Resource-Conscious Network Security for the IP-Based Internet of Things

Von der Fakultät für Mathematik, Informatik und Naturwissenschaften der RWTH
Aachen University zur Erlangung des akademischen Grades eines Doktors der
Naturwissenschaften genehmigte Dissertation

vorgelegt von

Diplom-Informatiker

René Hummen

aus Aachen, Deutschland

Berichter:

Prof. Dr.-Ing. Klaus Wehrle
Prof., Ph.D. Thiemo Voigt

Tag der mündlichen Prüfung: 02.06.2015

Diese Dissertation ist auf den Internetseiten der Hochschulbibliothek online verfügbar.

Abstract

The Internet of Things (IoT) envisions an unprecedented interleaving of the physical with the virtual world to enhance automation and to improve comfort in a variety of application domains ranging from home automation to healthcare and smart cities. Based on recent advances in standardization, many of these application domains are expected to employ IP-enabled embedded devices to realize the envisioned interconnection of the physical world. Such IP connectivity, however, also exposes networked embedded devices to similar network attacks as conventional IP-enabled hosts or services. The severity of these attacks is considerably aggravated in the IoT as attacks in the virtual world suddenly can have detrimental physical impact. Hence, effective network security is a vital precondition for a secure IP-based IoT.

Standard end-to-end security protocols such as TLS have the potential to provide an important building block for interoperable network security in the IoT. The device and network constraints in the embedded domain and the resource asymmetry in the IoT, however, challenge the design of existing security solutions. The resource constraints of embedded devices, e.g., require these solutions to be applicable in the context of only a few MHz of computational power, several kB of RAM, and several tens of kB of ROM. Similarly, energy constraints and low-power wireless communication demand for a high transmission efficiency. Research and standardization, thus, recently started to adapt standard IP security solutions to IoT requirements.

In this thesis, we contribute to these adaptation efforts by addressing emerging protocol design challenges for end-to-end IP security in the context of the IoT. In this, we specifically consider the IoT security protocol adaptations DTLS, HIP DEX, and Minimal IKEv2 that are currently proposed for standardization at the IETF. Notably, while these protocol adaptations should already satisfy IoT requirements, we identify several design-level efficiency and security issues that render the deployment of these protocols in their current state inefficient, infeasible, and even insecure.

First, the high computation overhead of DTLS, HIP DEX, and Minimal IKEv2 significantly hampers the availability and response time of networked embedded devices during the protocol handshake. We present three complementary protocol extensions that account for these computation overheads in the overall protocol design. Second, the extensive message wire-format of these protocol adaptations leads to undesirable transmission overheads in the embedded domain. We devise the Slimfit compression layer that addresses message conciseness issues in the context of HIP DEX. Combined, these two contributions considerably reduce the run-time overheads and improve the security properties of the considered end-to-end security protocols.

Third, extensive RAM and ROM requirements render the use of DTLS, HIP DEX, and Minimal IKEv2 infeasible for a wide range of memory-constrained embedded devices. To still enable these devices to communicate securely, we introduce the handshake delegation architecture that also provides an authorization framework for the embedded domain. Fourth, the 6LoWPAN packet fragmentation of the DTLS, HIP DEX, and Minimal IKEv2 handshake messages is vulnerable to DoS attacks. To protect against these attacks, we present two lightweight defense mechanisms.

Overall, our contributions in this thesis effectively complement each other and, in combination, achieve significant security and efficiency improvements for the considered standard end-to-end security protocols in the context of the IP-based IoT.

Kurzfassung

Die Vision des Internets der Dinge ist eine bisher unerreichte Vernetzung der physischen mit der virtuellen Welt. Hiervon sollen zum Beispiel die Hausautomatisierung aber auch neuartige Anwendungsbereiche wie die intelligente Stadt profitieren. Aktuelle Fortschritte bei der Standardisierung deuten dabei auf einen verstärkten Einsatz von IP-fähigen eingebetteten Systemen hin. Die einhergehende Erreichbarkeit macht vernetzte „Dinge“ jedoch ähnlich wie herkömmliche Rechner und Dienste über das Netzwerk angreifbar. Effektive Sicherheitslösungen sind daher eine wesentliche Voraussetzung für die sichere Vernetzung der physischen mit der virtuellen Welt.

Standardprotokolle für die Ende-zu-Ende-Sicherheit wie TLS haben das Potenzial einen wichtigen Bestandteil für diese sichere Vernetzung zu liefern. Die Geräte- und Netzwerkbeschränkungen im Bereich der eingebetteten Systeme sowie die Ressourcen-Asymmetrie im Internet der Dinge stellen bestehende Sicherheitslösungen jedoch vor enorme Herausforderungen. So setzen die knappen Ressourcen eingebetteter Systeme voraus, dass Lösungen bei stark beschränkter Rechenleistung und begrenztem Speicherplatz einsetzbar sind. Ebenso erfordern Energiebeschränkungen eine hohe Verarbeitungs- und Übertragungseffizienz. Daher müssen existierende Sicherheitslösungen an die speziellen Anforderungen im Internet der Dinge angepasst werden.

Diese Arbeit adressiert grundlegende Herausforderungen beim Entwurf von Ende-zu-Ende-IP-Sicherheitsprotokollen im Internet der Dinge. Hierbei liegt der Fokus auf den Protokollanpassungen DTLS, HIP DEX und Minimal IKEv2. Während diese Lösungen bereits den Anforderungen des Internets der Dinge genügen sollten, identifiziert diese Arbeit diverse Effizienz- und Sicherheitsfragen, die den Einsatz dieser Protokollanpassungen ineffizient, unmöglich, oder gar unsicher machen. Um diesen Problemstellungen zu begegnen, umfasst diese Arbeit insgesamt vier Beiträge.

Die durchgeführten Protokollanalysen zeigen, dass der erhebliche Berechnungsaufwand während der Protokollaushandlung die Verfügbarkeit und die Antwortzeit von eingebetteten Systemen deutlich beeinträchtigt. Der erste Beitrag besteht daher aus Protokollerweiterungen, die eine Berücksichtigung dieser Berechnungskosten ermöglichen. Darüber hinaus deuten die Analyseergebnisse auf umfangreiche Kompressionspotenziale bei den ausgetauschten Nachrichten hin. Zur Umsetzung dieser Potenziale bei HIP DEX führt der zweite Beitrag die Slimfit-Kompressionsschicht ein. Die Kombination dieser beider Beiträge erreicht eine deutliche Reduktion der Laufzeitkosten sowie eine wesentliche Verbesserung der Protokoll-Sicherheitseigenschaften.

Weiterhin decken die vorgenommenen Protokollanalysen umfangreiche Speicheranforderungen im Bezug auf die betrachteten Ende-zu-Ende-Sicherheitsprotokolle auf. Diese Anforderungen machen deren Einsatz auf stark speicherbeschränkten eingebetteten Systemen unmöglich. Der dritte Beitrag stellt eine Delegationsarchitektur vor, um diesen Geräten dennoch eine sichere Ende-zu-Ende-Kommunikation zu ermöglichen. Schließlich identifiziert die Analyse der 6LoWPAN-Anpassungsschicht die Anfälligkeit des dort eingesetzten Fragmentierungsmechanismus für DoS-Angriffe bei der DTLS-, HIP DEX- und Minimal IKEv2-Protokollaushandlung. Zum Schutz vor diesen Angriffen präsentiert der vierte Beitrag leichtgewichtige Abwehrmechanismen.

Die vorgestellten Beiträge lassen sich wirkungsvoll miteinander kombinieren und erzielen so erhebliche Sicherheits- und Effizienzsteigerungen für die betrachteten Ende-zu-Ende-IP-Sicherheitsprotokolle im Kontext des Internets der Dinge.

Acknowledgments

A number of people have contributed to my research, which ultimately resulted in this thesis. To all of you, a big and special **thank you**!

First and foremost, I would like to express my gratitude to my advisor Klaus Wehrle for asking me to join his group right after my diploma thesis. Without this initial impulse, I might not have embarked on this intriguing journey called the “PhD”. I am especially grateful to you, Klaus, for allowing me to freely pursue my own research interests and – at the same time – cannot thank you enough for your encouragement when finding an attractive research topic proved more challenging than first anticipated. I also thank Thiemo Voigt for welcoming me to SICS for a brief stay, for accepting to review my thesis, and for acting as the second opponent during my PhD defense. The feedback I received from you and your team definitely had a positive impact on my work and the presentation of my research in this thesis.

During my time as a PhD student, I had the pleasure to work with a number of very interested and motivated students. Here, I would like to especially mention Christian Röller, Henrik Ziegeldorf, Hossein Shafagh, Jens Hiller, Maximilian Bogner, and Timo Boetcher. You should all be able to find aspects of your work reflected in this thesis. To all 21, I learned a lot from each of you, professionally as well as personally! I will always remember our fruitful discussions about design improvements, implementation details, and of course also about non-work-related aspects of life. I am particularly proud to see that some of you became PhD students yourself and it is always very exciting to meet you guys in your new jobs after COMSYS.

Over the last couple of years, I also had the chance to closely work with several extremely talented researchers. One of them, Tobias Heer, sparked my interest in scientific research. I owe you my gratitude, Tobi, for encouraging me to start the PhD, for always offering helpful advice, and, equally important, for all the good times as a friend. Miika Komu woke my interest in the HIP protocol in particular and in end-to-end IP security in general. This thesis surely would have an entirely different scope without you, Miika. Moreover, I would like to thank Hanno Wirtz for his painstakingly honest and always spot-on feedback during paper and thesis writing. I could not have wished for better colleagues than Martin Henze and Nicolai Viol regarding all project-related tasks and I am grateful to Georg Kunz for ensuring the necessary creativity and fun in research. Besides these people, I would also like to thank all other PhD students and Postdocs at COMSYS. I am very proud to have worked among you highly talented, motivated, and supportive people. This of course also includes the administrative and technical staff of COMSYS. Thanks, Ulrike May, Petra Zeidler, and Rainer Krogull, for your practical help and patience.

While pursuing my research, I additionally had the opportunity to meet many highly skilled and supportive people at the IETF and beyond. A special “thanks” goes to Robert Moskowitz for his invitation to co-author the HIP DEX Internet-Draft as well as for the conversations on- and off-topic during various IETF meetings. Moreover, I would like to thank Stefanie Gerdes and Shahid Raza for inviting me to contribute to their research papers. A big “thank you” also goes to Johannes Gilger for helping me with the write-up of one of my Internet-Drafts. It was a pleasure to collaborate with all of you. I am glad to have met Ari Keränen and Matthias Kovatsch, who not only helped me network at the IETF but with whom I could also tour the different meeting locations. Finally, I am grateful to Carsten Bormann, Oscar García-Morcón, Klaus Hartke, Sandeep Kumar, Ludwig Seitz, René Struik and many others for all the interesting discussions revolving around network security for IP-enabled embedded devices and the motivation I could draw from these valuable conversations.

Last but certainly not least, my PhD journey was only possible thanks to the unconditional support of my family. Inge, Wilfried, and Torsten, I highly appreciate your constant stream of encouragement and your help when things did not go exactly as planned. Above all, I am deeply grateful to you, Cristina, for standing by my side, for sacrificing many evenings of leisure time, and for always being there for me!

