approach introduced above.

6.2. End-to-end Setup

Section 4.7.4 introduces two methods to handle end-to-end encryption while maintaining SCHC compression derived from the IPsec SA. In the first approach, compression of the outer header(s) is omitted to allow forwarding of the packet. This reduces compression efficiency and increases the total header size of the packet. In Section 6.2.1 we take a look at compression efficiency when using this method. In the second approach, the intermediate device(s) are informed about the compression context. During implementation it has become increasingly obvious that this method requires a considerable development and deployment overhead. This is especially true if the Gateway is not directly connected to the Server and routing needs to be performed. For this reason we cannot recommend the second approach. Section 6.2.2 introduces a possible alternative which also does not sacrifice compression efficiency.

6.2.1. Compression Efficiency

Table 6.8 shows the resulting compression efficiency if the outer IPv6 header is transmitted without compression. For reference, the first two rows indicate the total uncompressed header size for transport and tunnel mode. The four scenarios shown below are the best and worst case preset-mode scenarios with and without the use of tunnel mode. For transport mode especially, we can observe a significant decrease in compression efficiency and the total size of the Compression Residues increases to between 46 and 67 bytes. This limits the usefulness of the end-to-end setup significantly.

Table 6.8.: Compression efficiency when transmitting the outer IPv6 header without compression.

Mode	Total Compression Residue (in bits)	Space Savings (in %)
Transport Uncompressed	496	-
Tunnel Uncompressed	816	-
preset-mode best	368	28.80
preset-mode worst	408	17.74
tunnel mode best	368	54.90
tunnel mode worst	536	34.31

6.2.2. Alternatives

If the goal is to provide end-to-end encryption between the Device and a single known endpoint (e.g. server) we propose a different approach. To eliminate the need to decompress the outer IPv6 header on any of the intermediate devices, the Gateway has to establish an IP tunnel to the endpoint. Additionally, the Gateway must not process the compressed IPv6 header, but instead identify the Device by its Layer 2 address and forward the packet to the appropriate endpoint through the tunnel. This requires the Gateway to maintain a list of device addresses and their corresponding endpoints. For incoming traffic, the Gateway has to identify the corresponding layer 2 address of the device specified in the IPv6 tunnel. Since in IPv6 traffic the interface identifier is most commonly derived from the layer 2 address this could be solved by simply forwarding the packet to the device specified by the address of the tunnelled packet. The IP tunnel is used to encapsulate any ESP traffic in transport or tunnel mode as well as any IKE messages. This way the IKE SA and the corresponding compression context is established directly between Device and endpoint without requiring any of the intermediate devices to be informed about it.

6.3. Design Goals

In this section, we re-examine the design goals set in Section 4.1. For design goal (I.a) we have shown that in strict-mode, SCHC compression is possible using only context information gathered from the IPsec SA. This way no configuration of SCHC is needed and compression efficiency depends largely on the granularity of the SA. By dividing SCHC processing into the ciphertext and plaintext phases, we allow compression of the encrypted headers in the ESP payload as required in design goal (I.b). To allow compatibility with standard IPsec operation we require IKE to notify the communication partner of the mode of compression as described in Section 4.7.2. This involves some changes to the IKE implementation in the form of additional notify payload types. Additionally SCHC plaintext phase processing has to be integrated into ESP. Other than that, no substantial changes have been made to SCHC or IPsec as required by design goal (I.c). With the introduction of preset-mode and sync-mode, we have shown two options to increase compression efficiency by exploiting some basic assumptions about the traffic flow and extending the interaction with the IKE protocol (II.a). To address design goal (II.b) we have introduced two methods to reduce the overhead produced by the IKE protocol. Furthermore, SCHC compression is not only applied to the encrypted payload, but also to the ESP header itself.

6.4. Layer 2 Protocol Choices

As indicated in Chapter 5 our prototype uses a custom layer 2 protocol. The main reason for this is that at the time writing, most existing layer 2 protocols used in LPWANs do not yet support IPV6 due to their limited payload size.

The most popular layer 2 protocol for LoRa is LoRaWAN but there are several alternatives like Symphony Link and MAC on Time (MoT) [QGZ+19]. At the time of writing, Gimenez et al. have recently proposed the integration of SCHC into LoRaWAN [GP19]. The draft is part of the 'IPv6 over LPWAN' working group which aims to integrate the IPv6 protocol stack into LPWA technologies. The most popular alternative to LoRa is Sigfox. Similarly, integration of SCHC into Sigfox has been recently proposed by Zuniga et al. [ZGT20]. Any of the protocols mentioned above could be viable candidates for layer 2 protocols, however selection criteria and additional integration are out of the scope of this work.

6.5. Caveats and Limitations

In this section, we examine some caveats and limitations that were discovered during the course of implementation and the development of the concept.

- Although no SCHC configuration is required to enable the compression proposed in this work, some of the configuration cost can be considered to have moved from SCHC to IPsec. This is especially true if we acknowledge the correlation between the granularity of the SA and the quality of the compression. In short, a more specific SA configuration leads to better compression since more of the Context Parameters required for compression are specified in the SA.

- Considering that constrained devices operating in LPWANs are not only restricted by their network bandwidth but also in memory and computational power, some devices may not want to run any IPsec implementation due to these restrictions. Unfortunately, there is no obvious solution in the context of this work. However, we believe that future adaptations of the IP protocol stack for IoT use may include improvements that facilitate integration of IPsec on constrained devices.

- In some LPWAN scenarios the network return path may be restricted. In this case, the creation of an IPsec SA is severely impeded and may require the use of manual keying. This provides additional challenges for the concept introduced in this work and reintroduces a significant configuration cost.

- Due to the interaction between plaintext phase processing and ESP we suspect that retrofitting IPsec with SCHC compression is rather expensive. For this reason, we believe that, in an effort to keep configuration and setup cost down, SCHC compression needs to come pre-installed with the IPsec implementation.

- Although the Integrity Check Value (ICV) in ESP is designed to check the integrity of packets after transmission, the value can not be used to verify successful decompression. This is due to the fact that compression of the inner headers has to be performed before encryption and the ICV is calculated on the encrypted packet.

7. Conclusion

Driven by optimized radio modulation, low power requirements and long transmission ranges, LPWA technologies are gaining popularity in industrial and research communities. In recent development, numerous strives have been made to alleviate one of the major shortcomings of these new technologies, their lack of interoperability. Layer 2 protocols used in LPWA technologies are highly optimized and specific for each individual technology. For this reason, securing and expanding existing LPWA networks provides significant challenges. As a response, researchers have started introducing methods to provide IPv6 connectivity to LPWAN devices. Because of their limited network bandwidth, integration of the IPv6 protocol stack requires some means to adapt and compress the existing headers to fit into smaller transmissions. While similar in idea to 6LowPAN, the development for IPv6 over LPWAN has taken particular interest in the Static Context Header Compression (SCHC) algorithm. For operation, SCHC requires the presence of static compression context on all devices. This static context is usually provided through manual configuration and SCHC provides no mechanisms to collect or distribute compression context between multiple devices. In this work, we introduce an extension to SCHC that is able to leverage context information used and gathered by IPsec protocols to apply header compression without the need for manual configuration.

This work offers a method to derive compression context directly from IPsec Security Associations by first gathering parameters from the IPsec databases and then generating SCHC Rules specifically designed to fit the traffic flow in the SA. To guarantee that the resulting compression rule matches the expected flow, we have introduced the concept of SCHC Rule Templates as a basis for further adjustments of the rule. Each template can be adapted to match the granularity of the IPsec SA and ensures that missing context information leads to transmission of the data instead of the dismissal of the rule. In addition, this work provides several modes of operation that can operate on different assumptions about the traffic flow to improve the efficiency of the compression.

The proof of concept demonstrates the major development obstacles that have to be faced when integrating SCHC compression into IPsec protected traffic. The prototype is implemented based on a two phase design developed for this work, which allows compression of the encapsulated headers before encryption in addition to any outer unencrypted headers. Additionally, the prototype has been developed for the RIOT OS operating system, which offers support for a multitude of constrained devices as well as leading LPWA technologies.

The evaluation shows that the concept provided by this work is more than sufficient when it comes to reducing the overhead introduced by a common IPv6/UDP protocol stack for use in an LPWA environment. For all but one of the tested scenarios, we were able to fit the resulting protocol stack into any LoRa transmission even on the smallest payload settings. In a best case worst case analysis, the concept was able to reduce the total compression residue for ESP encapsulated traffic by 37% to 94%.