Contents

1 Introduction 1

1.1 Problem Space and Goal of This Thesis 2
1.2 Contributions 4
1.2.1 Tailored Protocol Handshake Mechanisms 4
1.2.2 Message Wire-Format Compression 5
1.2.3 Handshake Delegation Architecture 6
1.2.4 Secure 6LoWPAN Fragmentation 6
1.3 Interplay of our Contributions 7
1.4 Genesis and Attribution of our Contributions 8
1.5 Thesis Outline 9

2 The IP-based Internet of Things 11

2.1 Application Scenarios and Network Traffic Flows 11
2.2 Special Characteristics in the Embedded Domain 13
2.2.1 Constrained Devices 13
2.2.2 Wireless Technologies and Constrained Node Networks 14
2.3 Interconnecting Constrained Devices with IP 15
2.3.1 Network Architecture of the IP-based Internet of Things . . . 16
2.3.2 The Adapted IP Network Stack for Constrained Devices . . . 17
2.4 The 6LoWPAN Adaptation Layer for IPv6 20
2.4.1 6LoWPAN Header Compression Facilities 20
2.4.2 6LoWPAN Packet Fragmentation 22
2.5 Summary 24

3 Network Security in the IP-based IoT and Problem Statement 25

3.1 Network Threats in the IP-based IoT 26
  3.1.1 The Internet Threat Model 26
  3.1.2 Eavesdropping and Traffic Analysis 26
  3.1.3 Packet Injection and Modification Attacks 27
  3.1.4 Impersonation Attacks and Access Violations 28
  3.1.5 Denial of Service Attacks 28
3.2 Cryptography and Constrained Devices 29
  3.2.1 Fundamental Objectives of Cryptography 29
  3.2.2 Symmetric-Key Cryptography 30
  3.2.3 Public-key Cryptography 31
  3.2.4 Cryptographic Hash Functions 33
  3.2.5 Message Authentication and Integrity Protection 35
3.3 Network Security in the IP-based IoT 36
  3.3.1 Additional Objectives of Network Security 36
  3.3.2 Overview of Protocol-based Security Solutions 37
  3.3.3 Datagram Transport Layer Security 39
  3.3.4 Host Identity Protocol Diet EXchange 41
  3.3.5 Minimal Internet Key Exchange Protocol Version 2 43
  3.3.6 Protocol Comparison 44
3.4 Problem Statement and Research Challenges 46

4 Tailoring the Protocol Design 49

4.1 Designing for High Computation Requirements 50
  4.1.1 Impact on the Protocol Handshake 50
  4.1.2 Reducing the Cost of Repeated Connection Establishments . . 54
  4.1.3 DoS Protection Despite Resource Asymmetries 62
  4.1.4 Accounting for Varying Message Processing Times 68
  4.1.5 Security Considerations 72
  4.1.6 Evaluation 75
  4.1.7 Related Work 84
  4.1.8 Summary 86
4.2 Tailoring the Transmission Overhead 87

4.2.1 Analysis of the HIP DEX Message Wire-Format 88
4.2.2 The Slimfit Compression Layer 92
4.2.3 Evaluation 97
4.2.4 Security Considerations 102
4.2.5 Related Work 103
4.2.6 Summary 105
4.3 Conclusion 106

5 Delegating the Protocol Handshake 107

5.1 Memory Analysis for DTLS 108
5.1.1 RAM Requirements 109
5.1.2 ROM Requirements 110
5.1.3 Remarks About the Combined RAM and ROM Impact 111
5.1.4 The Need for Lightweight Key Provisioning 111
5.2 The Handshake Delegation Architecture 112
5.2.1 Entities and Assumptions 112
5.2.2 Lightweight Key Provisioning 113
5.2.3 Authorizing Inter- and Intra-Domain Communication 118
5.2.4 Integration with HIP DEX and Minimal IKEv2 122
5.3 Security Considerations 123
5.3.1 Impact of a Compromised Remote End-Point 123
5.3.2 Impact of a Compromised Delegation Server 124
5.3.3 Impact of a Compromised Constrained Device 124
5.4 Evaluation 124
5.4.1 Reduced RAM and ROM Requirements 126
5.4.2 Additional Run-time Improvements 128
5.5 Related Work 131
5.5.1 Delegation-based Approaches for End-to-End IP Security . . . 131
5.5.2 Authorization Frameworks for the IP-based IoT 133
5.6 Conclusion 135

6 Secure 6LoWPAN Fragmentation 137
6.1 Overview of Existing Fragmentation Attacks . . . 138
  6.1.1 Attacks Based on Implementation Deficiencies . . . 138
  6.1.2 Attacks Based on Design-Level Vulnerabilities . . . 138
6.2 The Assumed Attacker Model . . . 141
  6.2.1 Remarks About the Network-External Attacks . . . 142
6.3 The 6LoWPAN Fragmentation Attacks . . . 143
  6.3.1 The Fragment Duplication Attack . . . 143
  6.3.2 The Buffer Reservation Attack . . . 144
  6.3.3 Susceptibility to the 6LoWPAN Fragmentation Attacks . . . 146
6.4 The Content-Chaining Scheme . . . 149
  6.4.1 Partial and Non-Solutions . . . 149
  6.4.2 High-level Overview of the Content-Chaining Scheme . . . 150
  6.4.3 A Basic Fragment-Chaining Approach . . . 151
  6.4.4 Construction of a Content Chain . . . 152
  6.4.5 Verifying the Tokens of a Content Chain . . . 152
  6.4.6 Overhead Reduction Techniques . . . 154
6.5 The Split Buffer Approach . . . 157
  6.5.1 Partial and Non-Solutions . . . 157
  6.5.2 High-level Overview of the Split Buffer Approach . . . 157
  6.5.3 Fragment-sized Buffer Slots . . . 158
  6.5.4 Packet Discard Strategy . . . 159
6.6 Security Considerations . . . 163
  6.6.1 Attacks Against the Basic Content-Chaining Scheme . . . 164
  6.6.2 Attacks Targeting the Overhead Reduction Techniques . . . 165
  6.6.3 Attacks Against the Split-Buffer Approach . . . 166
6.7 Evaluation . . . 167
  6.7.1 Effective Defense Against the Identified Attacks . . . 168
  6.7.2 Transmission Overhead . . . 171
  6.7.3 Computation Overhead . . . 173
  6.7.4 Energy Evaluation . . . 176
  6.7.5 RAM and ROM Overhead . . . 177

6.8 Related Work 178
    6.8.1 Fragmentation Attacks and Defense Mechanisms 179
    6.8.2 Efficient Authentication Schemes for Packet Streams 180
    6.8.3 Related Non-Cryptographic Approaches 180
6.9 Conclusion 182

7 Discussion and Conclusion **183**
7.1 Contributions and Achievements 184
    7.1.1 Tailored Protocol Handshake Mechanisms 184
    7.1.2 Message Wire-Format Compression 185
    7.1.3 Handshake Delegation Architecture 186
    7.1.4 Secure 6LoWPAN Fragmentation 187
7.2 Impact at the Time of Writing 187
7.3 Future Research 188
    7.3.1 Cross-Domain Device-to-Device Authorizations 188
    7.3.2 Selective End-to-End Payload Security for the IoT 189
    7.3.3 End-Point-Assisted In-Network Security 191
7.4 Concluding Remarks 191

Glossary **193**

Bibliography **196**

1

Introduction

Since its introduction in the early 1980s, the Internet Protocol (IP) suite [Bra89] has become the foundation for the interconnection of a vast variety of network entities ranging from mobile personal devices to cloud-based services. With the proliferation of the *Internet of Things (IoT)*, a new class of network entities – objects from the physical world – is about to join the existing IP infrastructure. This interconnection is envisioned to enable an unprecedented interleaving of the physical with the virtual world [MF10]. Home appliances, e.g., are at the verge of being networked and controllable via smartphones or cloud-based services. Likewise, industrial machines increasingly become interconnected in factory networks. This connectivity, in turn, enables their autonomous organization as well as human monitoring and control from anywhere on the factory floor. Similar to these examples, the interconnection of the physical world in the IoT is foreseen to benefit a wide range of other application domains including healthcare, logistics, and smart cities [AIM10, KKV12].

Many of these application domains are expected to employ *networked embedded devices* for the envisioned interconnection of the physical world [WTJ$^+$11, SB10]. Such embedded devices often are designed for small size, low cost, and low energy consumption, thus enabling long-term autonomous operation. This design, in turn, leads to devices with highly limited computation power and scarce memory resources [BEK14]. Similarly, the low-power radio technologies employed by *wirelessly* networked embedded devices typically are limited to short-range links, low bandwidth, and small packet sizes [IEEE11b, BT13]. These *device and network constraints*, however, render viable networking solutions for the IoT challenging.

To facilitate the interconnection of networked embedded devices, research in the field of Wireless Sensor Networks (WSNs) as well as first commercial IoT products primarily focused on the development of application-specific solutions [YMG08, homematic]. The high specialization of the devised network stacks, the lack of a uniform addressing scheme, and the resulting need for application-level gateways, however, *thwart the interoperability* between a growing number of application- and vendor-specific solutions. Prompted by research showing the practicability of IP technology for networked embedded devices [Dun03], the Internet Engineering Task

Force (IETF) began adapting the IP network architecture and its supporting protocols to the special device and network characteristics in the embedded domain in 2005 [6lowpan]. With the goal to continue the eminent success of the Internet in the embedded domain, these standardization efforts most notably aim at providing *interoperable* and *efficient* IP connectivity for networked embedded devices. The resulting converged network architecture enables seamless end-to-end communication in an *IP-based IoT*[1] by employing standardized protocols at the network layer and above. In addition, this architecture also accommodates for domain-specific networking solutions below the network layer. Such flexibility affords an efficient operation of IP technology in the context of networked embedded devices [HC08].

The achieved IP connectivity, however, also exposes networked embedded devices to similar *network attacks* as conventional IP-enabled hosts or services. These attacks particularly include eavesdropping, impersonation, Denial-of-Service (DoS), and system compromise. The potentially devastating *physical impact* of these attacks in the context of the IoT has recently been shown by the Stuxnet virus [Lan11], the first known digital attack targeting objects from the physical world. This attack led to the complete destruction of the targeted industrial machinery by injecting malicious actuation commands into their embedded control units. As objects from the physical world increasingly rely on information from remote entities to perform their tasks in the physical environment, similar incidents are prone to occur on much larger scales (e.g., smart grid) as well as in highly individual scenarios (e.g., personal health services). Consequently, effective network security is a crucial requirement for the secure interconnection of objects, hosts, and services in the IP-based IoT.

With the goal to tackle these security requirements, an active community in the field of IoT security currently investigates the applicability of *standard end-to-end IP security protocols* such as Transport Layer Security (TLS) [DR08] for networked embedded devices. However, while initial results prove the general feasibility of such interoperable network security [GMF$^+$05], existing security protocols were originally designed with extensibility and flexibility as well as with the protection of comparably powerful Internet hosts and services in mind. As a result, their protocol specifications contain design decisions that *hinder the efficient operation* of these protocols in the context of networked embedded devices. Hence, research and standardization recently started *adapting* end-to-end IP security protocols to the special device and network characteristics in the embedded domain. On the one hand, these efforts comprise the development of protocol profiles and variants such as the Host Identity Protocol Diet EXchange (HIP DEX) [MH14] that reduce protocol complexity by limiting the flexibility of the original protocol specification. On the other hand, they involve the integration of alternative cryptographic primitives, e.g., polynomial schemes [KKG10, GMKK$^+$13b] and implicit certificates [PKG$^+$13], in existing end-to-end IP security protocols to reduce their cryptographic overhead.

1.1 Problem Space and Goal of This Thesis

IP-enabled networked embedded devices afford seamless end-to-end communication in the IoT. The device and network characteristics in the embedded domain, however, significantly differ from those typically encountered in conventional IP net-

---

[1] From now on, we use the terms “IoT” and “IP-based IoT” synonymously.

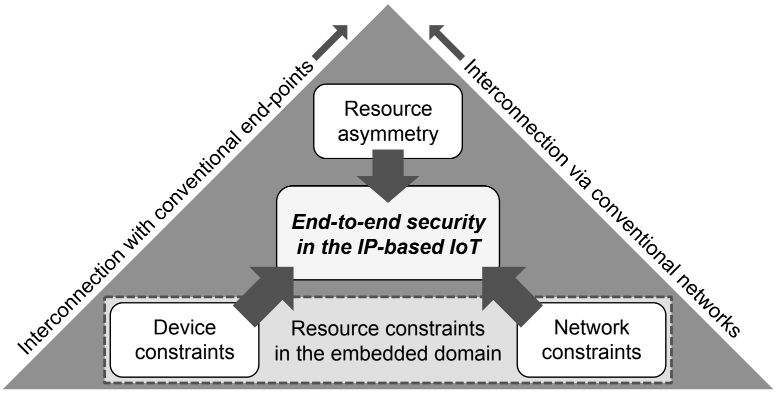

Interconnection with conventional end-points
Interconnection via conventional networks
Resource asymmetry
End-to-end security in the IP-based IoT
Device constraints
Resource constraints in the embedded domain
Network constraints

Figure 1.1 Main factors that challenge the design adequacy of standard end-to-end IP security protocols in the context of networked embedded devices and the IP-based IoT.

works. As shown in Figure 1.1, these disparate characteristics challenge the design adequacy of standard end-to-end IP security solutions in the context of the IoT.

More precisely, the *device constraints* in the embedded domain require end-to-end security solutions to be applicable in the context of only a few MHz of computational power, several kilobytes of RAM, several tens of kilobytes of ROM, and limited energy resources [BEK14]. At the same time, these solutions have to provide adequate security guarantees to protect network communication in the context of conventional hosts or services. Hence, while end-to-end security solutions for the IoT specifically have to be designed for computation, energy, and memory efficiency, the employed cryptographic primitives still need to provide a level of security that complies with current security recommendations for Internet-based communication. Moreover, the security protocol design has to account for the inherent *resource asymmetry* between networked embedded devices and conventional hosts or services, which an adversary may exploit, e.g., to mount a DoS attack against embedded devices in the IoT.

In addition, the design of end-to-end security solutions also has to cater to the *network constraints* that are caused by the low-power radio technologies of wirelessly networked embedded devices. These particularly include low-bandwidth links and small packet sizes, e.g., 250 kbit/s and 127 byte in case of IEEE 802.15.4 [IEEE11b], respectively. Moreover, the employed radio technologies often exhibit packet loss ratios that exceed those of conventional IP networks [SDTL10]. The efficient utilization of network resources, thus, is another important aspect for the design of end-to-end security solutions. This observation especially holds for energy-constrained embedded devices as radio transmissions consume a significant amount of energy [PK00].