Future Work

Based on the findings in this work, we recognize that additional research has to be conducted to reduce the traffic overhead generated by the IKEv2 protocol. Since the operation of the protocol is a basic requirement for the concept introduced in this work and bypassing the protocol by performing manual keying is not advised, a significant reduction in transmission size for the IKE Initial Exchanges is almost essential. Furthermore, the general impact of IPsec on memory usage and computational requirements in the context of constrained devices has to be explored. Some impact may be made by introducing additions like the Layered SCHC (LSCHC) approach by Abdelfadeel et al. [ACP17] to the SCHC implementation. In addition, before considering deployment of the concept introduced in this work, suitable layer 2 protocols for integration of the IP stack have to be identified.

Appendices

A. Sample Rule

Table A.1.: Finalized SCHC Rule based on the Context Parameters derived from the sample SA.

FID	FL	FP	DI	Target Value	MO	CDA
IPv6 Version	4	1	Bi	6	equal	not-sent
IPv6 Traffic Class	8	1	Bi	0	ignore	not-sent
IPv6 Flow Label	20	1	Bi	0	ignore	not-sent
IPv6 Payload Length	16	1	Bi	-	ignore	compute-*
IPv6 Next Header	8	1	Bi	ESP(50)	equal	not-sent
IPv6 Hop Limit	8	1	Bi	255	ignore	not-sent
IPv6 DevPrefix	64	1	Bi	cafe::0	equal	not-sent
IPv6 DevIID	64	1	Bi	0::102	equal	not-sent
IPv6 AppPrefix	64	1	Bi	cafe::0	equal	not-sent
IPv6 AppIID	64	1	Bi	0::2	equal	not-sent
ESP SPI	32	1	Bi	3186194719	MSB(28)	LSB(4)
ESP Sequence Number	32	1	Bi	0	MSB(28)	LSB(4)
ESP Payload Data	variable	1	Bi	-	ignore	value-sent
ESP Padding	variable	1	Bi	-	ignore	value-sent
IPv6 Version	4	1	Bi	6	equal	not-sent
...	.	.	.	.	...	...
IPv6 Next Header	8	1	Bi	UDP(17)	equal	not-sent
...	.	.	.	.	...	...
UDP Dev Port	16	1	Bi	12345	equal	not-sent
UDP App Port	16	1	Bi	12345	equal	not-sent
UDP Length	16	1	Bi	-	ignore	compute-*
UDP Checksum	16	1	Bi	-	ignore	compute-*
ESP Pad Length	8	1	Bi	-	ignore	value-sent
ESP Next Header	8	1	Bi	IPv6 encap(41)	equal	not-sent
ESP ICV	32	1	Bi	-	ignore	value-sent

B. Rule Templates

B.1. Strict-mode

Table B.1.: IPv6 Rule Template

FID	FL	FP	DI	Target Value	MO	CDA
IPv6 Version	4	1	Bi	ip_version	equal	not-sent
IPv6 Traffic Class	8	1	Bi	-	ignore	value-sent
IPv6 Flow Label	20	1	Bi	-	ignore	value-sent
IPv6 Payload Length	16	1	Bi	-	ignore	compute-*
IPv6 Next Header	8	1	Bi	l4_proto, ipsec_mode	equal	not-sent
IPv6 Hop Limit	8	1	Bi	-	ignore	value-sent
IPv6 DevPrefix	64	1	Bi	ip6_dev_prefix	equal	not-sent
IPv6 DevIID	64	1	Bi	ip6_dev_iid	equal	not-sent
IPv6 AppPrefix	64	1	Bi	ip6_app_prefix	equal	not-sent
IPv6 AppIID	64	1	Bi	ip6_app_iid	equal	not-sent

Table B.2.: ESP Rule Template

FID	FL	FP	Di	Target Value	MO	CDA
ESP SPI	32	1	Bi	-	ignore	value-sent
ESP Sequence Number	32	1	Bi	-	ignore	value-sent
ESP Payload Data	variable	1	Bi	-	ignore	value-sent
ESP Padding	variable	1	Bi	-	ignore	value-sent
ESP Pad Length	8	1	Bi	-	ignore	value-sent
ESP Next Header	8	1	Bi	l4_proto, ipsec_mode	equal	not-sent
ESP ICV	32	1	Bi	-	ignore	value-sent

Table B.3.: UDP Rule Template

FID	FL	FP	DI	Target Value	MO	CDA
UDP Dev Port	16	1	Bi	udp_dev_port	equal	not-sent
UDP App Port	16	1	Bi	udp_app_port	equal	not-sent
UDP Length	16	1	Bi	-	ignore	compute-*
UDP Checksum	16	1	Bi	-	ignore	compute-*

B.2. Preset-mode

Table B.4.: IPv6 Rule Template

FID	FL	FP	DI	Target Value	MO	CDA
IPv6 Version	4	1	Bi	ip_version	equal	not-sent
IPv6 Traffic Class	8	1	Bi	ip_tc	ignore	not-sent
IPv6 Flow Label	20	1	Bi	ip_fl	ignore	not-sent
IPv6 Payload Length	16	1	Bi	-	ignore	compute-*
IPv6 Next Header	8	1	Bi	l4_proto, ipsec_mode	equal	not-sent
IPv6 Hop Limit	8	1	Bi	ip_hl	ignore	not-sent
IPv6 DevPrefix	64	1	Bi	ip6_dev_prefix	equal	not-sent
IPv6 DevIID	64	1	Bi	ip6_dev_iid	equal	not-sent
IPv6 AppPrefix	64	1	Bi	ip6_app_prefix	equal	not-sent
IPv6 AppIID	64	1	Bi	ip6_app_iid	equal	not-sent

Table B.5.: ESP Rule Template

FID	FL	FP	Di	Target Value	MO	CDA
ESP SPI	32	1	Bi	esp_spi_lsb, esp_spi	MSB	LSB
ESP Sequence Number	32	1	Bi	esp_sn_lsb, esp_sn	MSB	LSB
ESP Payload Data	variable	1	Bi	-	ignore	value-sent
ESP Padding	variable	1	Bi	-	ignore	value-sent
ESP Pad Length	8	1	Bi	-	ignore	value-sent
ESP Next Header	8	1	Bi	l4_proto, ipsec_mode	equal	not-sent
ESP ICV	32	1	Bi	-	ignore	value-sent

Table B.6.: UDP Rule Template

FID	FL	FP	DI	Target Value	MO	CDA
UDP Dev Port	16	1	Bi	udp_dev_port	equal	not-sent
UDP App Port	16	1	Bi	udp_app_port	equal	not-sent
UDP Length	16	1	Bi	-	ignore	compute-*
UDP Checksum	16	1	Bi	-	ignore	compute-*

List of Figures

2.1. Typical LPWA-Network arranged in a star topology. Multiple IoT devices(left) connect through a gateway(centre) to a central server(right). 4
2.2. IPv6 Header as specified in RFC 8200 [DH17] 6
2.3. ESP header format as specified in [Ken05] 8
2.4. IKE_SA_INIT exchange without optional payloads [KHN+14] 9

3.1. SCHC adaptation layer: Compression and Fragmentation sublayers are located between an upper layer (eg. IPv6) and a lower layer (eg. a LPWAN technology) [MTG+20] 12

4.1. Simplified network topology showing a single Device connected to a Server via a Gateway. The Device is connected to the Gateway via a LPWA technology. The Gateway connects to the Server over the Internet. 20
4.2. Protocol stack with integrated IPsec implementation. 21
4.3. ESP header encapsulation in IPv6 [Ken05] 22
4.4. Topology for end-to-end encryption: Device and Server share a common compression context established through IKE. When attempting to transmit a compressed packet, the Gateway is unable to decompress the outer IPv6 header and forward the packet. 32
4.5. Overview of the Rule creation process. 33
4.6. IKE Compressed payload header as proposed in [Smy20] 37
4.7. Compressed IKE_SA_INIT exchange without optional payloads as proposed in [Smy20] 37

5.1. Sequence diagram for the IKE Initial Exchanges. 40
5.2. Generating authentication data from a Pre-shared key in IKEv2 [KHN+14] . 41
5.3. Generating keying material for IPsec as introduced in [KHN+14] 42
5.4. SCHC Compression Sequence 48
5.5. SCHC Decompression Sequence 49
5.6. SCHC integration into the IPsec stack: SCHC processing is integrated in two phases to allow compression of encrypted traffic. 51