Considering these device and network constraints in the embedded domain as well as the high resource asymmetry between networked embedded devices and conventional hosts or services, our **main goal** in this thesis is to comprehensively address the emerging protocol design challenges for end-to-end security in the context of the IP-based IoT. In this, we specifically focus on the following two key aspects:

1. *Security protocol efficiency* with respect to the device and network constraints in the embedded domain and
2. *Protocol security* regarding the resource constraints in the embedded domain as well as the resource asymmetry in the IoT.

We now continue with a brief overview of our core contributions in this thesis and then describe how these contributions address the above goal with its two aspects.

1.2 Contributions

In pursuing the main goal of this thesis, we aim at contributing to the on-going efforts of adapting IP technology to IoT requirements. Hence, we leverage recently proposed IoT security protocol adaptations of the Datagram Transport Layer Security (DTLS) protocol [RM12], the Host Identity Protocol Version 2 (HIPv2) [MHJH15], and the Internet Key Exchange Protocol Version 2 (IKEv2) [KHN+14] as a starting point.

Complementary to related work, our research focuses on protocol design and architectural considerations beyond protocol profiling and based on standard cryptographic primitives. Moreover, we acknowledge the fact that a comprehensive security analysis also requires a thorough review of the underlying transport mechanisms. We, therefore, additionally investigate the security implications of the Internet Protocol version 6 (IPv6) adaptations performed at the IPv6 over Low-Power Wireless Personal Area Networks (6LoWPAN) layer inside the embedded domain[2]. Overall, the contributions in this thesis fall into the following two categories:

Protocol analyses: We identify protocol *efficiency* and *security* issues in the design of the DTLS profile for the IoT[3] [TF15], HIP DEX [MH14], and the Minimal Internet Key Exchange Protocol Version 2 (Minimal IKEv2) [Kiv15] as well as in the design of the 6LoWPAN adaptation layer for IPv6. We note that we selected these protocols for our analyses as they denote general-purpose, application-independent network security and protocol adaptation approaches that can be employed in a wide range of IoT application domains.

Solution design: We develop resource-conscious, standard-compliant protocol mechanisms and a security architecture for the IP-based IoT that resolve the design-level protocol issues identified in the performed protocol analyses.

Our research contributes to the main goal of this thesis from different perspectives. Specifically, we investigate and address design-level issues of the considered end-to-end IP security protocol adaptations from a computation, transmission, and memory overhead point of view. Moreover, we analyze and resolve security issues of the 6LoWPAN fragmentation mechanism in the context of resource-constrained embedded devices. Hence, combined, our contributions significantly improve the efficiency and security of end-to-end security concerning the device and network constraints in the embedded domain as well as the high resource asymmetry in the IoT. We now provide a more detailed description of the *four core contributions* of this thesis as highlighted in Figure 1.2 and discuss the interplay between our developed solutions.

1.2.1 Tailored Protocol Handshake Mechanisms

The DTLS, HIP DEX, and Minimal IKEv2 protocol profiles and variants all mandate or recommend the use of public-key cryptography in their protocol specifications. As we will show in Chapter 4, the use of public-key cryptography in the

---

[2] The use of IPv6 is commonly preferred over IPv4 within the context of networked embedded devices due to the larger address space, the improved auto-configuration capabilities, and the general migration to IPv6 [HC08]. Hence, we focus our research on the IPv6 protocol adaptations.

[3] In this thesis, we use the term “DTLS” as a reference to the DTLS profile for the IoT [TF15] if not mentioned otherwise.

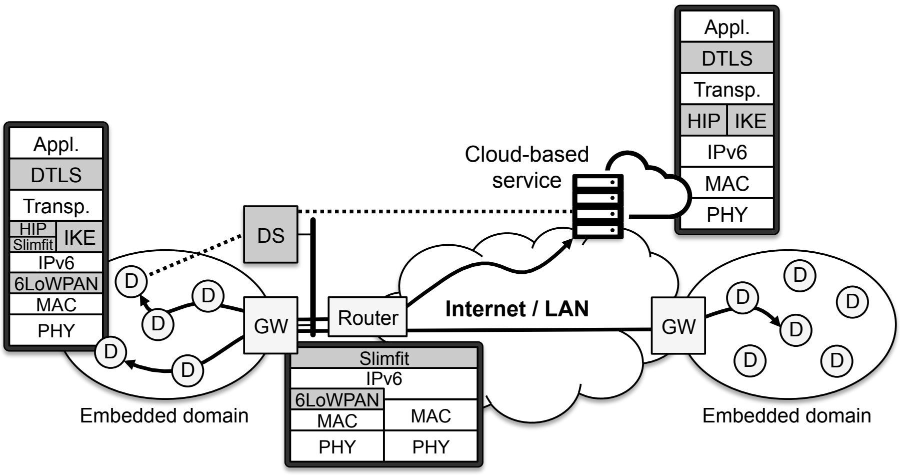


Figure 1.2 IoT network scenario consisting of networked embedded devices (D), interconnecting gateways (GW) and a cloud-based service. The layers in the network stack that are affected or newly introduced by our contributions are marked in gray. The arrows represent communication paths via local network infrastructure and potentially the global Internet. The dashed lines indicates handshake delegation via the newly introduced delegation server (DS).

context of networked embedded devices, however, causes several design-level protocol security and performance issues. Most importantly, we find that the processing time of public-key operations ranging from several hundred milliseconds to a few seconds significantly hampers the availability and response time of networked embedded devices during the protocol handshake. Hence, the design of end-to-end security protocols in the IoT must be tailored to reduce the need for computationally expensive cryptographic operations, to protect networked embedded devices against DoS attacks targeting these operations, and to account for the varying processing times of the different handshake messages. To this end, we present three complementary, lightweight protocol extensions for DTLS, HIP DEX, and Minimal IKEv2, i.e., a flexible session resumption mechanism, a collaborative puzzle-based DoS protection mechanism, and an adaptive retransmission mechanism. The evaluation results show that these protocol extensions afford significant computation and transmission reductions as well as security improvements at moderate RAM and ROM trade-offs.

1.2.2 Message Wire-Format Compression

A chief design goal of the DTLS, HIP DEX, and Minimal IKEv2 protocol specifications is to preserve the original protocol semantics of TLS, HIPv2, and IKEv2, respectively. TLS, HIPv2, and IKEv2, however, were primarily developed with extensibility and flexibility in mind. Message conciseness, consequently, only was a secondary goal. As a result, fixed-length header fields, e.g., were chosen conservatively large. To tackle these message conciseness issues, related work proposes compression mechanisms that primarily focus on the DTLS protocol [RSH$^+$13, RSD14] and the Internet Protocol Security (IPsec) suite [GMSS10, RDC$^+$11, RVJ12, MG14].

Similar to these approaches, the *Slimfit compression layer*, which we will present in Chapter 4, addresses message conciseness issues in the context of HIP DEX. Slimfit i) elides message content that is statically defined in the HIP DEX specification,

ii) rearranges the message wire-format to achieve a higher compression efficiency, and iii) introduces compression profiles to afford evolvability of the devised compression scheme. Notably, the latter two aspects distinguish the Slimfit compression layer from related compression mechanisms for end-to-end IP security protocols. The evaluation results show that Slimfit significantly reduces the transmission overhead, decreases retransmissions, and even marginally *reduces* the overall computation overhead compared to the standard HIP DEX protocol at a modest ROM overhead.

1.2.3 Handshake Delegation Architecture

Besides the computationally expensive handshake operations and the lack of a concise message wire-format, DTLS, HIP DEX, and Minimal IKEv2 also have significant RAM and ROM requirements when employing public-key cryptography during the connection establishment. As we will point out in Chapter 5, these memory overheads render the use of public-key cryptography in the design of end-to-end IP security protocols infeasible for a wide range of memory-constrained embedded devices. To still enable these devices to communicate securely via standard end-to-end IP security protocols, we present the design of the handshake delegation architecture.

The key idea behind this architecture is to separate the connection establishment from the protection of application data and to offload the connection establishment handshake to an off-path, trusted delegation server (see “DS” in Figure 1.2). By subsequently handing over the established connection context from the delegation server to the networked embedded device, the handshake delegation architecture enables this device to only implement a subset of the entire protocol specification as well as efficient symmetric-key cryptography for the protection of application data.

Moreover, the handshake delegation architecture naturally provides an authorization framework by leveraging the central role of the delegation server during the initial connection establishment handshake. Overall, the evaluation results confirm that the handshake delegation architecture considerably increases the feasibility of standard end-to-end IP security in the context of memory-constrained embedded devices.

1.2.4 Secure 6LoWPAN Fragmentation

As a final aspect, we will show in Chapter 6 that the packet processing of the DTLS, HIP DEX, and Minimal IKEv2 handshake messages at the lower layers in the network stack of networked embedded devices also impacts the security properties of the considered end-to-end IP security protocol adaptations. More precisely, we observe that the size of the messages that are transmitted during the various security protocol handshakes commonly causes fragmentation at the 6LoWPAN layer inside the embedded domain. Our security analysis of the 6LoWPAN fragmentation mechanism, however, reveals two design-level fragmentation-based DoS attacks that enable an adversary to prevent the correct reassembly of fragmented handshake messages at a target device by only sending a single protocol-compliant 6LoWPAN fragment.

To defend against these fragmentation attacks, we present two complementary, lightweight security mechanisms, i.e., a content-chaining scheme and a split buffer approach. The evaluation results confirm the practicability of the identified 6LoWPAN

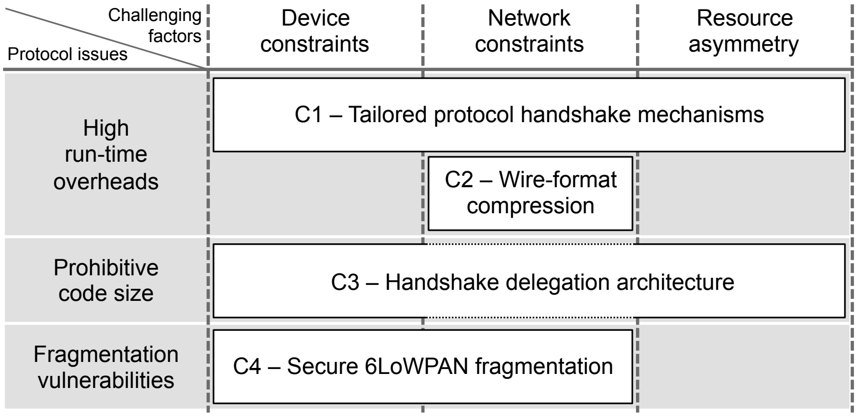

Challenging factors
Protocol issues
Device constraints
Network constraints
Resource asymmetry
High run-time overheads
Prohibitive code size
Fragmentation vulnerabilities
C1 – Tailored protocol handshake mechanisms
C2 – Wire-format compression
C3 – Handshake delegation architecture
C4 – Secure 6LoWPAN fragmentation

Figure 1.3 Mapping from our contributions to the main factors challenging the design adequacy of end-to-end IP security protocols and the resulting protocol efficiency and security issues.

fragmentation attacks and show the effectiveness of our defense mechanisms at moderate trade-offs. Importantly, 6LoWPAN packet fragmentation is not limited to end-to-end IP security protocols and equally applies to the wider scope of *large data transfers*, e.g., in case of firmware updates or incompressible sampling data [TH14]. Consequently, the above defense mechanisms also provide protection for other types of network traffic that are not specifically considered in the context of this thesis.

1.3 Interplay of our Contributions

Combined, the solutions devised as part of the four core contributions of this thesis comprehensively account for the efficiency and security issues that we identify in our protocol analyses. The contributions thereby address the main goal of this thesis from different perspectives. Hence, we now discuss the mapping from the four contributions to this main goal and highlight the relationships between them.

As depicted in Figure 1.3, the first contribution TAILORED PROTOCOL HANDSHAKE MECHANISMS ($C1$) addresses both aspects of the main goal, i.e., *security protocol efficiency* and *protocol security*, by accounting for the high run-time overheads of DTLS, HIP DEX, and Minimal IKEv2 in the context of networked embedded devices. In this, the flexible session resumption and adaptive retransmission mechanisms reduce the computation and transmission overheads that are caused by the protocol handshake, thus specifically accounting for the device and network constraints in the embedded domain. Moreover, the collaborative puzzle-based DoS protection mechanism mitigates attacks against the protocol handshake that exploit the resource asymmetry in the IoT. The second contribution MESSAGE WIRE-FORMAT COMPRESSION ($C2$) likewise concerns the identified run-time overheads. This contribution, however, focuses on improving the utilization of the available network resources. $C2$, therefore, contributes to the sub-goal *security protocol efficiency*.