List of Tables

3.1. List of SCHC Matching Operators(MOs) [MTG$^+$20] 13
3.2. List of SCHC Compression/Decompression Actions(CDAs) [MTG$^+$20] 14
3.3. List of ECH Context Parameters for the inner IPv6 header [MGBS19] 15
3.4. EHC Rules for the ESP header [MGBS19] 16
3.5. Diet-ESP Context Parameters for ESP [MGBS19] 17
3.6. Diet-ESP Rules for the ESP header [MGBS19] 17

4.1. Context Parameters for the ESP header. 24
4.2. Context Parameters for the IPv6 header. 25
4.3. Context Parameters for the UDP header. 25
4.4. Rule Template for the ESP header. 27
4.5. Rule Template for the IPv6 header. 27
4.6. Rule Template for the UDP header. 28
4.7. Default values for the Context Parameters in preset-mode. 30
4.8. Candidates for SCHC compression in the IKE header fields during *IKE_SA_INIT*. 35
4.9. Context Parameters for the IKE Header 36
4.10. Rule Template for the IKE Header. 36

5.1. Context Parameters derived from the sample SA. 54
5.2. Compression Residues for a test transmission in the sample SA. 55

6.1. Status of the individual Context Parameters in the worst case scenarios. . . . 58
6.2. Uncompressed header size for IPv6, ESP and UDP 58
6.3. Compression Residues for the best case scenario in strict-mode. 59
6.4. Compression Residues for the worst case scenario in strict-mode. 59
6.5. Compression Residues for the best case scenario in preset-mode. 60
6.6. Compression Residues for the worst case scenario in preset-mode. 60
6.7. Total Compression Residues for different scenarios in self-encapsulated tunnel mode. 61
6.8. Compression efficiency when transmitting the outer IPv6 header without compression. 63

A.1. Finalized SCHC Rule based on the Context Parameters derived from the sample SA. 71

B.1. IPv6 Rule Template 73
B.2. ESP Rule Template 73
B.3. UDP Rule Template 74
B.4. IPv6 Rule Template 74
B.5. ESP Rule Template 74

B.6. UDP Rule Template 75

Listings

5.1. Transform configuration for the ESP proposal in ike_config.h 42
5.2. Generating keying material for the IKE SA and ESP SA in ikev2.c 43
5.3. Integration of ESP processing into the GNRC network stack in gnrc_ipv6_ext.c 44
5.4. Declaration of the ESP processing methods for ingoing and outgoing traffic in esp.h 46
5.5. IPsec configuration for Libreswan as part of the proof of concept implementation. 53
5.6. Dbfrm command to transfer SA information from the IKE module to the ESP module as used in the proof of concept. 53

Bibliography

[ACJR11] Amante, S. ; Carpenter, B. ; Jiang, S. ; Rajahalme, J.: IPv6 Flow Label Specification / RFC Editor. RFC Editor, November 2011 (6437). – RFC. – ISSN 2070–1721

[ACP17] Abdelfadeel, Khaled Q. ; Cionca, Victor ; Pesch, Dirk: Lschc: Layered static context header compression for lpwans. In: *Proceedings of the 12th Workshop on Challenged Networks*, 2017, S. 13–18

[ACP18] Abdelfadeel, Khaled Q. ; Cionca, Victor ; Pesch, Dirk: Dynamic context for static context header compression in LPWANs. In: *2018 14th International Conference on Distributed Computing in Sensor Systems (DCOSS)* IEEE, 2018, S. 35–42

[AIM10] Atzori, Luigi ; Iera, Antonio ; Morabito, Giacomo: The internet of things: A survey. In: *Computer networks* 54 (2010), Nr. 15, S. 2787–2805

[BEK14] Bormann, C. ; Ersue, M. ; Keranen, A.: Terminology for Constrained-Node Networks / RFC Editor. Version: May 2014. `http://www.rfc-editor.org/rfc/rfc7228.txt`. RFC Editor, May 2014 (7228). – RFC. – ISSN 2070–1721. – `http://www.rfc-editor.org/rfc/rfc7228.txt`

[BGH$^+$18] Baccelli, E. ; Gündogan, C. ; Hahm, O. ; Kietzmann, P. ; Lenders, M. S. ; Petersen, H. ; Schleiser, K. ; Schmidt, T. C. ; Wählisch, M.: RIOT: An Open Source Operating System for Low-End Embedded Devices in the IoT. In: *IEEE Internet of Things Journal* 5 (2018), Nr. 6, S. 4428–4440

[BHG$^+$13] Baccelli, E. ; Hahm, O. ; Günes, M. ; Wählisch, M. ; Schmidt, T. C.: RIOT OS: Towards an OS for the Internet of Things. In: *2013 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, 2013, S. 79–80

[CVZZ16] Centenaro, M. ; Vangelista, L. ; Zanella, A. ; Zorzi, M.: Long-Range Communications in Unlicensed Bands: the Rising Stars in the IoT and Smart City Scenarios. In: *IEEE Wireless Communications* 23 (2016), October

[DH17] Deering, S. ; Hinden, R.: Internet Protocol, Version 6 (IPv6) Specification / RFC Editor. Version: Juli 2017. `https://tools.ietf.org/html/rfc8200`. RFC Editor, Juli 2017 (8200). – RFC. – 1–42 S.. – ISSN 2070–1721

[Enz19] Enzinger, Marinus: *Efficiently Re-Keying Multicast Groups with LKH in G-IKEv2*, Ludwig-Maximilians-Universität München, Diplomarbeit, 2019

[Far18] Farrell, S.: Low-Power Wide Area Network (LPWAN) Overview / RFC Editor. RFC Editor, May 2018 (8376). – RFC. – ISSN 2070–1721

[FK11] Frankel, S. ; Krishnan, S.: IP Security (IPsec) and Internet Key Exchange (IKE) Document Roadmap / RFC Editor. Version: Februar 2011. `https://tools.ietf.org/html/rfc6071`. RFC Editor, Februar 2011 (6071). – RFC. – 1–63 S.. – ISSN 2070–1721

[FM98] Friend, R. ; Monsour, R.: IP Payload Compression Using LZS / RFC Editor. RFC Editor, December 1998 (2395). – RFC. – ISSN 2070–1721

[GG16] Gupta, Reetu ; Gupta, Rahul: ABC of Internet of Things: Advancements, benefits, challenges, enablers and facilities of IoT. In: *2016 Symposium on Colossal Data Analysis and Networking (CDAN)* IEEE, 2016, S. 1–5

[GK98] Glenn, R. ; Kent, S.: The NULL Encryption Algorithm and Its Use With IPsec / RFC Editor. RFC Editor, November 1998 (2410). – RFC. – ISSN 2070–1721

[GP19] Gimenez, Olivier ; Petrov, Ivaylo: Static Context Header Compression (SCHC) over LoRaWAN / IETF Secretariat. Version: December 2019. `http://www.ietf.org/internet-drafts/draft-ietf-lpwan-schc-over-lorawan-05.txt`. 2019 (draft-ietf-lpwan-schc-over-lorawan-05). – Internet-Draft. – `http://www.ietf.org/internet-drafts/draft-ietf-lpwan-schc-over-lorawan-05.txt`

[Hei17] Heider, Tobias: *Minimal G-IKEv2 implementation for RIOT OS*, Ludwig-Maximilians-Universität München, Diplomarbeit, 2017

[Hei19] Heider, Tobias: *Towards a Verifiable Secure Quantum-Resistant Key Exchange in IKEv2*, Ludwig-Maximilians-Universität München, Diplomarbeit, 2019

[Hou04] Housley, R.: Using Advanced Encryption Standard (AES) Counter Mode With IPsec Encapsulating Security Payload (ESP) / RFC Editor. RFC Editor, January 2004 (3686). – RFC. – ISSN 2070–1721

[HT11] Hui, J. ; Thubert, P.: Compression Format for IPv6 Datagrams over IEEE 802.15.4-Based Networks / RFC Editor. Version: September 2011. `https://tools.ietf.org/html/rfc6282`. RFC Editor, September 2011 (6282). – RFC. – 1–24 S.. – ISSN 2070–1721