The HANDSHAKE DELEGATION ARCHITECTURE ($C3$) leverages the resource asymmetry in the IoT to address the device constraints in the embedded domain by offloading the connection establishment handshake to the unconstrained delegation server. This allows to address the prohibitive code size of the considered end-to-end security protocols for memory-constrained embedded devices. Thus, $C3$ primarily contributes to the sub-goal *security protocol efficiency*. Finally, SECURE 6LOWPAN

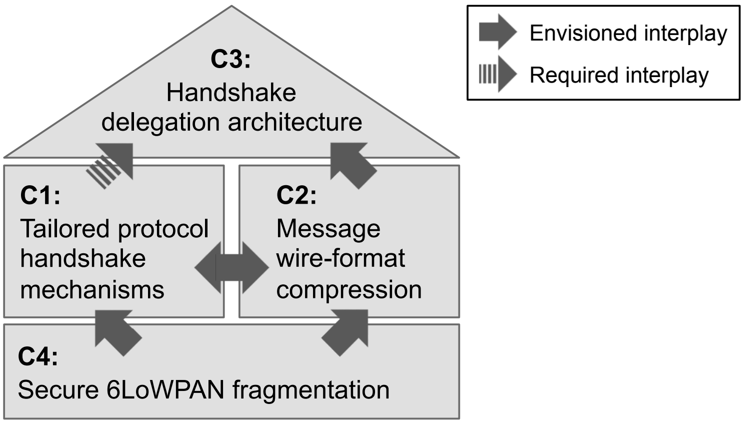

C3:
Handshake delegation architecture
C1:
Tailored protocol handshake mechanisms
C2:
Message wire-format compression
C4:
Secure 6LoWPAN fragmentation
Envisioned interplay
Required interplay

Figure 1.4 Interplay between our four core contributions $C1$ to $C4$ in this thesis. $C4$ provides the foundation for our remaining contributions. $C1$ and $C2$ complement each other. Moreover, $C3$ depends on one of our developed solutions from $C1$ and may additionally leverage $C2$.

FRAGMENTATION ($C4$) addresses the sub-goal *protocol security* by defending networked embedded devices against fragmentation attacks at the 6LoWPAN layer.

Regarding the relationships between the four contributions of this thesis (illustrated in Figure 1.4), contribution $C4$ builds the foundation for the remaining three contributions by securing 6LoWPAN as the underlying transport mechanism for DTLS, HIP DEX, and Minimal IKEv2 in the embedded domain. The complementary contributions $C1$ and $C2$ then provide for the efficient and secure operation of these protocols in the context of networked embedded devices. Contributions $C1$ and $C2$, however, assume these devices to be equipped with sufficient RAM and ROM resources for a comprehensive protocol implementation with support for public-key cryptography. Hence, to also cater to networked embedded devices with insufficient memory resources for such a comprehensive protocol implementation, contribution $C3$ additionally addresses the high memory requirements of the considered end-to-end IP security protocols. In doing so, contribution $C3$ builds on the flexible session resumption mechanism from contribution $C1$. Similarly, the message compression facilities from contribution $C2$ could also be employed in contribution $C3$ to further reduce the transmission overhead in the embedded domain. As such, the contributions in thesis effectively complement each other and, combined, significantly improve the efficiency and security of end-to-end security in the context of the IoT.

1.4 Genesis and Attribution of our Contributions

In this section, we outline the genesis and the attribution of the four core contributions of this thesis. Overall, these contributions were developed in collaboration with several students in the context of their Bachelor's, Master's, and Diploma theses at COMSYS and include the input of various co-authors concerning the respective publications. If not mentioned otherwise, the author of this thesis was responsible for the initial conceptual ideas, the detailed solution design, and the final publication.

The flexible session resumption extension and an early version of the collaborative puzzle-based DoS protection mechanism of contribution $C1$ were developed jointly with Jens Hiller in the context of his Bachelor's thesis [Hil12]. Jens Hiller implemented and evaluated the devised protocol extensions. Similarly, he realized the adaptive retransmission mechanism. The feedback of the co-authors allowed to improve the presentation of the devised protocols extensions in a scientific publication [HWZ$^+$13] and several standardization documents [HGS13, HHH13, MH14].

The abstract idea of contribution $C2$ was published in [HHW11]. Here, Tobias Heer contributed with valuable discussions about the adaption requirements of end-to-end security in the IP-based IoT. The author of this thesis further substantiated these initial design ideas in collaboration with Jens Hiller in the context of his Bachelor's thesis [Hil12]. In doing so, the author of this thesis identified the compression potentials with respect to the HIP DEX message wire-format. Afterwards, Jens Hiller implemented the devised Slimfit layer and evaluated the corresponding prototype. The results were published in [HHHW13]. In this context, Martin Henze contributed to the comparison of the Slimfit layer with various generic compression mechanisms.

Contribution $C3$ is the result of early work with Christian Röller in the context of his Diploma thesis [Röl12] and several discussions at the IETF based on [HRW12]. The author of this thesis further improved on these initial ideas in collaboration with Hossein Shafagh in the context of his Master's thesis [Sha13]. In this, Hossein Shafagh implemented the handshake delegation architecture according to author's design decisions and compared the resulting architecture with standard DTLS handshakes. The feedback of the co-authors contributed to the presentation of this work in scientific publications [HZS$^+$13, HSR$^+$14] and standardization documents [HGS13].

Finally, contribution $C4$ was published in [HHW$^+$13]. Here, Timo Boetcher and the author of this thesis jointly performed the security analysis of the 6LoWPAN fragmentation mechanism in the context of a Diploma thesis [Boe11]. The corresponding defense mechanisms then were partially implemented by Timo Boetcher. Moreover, Hossein Shafagh and Jens Hiller contributed to the implementation and the evaluation of the devised defense mechanisms as part of their work at COMSYS.

1.5 Thesis Outline

The remainder of this thesis is structured as follows. In Chapter 2, we lay the foundation of our work by introducing the basic network scenario and by describing the general network architecture as well as the protocol landscape of the IP-based IoT. Chapter 3 then motivates the need for network security in the IoT and introduces the cryptographic primitives and the IP security protocols that provide the basis for our contributions. Based on the provided protocol descriptions and an analytical protocol comparison, we derive the problem statement of this thesis. Chapters 4 through Chapter 6 then present the four core contributions. Because two of these contributions focus on run-time-related protocol issues, we cover these contributions in a single chapter. For each contribution, we analyze the problem space, describe the solution design, discuss the security considerations as well as related work, and present the evaluation results. We conclude this thesis in Chapter 7 by revisiting the key challenges regarding our contributions and by identifying future work.

2

The IP-based Internet of Things

Our main goal in this thesis is to contribute to the secure communication of networked embedded devices in the IP-based IoT. IP technology thereby enables the transparent interconnection of these devices with the existing IP infrastructure and affords interoperable communication across application and network domains. The special device and network characteristics in the embedded domain, however, render an unmodified adoption of IP technology in the IoT inefficient. The IETF, therefore, currently standardizes dedicated protocols and protocol adaptations for the IoT.

In this chapter, we lay the foundation of this thesis by introducing the basic network scenario and its enabling IP technology. Specifically, we first outline exemplary IoT application scenarios and common IoT traffic flows in Section 2.1. We describe the special device and network characteristics concerning the embedded domain of these scenarios in Section 2.2. Section 2.3 then presents the adapted IP network architecture for networked embedded devices and highlights the relationship of its comprising protocols to our contributions. Section 2.4 further details the 6LoWPAN adaptation layer for IPv6 as this new protocol layer is especially relevant within the context of this thesis. We conclude this chapter with a brief summary in Section 2.5.

2.1 Application Scenarios and Network Traffic Flows

The key idea behind the IP-based IoT is to transparently interconnect objects from the physical world with the existing IP infrastructure in order to provide an enhanced understanding and control over the physical environment and the objects residing in it. Application scenarios that are destined to benefit from such an interconnection range from personal health monitoring and home automation solutions to agricultural monitoring, vehicular telematics, and industrial control systems [KKV12]. Notably, communication in these scenarios often is envisioned to involve direct device-to-device interactions. For instance, a thermostat in a building automation scenario may directly interact with a radiator control unit to adjust the room temperature.

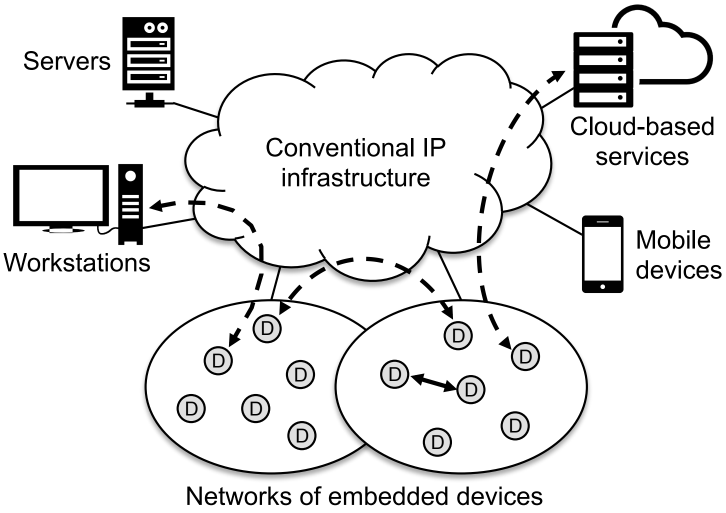


Figure 2.1 Abstract IoT network scenario consisting of networked embedded devices (D) and conventional communication end-points such as workstations and servers. Arrows indicate network traffic flows with a special focus on interactions that involve networked embedded devices. Dashed arrows represent traffic flows that involve conventional IP infrastructure.

Such device-to-device communication can also extend beyond the scope of a single network domain. This, for example, is the case if spacial (the building automation solution extends beyond the scope of a few adjacent rooms) or technical restrictions (the involved devices employ different radio technologies) need to be accounted for. As depicted in Figure 2.1, the interconnection of the different devices in such inter-domain scenarios then often involves conventional IP infrastructure.

Interactions in the IoT, however, are not limited to communication between networked embedded devices (see Figure 2.1). Direct end-to-end communication between networked embedded devices and conventional communication end-points such as workstations or mobile devices, for example, has the potential to ease the configuration, monitoring, and on-site maintenance of these embedded devices. Moreover, networked embedded devices may also report their sensed information to a central data collector such as a local server or a cloud-based service as depicted in Figure 2.1 (see rightmost arrow) [HHCW12]. With respect to industrial monitoring, the machinery in a production plant, for instance, may report critical equipment parameters such as temperature or vibrations to its manufacturer with the goal to extend periodic maintenance cycles and to trigger short-term maintenance operations in case of sensed abnormalities that are indicative of an impending equipment failure. Similarly, networked embedded devices may also tap into the vast number of information sources on the Internet to fulfill their tasks in the physical world. For example, an automated sprinkler system of an agricultural monitoring solution may schedule its watering cycles depending on an Internet-based weather forecast.

As highlighted by the examples given above, IP technology for networked embedded devices enables objects from the physical world to *seamlessly* interact with the existing IP infrastructure as *active network participants*. The special device and network characteristics in the embedded domain, however, necessitate adaptations to the employed IP protocols in order to guarantee their efficient operation in the IoT. We now first describe these special characteristics in more detail and then provide an overview of the standardization efforts at the IETF that aim at adapting IP technology to the device and network constraints in the embedded domain.

2.2 Special Characteristics in the Embedded Domain

Networked embedded devices often exhibit resource constraints that differentiate these devices from conventional IP-enabled hosts or services. To highlight this fact, we denote resource-constrained networked embedded devices with their short form: *constrained devices*[1]. Similarly, we call the networks formed by constrained devices – which then become constrained nodes in these networks – *constrained node networks*. We now continue with a description of the most important properties of constrained devices. Moreover, we introduce the network technologies that typically are used for the interconnection of constrained devices, focusing on wireless technologies.

2.2.1 Constrained Devices

With annual sales in the order of billions of units [Ins15], embedded devices build the basis for most of today's electronic consumer goods and industrial equipment. These embedded devices are typically designed for *small size and low power consumption*. Moreover, *low production costs* often play a vital role in the design of embedded devices: As these devices are produced on a massive scale, savings in the order of a few cents per device constitute considerable cost reductions with respect to the production of billions of units. As a result, the design of embedded devices commonly trades a small size, low costs, and high energy efficiency for *low clock frequencies* and *limited memory resources* [BEK14]. More precisely, embedded devices typically employ microcontrollers (MCUs) that operate at clock frequencies in the order of a few MHz and provide memory resources in the order of several kilobytes of RAM and several tens of kilobytes of ROM. These limited resources stand in stark contrast to the vast amount of hardware resources of conventional IP-enabled hosts or services. For example, even mobile devices such as smartphones nowadays are equipped with a few GBs of RAM, tens of GBs of persistent storage, and multi-core CPUs that provide computation power in the order of GHz. Hence, these conventional communication end-points vastly outperform embedded devices.

With continuing advances in low-power radio technology, embedded devices are increasingly supplemented with wireless radio modules [HC02]. This enables embedded devices to also consider information from other devices in their vicinity when performing their tasks in the physical world. To investigate challenges originating from the low-power wireless interconnection of networked embedded devices, several experimentation platforms were created over the past decade within the scope of WSN research. In this thesis, we utilize i) the TelosB [PSC05], ii) the WiSMote [wismote], and iii) the Zolertia Z1 [zolertia] platforms as instances of constrained devices for our prototyping and evaluation purposes. These platforms are built from off-the-shelf components that are also used in commercial embedded devices. We now briefly describe the most important properties of these hardware platforms to provide a more detailed impression of the device constraints in the IP-based IoT.

---

[1] We note that this terminology follows established terminology at the IETF [BEK14].

The *TelosB* platform is equipped with a 16-bit MSP430-based MCU that operates at 8 MHz and provides 10 KB of RAM and 48 KB of ROM. It is one of the most widely used platforms in WSN research. Both, the *WiSMote* and the *Zolertia Z1* platform, constitute revised variants of the TelosB platform. More precisely, they are based on newer generations of the 16-bit MSP430 MCU that operates at 16 MHz and supports 20-bit memory addressing. Thus, the WiSMote and Z1 platforms can address a larger memory region than pure 16-bit platforms such as the TelosB. This allows to equip the WiSMote platform with 16 KB of RAM and 128 to 256 KB of ROM. Similarly, the Zolertia Z1 platform has 8 kB of RAM and 92 kB of ROM.

We note that more powerful platforms such as the 32-bit System on a Chip (SoC) ARM Cortex-M3 have also become available on the embedded market. These platforms, however, do not yet achieve the same low production cost and high energy efficiency as their 8-bit or 16-bit counterparts [KKR+12]. Hence, these platforms currently are primarily intended for application scenarios with comparably high computation demands such as medical sensing scenarios. In contrast, our work focuses on application scenarios that do not necessitate the high processing power of these more powerful platforms. Still, it is worth noting that the contributions of this thesis may also find their application in the context of these more powerful platforms.

2.2.2 Wireless Technologies and Constrained Node Networks

Similar to embedded devices, their designated radio technologies are also designed for low-power operation and cost-efficient production of the corresponding radio modules. To this end, these technologies, e.g., employ low-bandwidth wireless links and simple radio components. Furthermore, radio duty cycling techniques are commonly employed in order to achieve an energy-efficient utilization of the transceiver by switching off the radio module for most of the time [MBC+01, PSC05]. As a result of these design traits, the radio links between constrained devices, however, often exhibit a *higher packet loss ratio* than observed in conventional IP networks [SDTL10].

To achieve interoperable communication within the embedded domain, a small number of radio standards specifies the physical and the data link layers for the wireless links between networked embedded devices. IEEE 802.15.4 [IEEE11b] is one of the most prominent standards. It forms the basis for several other standards including ZigBee [ZB12], ISA100.11a [ISA11], and WirelessHART [WH13]. IEEE 802.15.4 allows to transmit data with a rate of up to 250 kbit/s. Moreover, it defines a maximum frame size of 127 byte. Concerning upper layer payload data, these 127 byte are further reduced by a minimum of 9 byte and a maximum of 46 byte for header

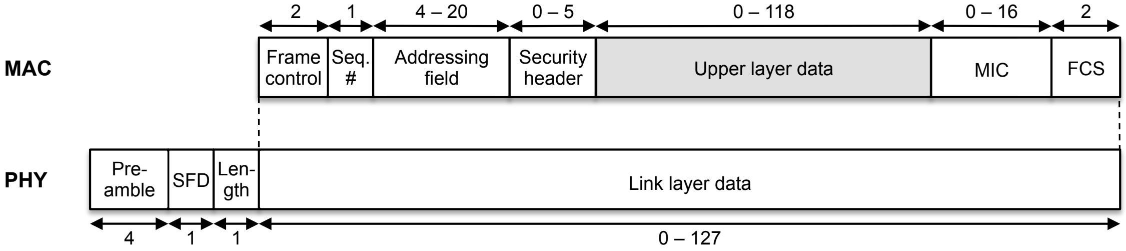
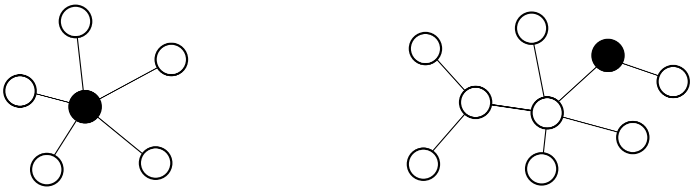


Figure 2.2 IEEE 802.15.4-2003 frame format at the physical and the link layer. The frame space available for upper layer protocol data is marked gray. Size indications are given in byte.



(a) Star topology **(b)** Mesh topology

Figure 2.3 Network topologies supported by the IEEE 802.15.4 radio standard. The node marked in black provides the uplink to an adjacent network for all other nodes.

information at the data link layer depending on the employed addressing and security modes (see Figure 2.2). Hence, in the worst case, no more than *81 byte of upper layer protocol data* can be transmitted in a single IEEE 802.15.4 frame [IEEE03].

We note that the IEEE 802.15.4 protocol revisions from 2006 [IEEE06] and later add additional information to the *security header*. As a result, the overall header overhead with these revisions further increases by up to 9 byte. Correspondingly, the worst case payload size decreases to 72 byte. Importantly, the CC2420 [cc2420] radio modules of the TelosB and Zolertia Z1 platforms only support the IEEE 802.15.4-2003 standard, whereas the CC2520 [cc2520] radio interface of the WiSMote platform supports IEEE 802.15.4-2006. Hence, our evaluation setups in this thesis exhibit different per-frame payload lengths depending on the employed hardware platform.

With respect to the supported network topologies, the networks formed by IEEE 802.15.4-enabled constrained devices can either take the shape of a star topology or of a *multi-hop mesh topology* (see Figure 2.3). One node in these topologies then commonly provides the interconnection with potential adjacent networks for all other nodes of the constrained node network. In contrast to the star topology, the multi-hop characteristics of the mesh topology also allow the constrained node network to span beyond the range of a single radio link. The intermediate nodes in this topology then, however, require additional routing information to forward packets between end-points. It is worth noting that we do not make specific assumptions about the underlying topology of the constrained node networks in this thesis.

Finally, a few alternatives to IEEE 802.15.4 exist with respect to the wireless interconnection of constrained devices. These include Bluetooth Low Energy [BT13] and cellular network technologies such as GPRS [BDHS$^+$11]. However, as the experimentation platforms employed in this thesis are equipped with IEEE 802.15.4 radio modules, we perform our research based on this wireless standard as one specific radio technology for constrained node networks. Still, we note that the contributions in Chapters 4 and 5 are independent from the employed link layer technology and, thus, also apply in application scenarios with alternative radio technologies.

2.3 Interconnecting Constrained Devices with IP

The above device and network characteristics render an unmodified adoption of IP technology in the IoT inefficient. The IETF, therefore, currently standardizes dedicated protocols and protocol adaptations for the IP-based IoT. We now describe the network architecture that affords the seamless integration of constrained devices

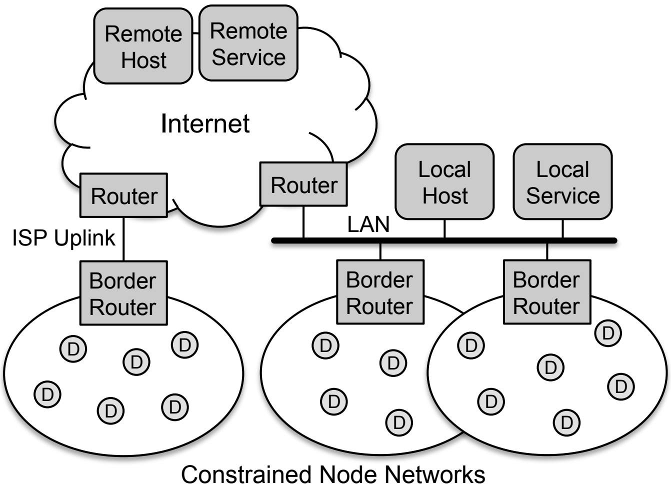


Figure 2.4 The network scenario in scope of our work. IP-enabled constrained devices (D) form constrained node networks. Border routers connect these networks, e.g., to a local area network or to the Internet. The connectivity affords communication between constrained devices from separate network domains and enables the interconnection with local/remote hosts or services.

with the existing IP infrastructure. This architecture denotes the *basic network scenario* of our work. Moreover, we introduce the adapted IP network stack for constrained devices that enables the efficient operation of IP technology in constrained node networks. Its comprising protocols denote the enabling foundation of our work.

2.3.1 Network Architecture of the IP-based Internet of Things

One of the core architectural principles of the Internet is to interconnect *independent network domains* by forming a *network of networks* [KR12]. Following this idea, IP-enabled constrained devices can be interconnected by forming independent constrained node networks and by attaching these networks to the existing IP infrastructure. As shown in Figure 2.4, this network attachment can, e.g., be a direct uplink to an Internet Service Provider (ISP) or to a Local Area Network (LAN).

The separation into independent network domains allows each network domain to operate its preferred link layer technology. Interconnecting *border routers* then translate between the different link layer technologies and leverage routing information at the IP layer for packet forwarding purposes. Thus, communication of constrained devices is no longer confined to the local wireless network, but may also *transparently* take place *across* network domains without the need for dedicated application-level gateways. In fact, this transparent communication may involve interactions with other constrained devices as well as with conventional *hosts or services* that are situated in adjacent LANs or that are reachable via the Internet. These hosts and services as well as the interconnecting routers typically rely on commodity, mains-powered hardware and do not exhibit the same resource limitations as constrained devices. Hence, they denote *comparably powerful network entities* in the IoT.

Regarding the network layer protocol that enables such transparent communication in the IoT, the IPv6 standard is commonly preferred over the Internet Protocol version 4 (IPv4) [HC08]. This is primarily due to the limited IPv4 address space

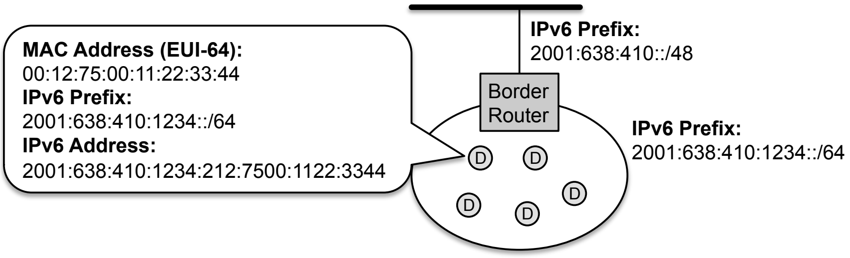


Figure 2.5 Stateless address autoconfiguration for constrained devices (D) with IPv6. The border router disseminates the IPv6 prefix inside the constrained node network. The constrained devices then complement this prefix to an IPv6 address, e.g., based on their MAC address.

and the improved auto-configuration mechanisms of the IPv6 standard [SB10]. The latter is especially important as constrained devices may follow an *unattended mode of operation* and often *do not provide user interfaces* other than their wireless links.

To enable forwarding of IP packets to or from a constrained device with IPv6, each constrained node network is assigned an IPv6 address prefix. This prefix must be valid and unique within the scope of the interconnecting uplink in order to provide connectivity with remote communication end-points. The management of this prefix is delegated to a border router. This router is located at the boundary of the constrained node network (see Figure 2.5). As one of its main tasks, the border router disseminates its IPv6 prefix inside the wireless network, e.g., via the IPv6 Neighbor Discovery (ND) mechanism [SCNB12]. Thus, all constrained devices belonging to the same constrained node network share a common IPv6 prefix, i.e., the first 64 bit of the IPv6 address. When learning this prefix, a constrained device complements the prefix to a complete IPv6 address by appending a 64 bit interface identifier that must be unique for a given constrained node network [MKHC07]. Hence, an external network packet sent to a constrained device can first be forwarded to the corresponding border router based on the IPv6 prefix of the destination address. From there, the packet is forwarded to the final destination inside the constrained node network based on network-internal routing information. This information must be provided by an additional routing protocol, e.g., the IPv6 Routing Protocol for Low-Power and Lossy Networks (RPL) [WTB$^+$12]. Similarly, a network packet sent to an end-point that is located in an external network domain is first forwarded to the border router since the IPv6 destination address prefix is not used inside the constrained node network. The border router then forwards the received packet towards its final destination based on the border router's local routing information.

To summarize, the IP network architecture enables a constrained device to uniquely address another constrained device, host, or service independent from the network that this communication end-point belongs to. Similarly, the communication end-point can address the device once its IP address is known. This addressability enables *transparent end-to-end communication across network domains* in the IP-based IoT.