[Ken05] Kent, S.: IP Encapsulating Security Payload (ESP) / RFC Editor. Version: December 2005. `https://tools.ietf.org/html/rfc4303`. RFC Editor, December 2005 (4303). – RFC. – 1–44 S.. – ISSN 2070–1721

[KHN⁺14] Kaufman, C. ; Hoffman, P. ; Nir, Y. ; Eronne, P. ; Kivinen, T.: Internet Key Exchange Protocol Version 2 (IKEv2) / RFC Editor. Version: Oktober 2014. `https://tools.ietf.org/html/rfc7296`. RFC Editor, Oktober 2014 (7296). – RFC. – 1–142 S.. – ISSN 2070–1721

[KK05] Kent, S. ; K.Seo: Security Architecture for the Internet Protocol / RFC Editor. Version: Dezember 2005. `https://tools.ietf.org/html/rfc4301`. RFC Editor, Dezember 2005 (4301). – RFC. – 1–101 S.. – ISSN 2070–1721

[LKH+18] Lenders, Martine ; Kietzmann, Peter ; Hahm, Oliver ; Petersen, Hauke ; Gündoğan, Cenk ; Baccelli, Emmanuel ; Schleiser, Kaspar ; Schmidt, Thomas C. ; Wählisch, Matthias: Connecting the world of embedded mobiles: The RIOT approach to ubiquitous networking for the Internet of Things. In: *arXiv preprint arXiv:1801.02833* (2018)

[Mal19] Malkus, Maximilian-Matthias: *Implementing IPsec ESP in RIOT OS*, Ludwig-Maximilians-Universität München, Diplomarbeit, 2019

[MBCM19] Mekki, Kais ; Bajic, Eddy ; Chaxel, Frederic ; Meyer, Fernand: A comparative study of LPWAN technologies for large-scale IoT deployment. In: *ICT express* 5 (2019), Nr. 1, S. 1–7

[MGBS19] Migault, Daniel ; Guggemos, Tobias ; Bormann, Carsten ; Schinazi, David: ESP Header Compression and Diet-ESP / IETF Secretariat. Version: March 2019. `http://www.ietf.org/internet-drafts/draft-mglt-ipsecme-diet-esp-07.txt`. 2019 (draft-mglt-ipsecme-diet-esp-07). – Internet-Draft. – `http://www.ietf.org/internet-drafts/draft-mglt-ipsecme-diet-esp-07.txt`

[MGS18] Migault, Daniel ; Guggemos, Tobias ; Schinazi, David: Internet Key Exchange version 2 (IKEv2) extension for the ESP Header Compression (EHC) Strategy / IETF Secretariat. Version: June 2018. `http://www.ietf.org/internet-drafts/draft-mglt-ipsecme-ikev2-diet-esp-extension-01.txt`. 2018 (draft-mglt-ipsecme-ikev2-diet-esp-extension-01). – Internet-Draft. – `http://www.ietf.org/internet-drafts/draft-mglt-ipsecme-ikev2-diet-esp-extension-01.txt`

[MKHC07] Montenegro, G. ; Kushalnagar, N. ; Hui, J. ; Culler, D.: Transmission of IPv6 Packets over IEEE 802.15.4 Networks / RFC Editor. Version: September 2007. `https://tools.ietf.org/html/rfc4944`. RFC Editor, September 2007 (4944). – RFC. – 1–30 S.. – ISSN 2070–1721

[MTG+20] Minaburo, A. ; Toutain, L. ; Gomez, C. ; Barthel, D. ; Zúñiga, JC.: SCHC: Generic Framework for Static Context Header Compression and Fragmentation / RFC Editor. Version: April 2020. `https://tools.ietf.org/html/rfc8724`. RFC Editor, April 2020 (8724). – RFC. – 1–71 S.. – ISSN 2070–1721

[NBBB98] Nichols, Kathleen ; Blake, Steven ; Baker, Fred ; Black, David L.: Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers / RFC Editor. Version: December 1998. `http://www.rfc-editor.org/rfc/rfc2474.txt`. RFC Editor, December 1998 (2474). – RFC. – ISSN 2070–1721. – `http://www.rfc-editor.org/rfc/rfc2474.txt`