2.3.2 The Adapted IP Network Stack for Constrained Devices

The IP network architecture affords transparent end-to-end communication in the IoT. Its supporting protocols, however, were not developed with the special device

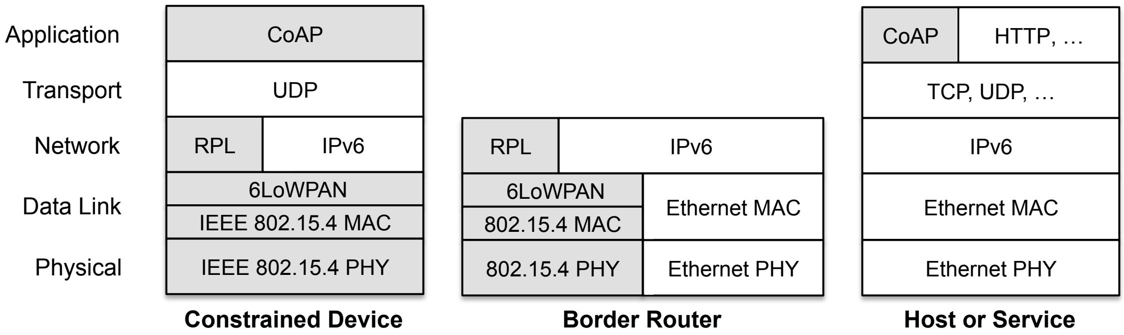


Figure 2.6 Comparison of the IP network stacks for constrained devices, border routers, and conventional hosts or services. Protocols that are specifically designed for the IP-based IoT are marked in gray. Notably, the IPv6 adaptation layer 6LoWPAN and the RPL routing protocol are only deployed on constrained devices and border routers, whereas the application protocol CoAP should also be supported by conventional hosts and services.

and network limitations of constrained node networks in mind (see Section 2.2). As a result, these protocols make assumptions about properties of the underlying link layer that often do not hold in the context of constrained node networks. Several standardization efforts at the IETF, therefore, aim at adapting the IP network stack to the special device and network characteristics in the embedded domain. The main goal of these protocol adaptations is to reduce the header overhead, the processing requirements, and consequently the power consumption for constrained devices.

For IEEE 802.15.4 networks, these standardization efforts resulted in a network stack that primarily differs from conventional network stacks with respect to the following three protocols: i) the RPL routing protocol, ii) the 6LoWPAN adaptation layer for IPv6 [MKHC07], and iii) the Constrained Application Protocol (CoAP) [SHB14].

As depicted in Figure 2.6, the deployment of RPL and 6LoWPAN is limited to the constrained node networks. Consequently, the use of these protocols is transparent to the communication end-points that are located outside a constrained node network. In contrast, CoAP is intended to be used in an end-to-end fashion and, thus, must be supported by all communication partners of constrained devices. We now provide a brief overview of these protocols and highlight their relationship to our contributions in this thesis. We then describe the 6LoWPAN adaptation layer in more detail as this newly introduced protocol layer is highly relevant in the context of this thesis.

The RPL routing protocol for multi-hop mesh topologies

Multi-hop mesh topologies require routing information at each intermediate hop in the constrained node network in order to afford packet forwarding towards the final destination. To establish this routing information, the IP network stack for constrained devices includes the RPL routing protocol. RPL builds a tree-based routing topology on top of the available wireless links. This routing tree is typically rooted at the border router. To build this tree, RPL facilitates a distance vector routing approach [TdOV10]. We note that alternatives to RPL may also be used for routing purposes in constrained node networks as the routing protocol is not exposed to adjacent networks and therefore is network-specific. In our work, we assume the deployment of a routing approach that affords multi-hop packet forwarding in constrained node networks. Still, we do not rely on a specific routing protocol.

IPv6 protocol adaptations in the context of the 6LoWPAN layer

As shown in Figure 2.6, the 6LoWPAN layer is located between the data link layer and the IPv6 layer. Its main task is to adapt the IPv6 packet format for transmission over IEEE 802.15.4 links. To this end, the 6LoWPAN layer provides a *header compression mechanism* that trades an increased computation overhead for a reduced transmission overhead. This specific trade-off is highly advantageous in constrained node networks as the costs of transmissions largely outweigh the cost of computations with respect to their energy expenditure [PK00]. In addition, compression and decompression typically only have to be performed by the communication end-points, whereas transmissions often involve several constrained devices in multi-hop topologies. Hence, packet compression has the potential to not only improve the lifetime of the source and the destination of a packet flow, but also to increase the overall lifetime of an entire constrained node network. Following the same goal, we will present the Slimfit compression layer as *one of the contributions* of this thesis.

The 6LoWPAN standard additionally introduces a *fragmentation mechanism* for IPv6 packets. This mechanism affords the standard-conform use of the IPv6 protocol in constrained node networks and enables the transmission of IPv6 packets that exceed the maximum frame size of IEEE 802.15.4. As *a second contribution*, we will show that this mechanism is vulnerable to potential DoS attacks. For further details about the 6LoWPAN fragmentation mechanism, we refer to Section 2.4.2.

Further modifications of the IPv6 standard in the context of the 6LoWPAN adaptation layer concern the IPv6 Neighbor Discovery mechanism [SCNB12]. These modifications include the limitation of the IPv6 Neighbor Discovery mechanism to host-initiated protocol interactions as well as the elimination of multicast traffic. This allows to reduce unsolicited network transmissions inside constrained node networks and enables the use of radio duty cycling techniques to decrease the energy expenditure on constrained devices. Moreover, the adapted IPv6 Neighbor Discovery mechanism also includes the possibility to disseminate 6LoWPAN header compression contexts. As a result, the 6LoWPAN layer can apply additional context-based header compression mechanisms to IPv6 packets. We refer to Section 2.4.1 for further information about these context-based header compression mechanisms.

The CoAP protocol for web services in the IoT

For end-point interactions at the application layer, the adapted IP network stack includes the *CoAP protocol*. CoAP is specifically designed to realize web services in the field of machine-to-machine (M2M) communication [She10]. As such, it is centered around the same RESTful client/server architecture as the Hypertext Transfer Protocol (HTTP). Specifically, CoAP provides a subset of the request methods offered by HTTP, i.e., GET, PUT, POST, and DELETE, and is based on a Uniform Resource Identifier (URI) addressing scheme. However, in contrast to HTTP, CoAP uses the User Datagram Protocol (UDP) as its underlying transport protocol in order to avoid the complexities of a reliable transport protocol such as the Transmission Control Protocol (TCP). Moreover, it employs a compact header format to reduce the transmission overhead in the context of constrained node networks. CoAP additionally extends the synchronous pull interaction model of HTTP with an asynchronous subscription-based model [Har14b]. As a result, continuous polling is no longer required to detect changes to a REST resource of a constrained device.

In this thesis, we do not rely on a specific application layer protocol. Instead, we focus on *general-purpose* security solutions that allow to protect a wide range of application protocols and application data. Still, the CoAP specification has key implications on our work as it mandates an equal level of security as currently recommended for regular Internet-based communication. As a *third and fourth contribution*, we will introduce lightweight protocol extensions and a security architecture that allow to realize this level of security in the context of constrained devices.

2.4 The 6LoWPAN Adaptation Layer for IPv6

We now provide a detailed description of the 6LoWPAN header compression and fragmentation mechanisms. Regarding the 6LoWPAN header compression mechanism, it is our intention to familiarize the reader with related efforts to the Slimfit compression layer that we will present in Chapter 4. The discussion of the 6LoWPAN fragmentation mechanism, in turn, aims at providing a basic understanding of the protocol issues below the network layer that we will address in Chapter 6.

2.4.1 6LoWPAN Header Compression Facilities

A central goal of the 6LoWPAN standard is to reduce the packet overhead of the IPv6 and UDP headers. This overhead reduction is highly desirable for constrained node networks as the IEEE 802.15.4 standard only allows for IPv6 packet sizes of up to 81 byte for network setups involving maximum-size link layer headers (assuming the IEEE 802.15.4-2003 standard). An IPv6 header in its shortest form, i.e., excluding IPv6 extension headers, however, already requires 40 byte of this payload space. An 8 byte UDP header further reduces this payload space, thus leading to a maximum of only 33 byte or about 26 % of an IEEE 802.15.4 frame for the actual application data. As a result of these header overheads, each link layer frame would primarily convey protocol header information instead of the intended application data.

To improve this payload ratio, the 6LoWPAN standard defines a number of header compression mechanisms for the IPv6 and UDP protocols [MKHC07]. These mechanisms operate on a *hop-by-hop basis* and follow a *per packet approach*. This design decision denotes a trade-off between an optimal compression ratio and low requirements regarding the RAM and ROM resources needed on constrained devices. In fact, flow-based compression mechanisms such as RObust Header Compression (ROHC) [SPJ10] commonly achieve better compression results than purely packet-based approaches as these mechanisms can additionally exploit redundancies across a continuous packet flow. However, these approaches often incur extensive book-keeping and synchronization overheads for the recovery mechanisms that are needed to handle packet loss in a continuous packet flow. Thus, packet-based approaches were considered preferable in constrained node networks when standardizing the 6LoWPAN layer at the IETF [SB10]. We note that we follow this assessment in the design of the Slimfit compression layer for the HIP DEX protocol.

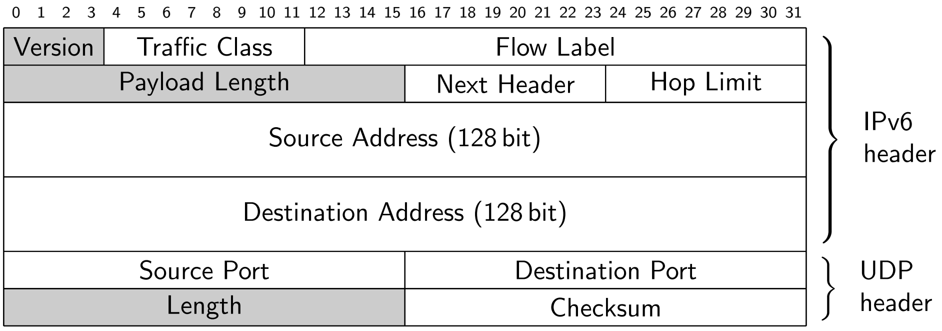


Figure 2.7 IPv6 and UDP header structure. Header fields that always are compressible are marked in gray. All other fields are only compressible depending on upper or lower layer header information or require the header field to contain specific values for compression purposes.

The 6LoWPAN standard initially defined two *stateless* compression mechanisms, i.e., LOWPAN_HC1 for IPv6 and LOWPAN_HC2 for UDP. Both mechanisms compress typically unused or redundant header fields. As depicted in Figure 2.7, such compressible header fields, e.g., include the IP version field, which is expected to commonly refer to version 6 in the context of the 6LoWPAN. Similarly, the IPv6 and UDP length information can typically be derived from link layer header information and thus can be omitted for transmission inside a constrained node network.

The LOWPAN_HC1 mechanism, however, is limited to the compression of link-local IPv6 addresses. This is because link-local IPv6 addresses commonly are based on the well-defined IPv6 prefix *fe80::/64* and the link layer address of a constrained device, which also is contained in the link layer header for link-local communication [MKHC07]. In contrast, the compression of network-external, globally routable IPv6 addresses would require additional shared knowledge of the IPv6 prefix and the complementing interface identifier as this information is not readily available inside a constrained node network. To address this short-coming, a *context-based* compression mechanism called LOWPAN_IPHC recently replaced the stateless LOWPAN_HC1 approach [HT11]. This compression mechanism employs *compression contexts* that are configured on a network-wide basis and that identify omissible IPv6 address information, e.g., the prefix of a globally routable IPv6 address. The dissemination of these context inside a constrained node network can, e.g., be achieved via the adapted IPv6 Neighbor Discovery mechanism for constrained devices [SCNB12].

Notably, the LOWPAN_IPHC mechanism does not leverage other redundancies in a packet flow beyond IPv6 addresses that behave in a predictable manner, e.g., sequence numbers. As a result, it does not incur the extensive book-keeping and synchronization overheads of flow-based compression mechanisms. For the same reason, the Slimfit compression layer, which we will present in Chapter 4, employs compression contexts to afford evolvability of the devised compression scheme.

When extending the 6LoWPAN header compression facilities with LOWPAN_IPHC, a new upper layer header compression mechanism called LOWPAN_NHC also was defined as a replacement of LOWPAN_HC2. Compared to LOWPAN_HC2, LOWPAN_NHC additionally affords the compression of IPv6 extension headers as well as stacks of compressed IPv6 extension and transport protocol headers. Moreover, LOWPAN_NHC allows to indicate the compression mechanism employed in a 6LoWPAN packet via a dedicated Next Header Compression (NHC) ID. We leverage this NHC ID field to signal the use of our Slimfit compression layer.

Achievable 6LoWPAN compression ratios

We now provide a brief overview of the compression ratios that can be achieved with the LOWPAN_IPHC and LOWPAN_NHC mechanisms. These ratios are important with respect to the evaluation of our contributions in Chapters 4 to 6.

The compression ratios with LOWPAN_IPHC and LOWPAN_NHC strongly depend on the header information at the data link layer, the transport layer, and the availability of additional 6LoWPAN compression contexts. In the best case, LOWPAN_IPHC reduces the IPv6 header size to 2 byte with link-local communication [HT11]. Similarly, LOWPAN_NHC allows to compress UDP headers to 2 byte. Hence, 77 byte of payload remain for application data on a per-frame basis, i.e., assuming IEEE 802.15.4-2003. In the worst case, however, the IPv6 header can only be compressed from 40 to 39.5 byte and the UDP header from 8 to 7 byte. Thus, a single 6LoWPAN packet may still be limited to 34.5 byte of application data.

As the actual compression ratio is network- and scenario-specific, we consider a conservative compression ratio for our evaluation that lies in between these two extremes. Specifically, we assume the common case that the Traffic Class, the Flow Label, and the Next Header fields in the IPv6 header as well as the port information and the checksum field in the UDP header can be compressed (see Figure 2.7). Moreover, we assume that no end-point-specific 6LoWPAN compression contexts are available that would allow to omit network-external IP addresses in the IPv6 header via the LOWPAN_IPHC mechanism. As a result, IEEE 802.15.4 frames in our evaluation setups in Chapters 4 and 6 can carry a maximum of 42 byte (i.e., with IEEE 802.15.4-2003). Conversely, our evaluation setup in Chapter 5 has a maximum per-frame payload size that is further reduced by 9 byte to a total of 33 byte since this setup employs IEEE 802.15.4-2006 inside the constrained node network.

2.4.2 6LoWPAN Packet Fragmentation

The IPv6 standard mandates a minimum Maximum Transmission Unit (MTU) of 1280 byte from the underlying data link layers that carry IPv6 packets [DH98]. As a result, IPv6-compliant communication end-points only fragment packets that exceed this well-defined threshold. However, even with maximum header compression, the available frame size of link layer technologies such as IEEE 802.15.4 does not suffice to transmit IPv6 packets of 1280 byte in an unfragmented manner. For such situations, the IPv6 standard specifies that link-specific fragmentation and reassembly mechanisms must be provided below the IPv6 layer. The 6LoWPAN standard defines these mechanisms for IEEE 802.15.4 and similarly constrained data link layers with the goal to split exceedingly large IPv6 packets into frame-sized fragments.

The 6LoWPAN fragmentation mechanism operates on readily compressed IPv6 packets. Hence, the 6LoWPAN layer first applies its header compression mechanisms to the original IPv6 packet and then performs the necessary packet fragmentation operations. The 6LoWPAN fragmentation mechanism thereby first checks if the compressed IPv6 packet exceeds the available payload size of the underlying link layer. In case of an exceeding size, it treats the entire packet as a single data field and iteratively segments this field into frame-sized 6LoWPAN fragments. In doing so, the fragmentation mechanism prefixes each fragment with a fixed-size

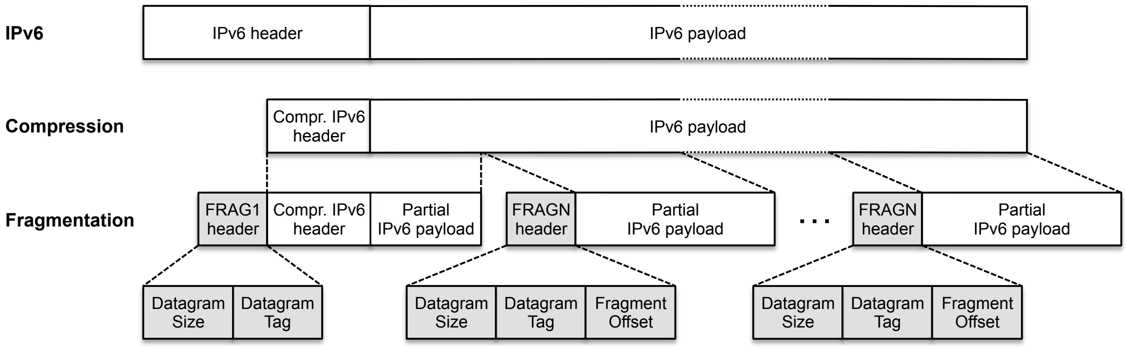


Figure 2.8 The packet structure before (top) and after (bottom) after the 6LoWPAN header compression and packet fragmentation mechanisms were applied to an IPv6 packet. The additional 6LoWPAN fragmentation headers are marked in gray. Notably, only the first 6LoWPAN fragment (i.e., the FRAG1) contains routable IPv6 header information.

6LoWPAN fragmentation header (see Figure 2.8). The fragmentation header of the first 6LoWPAN fragment, called *FRAG1*, thereby contains the *datagram size* of the entire uncompressed IPv6 packet and a *datagram tag*, which is unique per sender and fragmented IPv6 packet, for packet identification purposes. The remaining 6LoWPAN fragments, called *FRAGNs*, additionally include a *datagram offset* that indicates the position of the fragment payload in the uncompressed IPv6 packet.

Upon reception at a reassembling node, the information contained in the 6LoWPAN fragmentation header affords an efficient *in-place reassembly* of the fragmented IPv6 packet. More precisely, the conveyed datagram size enables a receiving node to reserve the necessary buffer space for the reassembly and decompression of the original IPv6 packet. The order, in which 6LoWPAN fragments are received, thereby does not impact the reassembly procedure as all fragments carry size information about the original IPv6 packet and, thus, afford the reservation of the necessary buffer space. Moreover, the fragment offset contained in the FRAGN header enables a reassembling node to immediately store the payload of the received 6LoWPAN fragments at the correct position in the reserved reassembly buffer. To verify that a newly received 6LoWPAN fragment indeed belongs to the same IPv6 packet as the ones already occupying the reassembly buffer, the reassembling node matches the datagram tag, the datagram size, and the link layer source and destination addresses of the received 6LoWPAN fragment to the partially reassembled IPv6 packet in the reassembly buffer. These header fields uniquely identify a fragmented IPv6 packet.

Concerning the reassembly procedure, the 6LoWPAN standard recommends that *overlapping fragments* in the reassembly buffer should cause this buffer to be flushed. The newly received fragment then may be used to begin the reassembly of a new fragmented packet. Moreover, the standard mandates a *reassembly timeout* of up to 60 seconds to free the reassembly buffer in case of an incomplete reception of a fragmented IPv6 packet, e.g., due to fragment loss. As we show in Chapter 6, an adversary can exploit these design decisions to block the reassembly of fragmented packets inside a constrained node network. This is especially critical as end-to-end IP security protocols typically require the transmission of packets that exceed the available frame size of IEEE 802.15.4 links during their protocol handshakes. Consequently, the identified 6LoWPAN fragmentation attacks enable an adversary to *block the establishment of secure end-to-end communication* in the IP-based IoT.

2.5 Summary

IoT network scenarios commonly involve the interconnection of constrained devices, hosts, and services. IP technology for constrained devices realizes this interconnection in an end-to-end manner and transparently across application and network domains. However, while largely enabled by the same network architecture as employed in conventional network scenarios, the special device and network characteristics of constrained node networks necessitate the adaptation of existing IP protocols. Such protocol adaptations must specifically consider the following three aspects:

Device constraints: Constrained devices are equipped with only a few MHz of computational power, several kilobytes of RAM, and several tens of kilobytes of ROM. Moreover, they may be battery-powered or employ energy harvesting techniques. Thus, constrained devices potentially only have a very limited energy budget. In addition, these devices often follow unattended modes of operation and do not provide user interfaces other than their radio modules.

Network constraints: Network transmissions within constrained node networks typically involve low-power, short-range wireless links. These links exhibit severe bandwidth and packet size limitations as well as packet loss rates that exceed these of conventional link layer technologies. For example, IEEE 802.15.4 links are limited to a maximum frame size of 127 byte at the link layer. As a result, header information and application data that extend beyond this frame size must be fragmented for transmission inside a constrained node network.

Resource asymmetry: Network interactions in the IoT not only involve communication between constrained devices but also affords end-to-end communication with conventional hosts or services. In contrast to constrained devices, these conventional communication end-points are equipped with comparably powerful hardware including multi-core CPUs with computational power in the order of GHz and memory resources in the order of several GBs of RAM and several hundred GBs of persistent storage. Moreover, these conventional end-points are attached to their networks via regular wired or wireless links. These links, however, typically do not exhibit the same bandwidth and packet size limitations as the wireless links employed in constrained node networks.

As we will discuss in the next chapter, these three aspects lead to several protocol efficiency and security issues in the context of standard end-to-end IP security solutions for the IoT. The detailed analysis of these protocol issues and the development of lightweight protocol extensions and a security architecture that resolve these design-level issues constitute the main contribution of this thesis.

3

Network Security in the IP-based IoT and Problem Statement

Global addressability in the IP-based IoT affords constrained devices, hosts, and services from independent network domains to interact with each other in an end-to-end manner. The achieved IP connectivity, however, also enables *malicious network entities* to communicate with these IP-enabled constrained devices. This is especially true for Internet-based interactions because such communication scenarios commonly lack global supervision and control of the interconnected communication end-points. As a result, constrained devices in the IoT are prone to be subject to a variety of network attacks. Hence, to protect constrained devices against such network attacks, authentication, authorization, and secure communication are vital preconditions for the secure interconnection of constrained devices in the IoT.

In this chapter, we motivate the need for end-to-end security and introduce the cryptographic mechanisms and IP security protocols that provide the basis of our work. To this end, Section 3.1 describes the underlying attacker model that we assume throughout the course of this thesis and outlines different types of network attacks that an adversary can mount with respect to this attacker model. We then present how cryptography can assist in mitigating these attacks in Section 3.2. A single cryptographic primitive, however, often does not suffice to comprehensively defend constrained devices against all types of network attacks. Security protocols, therefore, typically combine multiple cryptographic primitives and incorporate additional defense mechanisms in their protocol design. In Section 3.3, we first provide a brief overview of IP-based network security protocols for the IoT. We then describe the most prominent general-purpose end-to-end IP security protocol adaptions that currently are devised at the IETF, i.e., DTLS, HIP DEX, and Minimal IKEv2. These protocol adaptations denote the starting point of our work in this thesis. Based on these protocol descriptions and an analytical protocol comparison, we conclude this chapter with the definition of the problem statement for this thesis in Section 3.4.

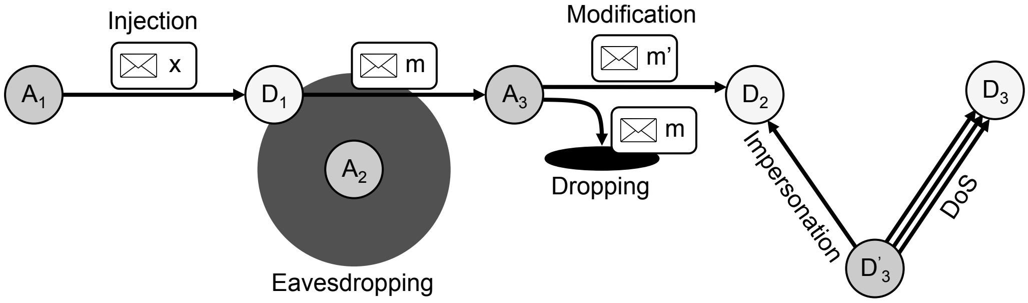


Figure 3.1 Overview of the network attacks that an adversary can mount in the assumed attacker model. The dark gray circle indicates the radio range of adversary A_2. A_2 can eavesdrop on network traffic from D_1. A_1, A_3, and D'_3 can mount injection, eavesdropping, dropping, modification, impersonation, and DoS attacks against the constrained devices D_j.

3.1 Network Threats in the IP-based IoT

In this section, we describe the Internet Threat Model [RK03] and outline high-level network attacks that an adversary can mount in this attacker model. We also discuss the relevance of these attacks in the context of the IoT and highlight their relationships to our contributions in this thesis. While not comprehensive, the provided overview contains the most prominent attacks for end-to-end security in the IoT. We note that the following descriptions are based on [RK03, TK05].

3.1.1 The Internet Threat Model

The Internet Threat Model describes the capabilities of an adversary when targeting communication end-points in Internet-based network scenarios. The model assumes that the target end-point itself is not compromised. The adversary, however, is given full control over the network infrastructure, which the target end-point uses for communication purposes. This implies that the adversary can inspect network packets at any layer of the network stack as well as drop, modify, and inject forged network packets. Consequently, the Internet Threat Model can be summarized as the *network infrastructure itself being the adversary*. Still, it is worth noting that the adversary is not granted indefinite computation or storage resources. Hence, the adversary is unable to break cryptographic mechanisms with brute force that are put in place to prevent potential network attacks. As depicted in Figure 3.1, the granted capabilities enable an adversary to mount several types of network attacks. We now briefly introduce these network attacks in the following sections.