[PMP+17] Petäjäjärvi, Juha ; Mikhaylov, Konstantin ; Pettissalo, Marko ; Janhunen, Janne ; Iinatti, Jari: Performance of a low-power wide-area network based on LoRa technology: Doppler robustness, scalability, and coverage. In: *International Journal of Distributed Sensor Networks* 13 (2017), Nr. 3, S. 1550147717699412

[PW17] Patel, Dhaval ; Won, Myounggyu: Experimental study on low power wide area networks (LPWAN) for mobile Internet of Things. In: *2017 IEEE 85th Vehicular Technology Conference (VTC Spring)* IEEE, 2017, S. 1–5

[QGZ+19] Queralta, J P. ; Gia, Tuan N. ; Zou, Z ; Tenhunen, Hannu ; Westerlund, T: Comparative study of LPWAN technologies on unlicensed bands for M2M communication in the IoT: Beyond LoRa and LoRaWAN. In: *Procedia Computer Science* 155 (2019), S. 343–350

[Ray20] Ray, Bill: Hype Cycle for IoT Standards and Protocols, 2020. In: *Gartner Inc.* (2020)

[RFB01] Ramakrishnan, K. ; Floyd, S. ; Black, D.: The Addition of Explicit Congestion Notification (ECN) to IP / RFC Editor. Version: September 2001. `http://www.rfc-editor.org/rfc/rfc3168.txt`. RFC Editor, September 2001 (3168). – RFC. – ISSN 2070–1721. – `http://www.rfc-editor.org/rfc/rfc3168.txt`

[Sel17] Seller, Olivier Bernard A.: *Wireless Communication Method.* `https://patentscope.wipo.int/search/en/detail.jsf?docId=US160611627`. Version: 2017

[Sem19] Semtech: LoRa® and LoRaWAN®:A Technical Overview / Semtech Corporation. Version: 2019. `https://lora-developers.semtech.com/library/tech-papers-and-guides/lora-and-lorawan/`. 2019. – Forschungsbericht

[Sig] Sigfox: *Sigfox, The world's leading service provider for Internet of Things (IoT).* `https://www.sigfox.com/`, . – Last accessed 27.07.2020

[SMPT01] Shacham, A. ; Monsour, B. ; Pereira, R. ; Thomas, M.: IP Payload Compression Protocol (IPComp) / RFC Editor. RFC Editor, September 2001 (3173). – RFC. – ISSN 2070–1721

[Smy20] Smyslov, Valery: Using compression in the Internet Key Exchange Protocol Version 2 (IKEv2) / IETF Secretariat. Version: March 2020. `http://www.ietf.org/internet-drafts/draft-smyslov-ipsecme-ikev2-compression-09.txt`. 2020 (draft-smyslov-ipsecme-ikev2-compression-09). – Internet-Draft. – `http://www.ietf.org/internet-drafts/draft-smyslov-ipsecme-ikev2-compression-09.txt`

[SPJ10] Sandlund, K. ; Pelletier, G. ; Jonsson, L-E.: The RObust Header Compression (ROHC) Framework / RFC Editor. Version: März 2010. `https://tools.ietf.org/html/rfc5795`. RFC Editor, März 2010 (5795). – RFC. – 1–41 S.. – ISSN 2070–1721

[SW20] Smyslov, Valery ; Weis, Brian: Group Key Management using IKEv2 / IETF Secretariat. Version: July 2020. `http://www.ietf.org/internet-drafts/draft-ietf-ipsecme-g-ikev2-01.txt`. 2020 (draft-ietf-ipsecme-g-ikev2-01). – Internet-Draft. – `http://www.ietf.org/internet-drafts/draft-ietf-ipsecme-g-ikev2-01.txt`

[XYWV12] Xia, Feng ; Yang, Laurence T. ; Wang, Lizhe ; Vinel, Alexey: Internet of things. In: *International journal of communication systems* 25 (2012), Nr. 9, S. 1101

[ZGT20] Zuniga, Juan ; Gomez, Carles ; Toutain, Laurent: SCHC over Sigfox LPWAN / IETF Secretariat. Version: July 2020. `http://www.ietf.org/internet-drafts/draft-ietf-lpwan-schc-over-sigfox-03.txt`. 2020 (draft-ietf-lpwan-schc-over-sigfox-03). – Internet-Draft. – `http://www.ietf.org/internet-drafts/draft-ietf-lpwan-schc-over-sigfox-03.txt`