3.1.2 Eavesdropping and Traffic Analysis

During an eavesdropping attack, an adversary overhears network packets that are transmitted between legitimately communicating end-points (see Figure 3.1). This enables the adversary to retrieve information that may not be intended for disclosure to a third party. In case of personal health monitoring, for instance, such information may consist of privacy-sensitive data from a networked heart-rate monitor. In addition to such immediate attacks on the transmitted information itself, eavesdropping often also denotes a first step towards further attacks [RKA09]. For example, as we will show in Chapter 6, eavesdropping is a prerequisite to mount an attack against the 6LoWPAN fragmentation mechanism, i.e., the fragment duplication attack.

Eavesdropping is a *passive* attack that does not require the adversary to be able to communicate in a specific network. The difficulty of mounting an eavesdropping attack strongly depends on the type of network. For wired, switched networks, eavesdropping is hard to accomplish as the adversary first has to subvert the network infrastructure to be placed on the packet forwarding path, i.e., *on-path*. In contrast, broadcast media, e.g., wireless channels as typically employed in constrained node networks, provide a shared communication platform. Hence, in these networks, an adversary can overhear legitimate communication while located besides the forwarding path, i.e., *off-path*, but within radio range of the target device (see Figure 3.1).

3.1.3 Packet Injection and Modification Attacks

Packet injection and modification attacks aim at interfering with legitimate communication or at disguising the identity of an adversary. In a packet injection attack, the adversary generates maliciously crafted network packets and injects these packets into the interconnecting network substrate. Such malicious packets may, e.g., contain a forged source address to hide the adversary's true point of network attachment. This falsifying of the source address is commonly referred to as *spoofing*.

Conversely, the adversary directly operates on legitimate network packets when mounting a packet modification attack. Hence, the adversary may delay, drop, or reorder packet transmissions as well as modify the original packet content. In the context of the IoT, an adversary can, e.g., exploit packet injection and modification attacks to maliciously inject new or modify legitimate control commands to achieve harmful actuation in the physical world [LRA09]. Moreover, we will show in Chapter 6 that an adversary can mount a 6LoWPAN fragmentation attack by overhearing fragmented IPv6 packets and sending a spoofed 6LoWPAN fragment in response.

Similarly important within the scope of this thesis, an adversary may also try to adversely influence the selection of the security mechanisms and parameters, e.g., during the connection establishment phase of end-to-end IP security protocols. In such *downgrading attacks*, the adversary eavesdrops on the connection establishment handshake and modifies the content of the corresponding handshake packets. In case of an insecure design of the negotiation mechanisms, the adversary then can enforce the selection of a weaker security mechanism than mutually desired without the end-points noticing. A key requirement for the security mechanisms, which we will present in Chapters 4, 5, and 6, is not to be affected by such downgrading attacks.

Packet injection and modification both denote *active* network attacks. Packet injection, thereby, does not necessitate eavesdropping capabilities from the adversary. However, without eavesdropping, the adversary is typically restricted to injecting forged packets *blindly*. Thus, the adversary may need to guess certain state information, e.g. sequence numbers, at the target device when trying to interfere with legitimate communication. In contrast, an eavesdropping adversary does not require guessing and can directly react to overheard packets. Importantly, packet modification attacks typically require an *on-path* adversary who performs the desired action on traversing packets before forwarding them towards the target device.

3.1.4 Impersonation Attacks and Access Violations

Most networked systems are not intended to be equally accessible by all communication end-points. This is also true for constrained devices. While resources such as interfaces to uncritical sensor information may be publicly accessible, resources serving sensitive information or exposing actuation and configuration interfaces will commonly be restricted to a confined set of communication end-points. An adversary may try to access such restricted resources by *impersonating* a privileged end-point such as a device administrator [RL09]. To this end, the adversary may, e.g., spoof the IP source address if access restrictions are solely based on IP Access Control Lists (ACLs). Such *access violations* can enable the adversary to retrieve sensitive information or may even allow to take over the constrained device by overwriting its firmware via compromised configuration interfaces. To prevent impersonation attacks, the security protocols in focus of this thesis employ authentication mechanisms that cryptographically identify the communicating end-points. Moreover, the handshake delegation architecture, which we will present in Chapter 5, allows to restrict resource access based on cryptographically assured end-point identities.

Notably, *replay attacks* may enable an adversary to subvert security mechanisms put in place to prevent impersonation attacks. To this end, the adversary first eavesdrops and records packets transmitted between two legitimately communicating end-point. The adversary then replays these packets – potentially delayed, re-ordered, in parts, or from a different network location – to one of the legitimate end-points thereby impersonating the other end-point. Another important requirement for our security mechanisms in Chapters 4, 5, and 6, therefore, is to also consider such subsequent attacks that may follow upon secure legitimate communication.

3.1.5 Denial of Service Attacks

The goal of DoS attacks is to exhaust the available system resources of a target device in order for this device to be unable to offer its services to other end-points. In doing so, an adversary typically targets scarce resources of a target device, e.g., CPU cycles, and exploits *resource asymmetries* in the design of the employed protocol. Even if the resources of the target device are similar to these of a single adversary and if the employed protocol does not exhibit any significant resource asymmetries, the adversary may still be able to artificially generate resource asymmetries that are required to mount a DoS attack. To this end, the adversary orchestrates a large number of compromised communication end-points to simultaneously interact with the target device. Such Distributed Denial-of-Service (DDoS) attacks are typically used in large-scale network attacks that target *high-value* Internet services.

An adversary can mount a DoS attack against any device that accepts inbound network connections. This property is often true in the IoT, especially in case of M2M communication scenarios. Moreover, the inherent resource asymmetry between a *single* adversary, who is, e.g., equipped with commodity workstation hardware, and a resource-constrained device enables this adversary to mount a DoS attack without the need to resort to DDoS attacks [RNL11]. This fact is further aggravated in the context of cryptographic protocol operation as these operations often incur a considerable computation overhead on constrained devices (see Section 3.2.3).

Threat **Crypto**	Eaves-dropping	Paket Injection	Modifi-cation	Imperso-nation	Access Violation	DoS
Authentication		x		x		
Confidentiality	x					
Data Integrity			x			
Non-repudiation						

Table 3.1 Overview of the fundamental objectives of cryptography and the presented network attacks. Non-repudiation beyond authentication is out-of-scope of this thesis. Notably, cryptography alone does not suffice to rigorously protect against network attacks.

It is important to note that DoS attacks in the IoT are not limited to the scarce computation resources of a constrained device. In fact, an adversary may also aim at exhausting its limited memory resources. To this end, the adversary, e.g., opens several network connections to the target device and exploits the corresponding protocol state. In addition, an adversary may also try to prevent an energy-constrained device from sleeping, thus draining its scarce energy resources and potentially limiting the overall lifetime of the entire constrained node network. In Chapter 4, we will introduce a DoS protection mechanism for end-to-end IP security protocols that we specifically design with the resource asymmetry in the IoT in mind. Moreover, we require the security mechanisms, which we will present in Chapters 4, 5, and 6, not to open new attack vectors that an adversary can exploit in a DoS attack.

3.2 Cryptography and Constrained Devices

Cryptography provides an algorithmic approach to protect communication end-points against the network attacks discussed in the previous section. We now first introduce the fundamental objectives of cryptography and highlight their relevance for the presented network attacks. We then continue with a brief overview of the main cryptographic primitives realizing these objectives. We thereby focus on the presentation of the basic ideas of the considered cryptographic primitives and discuss recent considerations regarding their use in the context of constrained devices.

3.2.1 Fundamental Objectives of Cryptography

Cryptographic primitives are designed to fulfill at least one of the following fundamental objectives of cryptography: i) authentication, ii) confidentiality, iii) data integrity, and iv) non-repudiation [DK07]. As shown in Table 3.1, each of these objectives mitigates one or more of network attacks that we presented in the previous section. We now discuss the individual objectives in more detail [DK07]:

Authentication: Authentication typically is further differentiated into peer authentication and data origin authentication. With *peer authentication*, the main goal is to establish the identity of the remote communication end-point when initiating network interactions. Thus, peer authentication protects against *packet injection and impersonation attacks* if also subsequent network transmissions are ensured to originate from the same authenticated end-point. Such correlation of subsequently received network packets to a previously established identity is also known as *data origin authentication.* Important in the scope of the IP-based IoT, global addressability necessitates peer and data origin authentication to restrict network communication to unambiguously identifiable end-points. One of the main contributions in Chapters 4 and 5 is the design of protocol mechanisms and a delegation architecture that afford secure and efficient authentication in the context of constrained node networks.

Confidentiality: The key objective of confidentiality is to conceal transmitted information from network participants that are not the intended recipients. Thus, confidentiality is of particular importance in the IoT when network packets traverse untrusted network infrastructure or are transmitted over broadcast media with untrusted network participants. Confidentiality then prevents adversaries from *eavesdropping* on privacy-sensitive or otherwise critical information that is exchanged between constrained devices, hosts, and services.

Data Integrity: The underlying purpose of data integrity protection is to allow the recipient of network packets to verify that the content of these packets was received without *modification* on the communication path. Similar to confidentiality, data integrity, therefore, is crucial when constrained devices transmit sensitive information in untrusted network environments. We note that confidentiality and data integrity are both commonly provided by the end-to-end IP security protocols that denote the main focus of this thesis.

Non-repudiation: With non-repudiation, a communication end-point should not be able to falsely deny its participation in network communication towards a third party. Non-repudiation is especially important in the context of network accounting. Reliable non-repudiation mechanisms, however, often require substantial infrastructure support, e.g., for secure remote logging [RK03]. In this thesis, we consider non-repudiation beyond authentication to be out-of-scope.

We now proceed with a brief overview of the most important cryptographic primitives in this thesis and highlight their special considerations for constrained devices.

3.2.2 Symmetric-Key Cryptography

In the context of network security, symmetric-key cryptography is typically used to provide confidentiality and integrity protection of the transmitted information. The employed algorithms are built from computationally efficient arithmetic, logical, permutation, or bit-shift operations. The security properties of symmetric-key algorithms then stem from a complex arrangement of these basic building blocks and from their iterated application in multiple rounds. Important in the scope of constrained devices, symmetric-key algorithms exhibit among the fastest implementations in software and hardware [DK07]. Hence, symmetric-key cryptography constitutes a well-suited cryptographic primitive in the context of the IoT. One of the most popular symmetric-key algorithms is the Advanced Encryption Standard (AES) [NIST01]. This specific cryptographic algorithm is used, e.g., in the IEEE 802.15.4 standard to protect transmitted information at the data link layer.

As a basic design trait, symmetric-key algorithms use the same *secret key* for the encryption of a *plaintext* and the decryption of the encrypted *ciphertext*. Thus, symmetric-key algorithms require the communication end-points to share a common secret key *prior* to secure communication. This requirement, however, raises the problem of securely exchanging the secret key without revealing it to untrusted infrastructure or malicious network participants. The current best practice on the Internet, therefore, is to employ public-key cryptography for a secure exchange of the secret key [BH05, SHSA15]. This use of public-key cryptography in end-to-end security protocols for the IP-based IoT is a key aspect of our research in this thesis.

3.2.3 Public-key Cryptography

Public-key cryptography provides authenticity, confidentiality, integrity protection, and non-repudiation via encryption or signing of transmitted information. Unlike symmetric-key cryptography, the main idea of public-key cryptography is that communication end-points are *not* required to share a common secret key prior to secure communication. Instead, each end-point possesses its own *public/private key-pair*. The private key is secret information that must not be revealed to a third party. In contrast, a core property of public-key cryptography is that the knowledge of the public key does not afford computation of the corresponding private key. Hence, end-points can exchange their public keys as plaintext over untrusted infrastructure and, thus, use public-key cryptography to bootstrap symmetric-key-based primitives. The most prominent public-key algorithms are the Ron Rivest, Adi Shamir, and Leonard Adleman (RSA) [RSA78] and the Digital Signature Algorithm (DSA) [NIST13b].

Public-key cryptography commonly involves big number operations in order to render algorithmic approaches to solve the underlying mathematical problems, e.g., the computation of the discrete logarithm, infeasible. Due to these big number operations, public-key-based algorithms, however, are significantly more resource demanding than their symmetric-key counterparts [DK07]. Consequently, the application of public-key cryptography was long assumed to be impractical for constrained devices [PSW04]. In 2004, Watro et al. [WKC$^+$04], however, showed that public-key cryptography is indeed feasible in the context of constrained devices if used sparsely. Moreover, Elliptic Curve Cryptography (ECC) [Mil86, Kob87] – a special approach to public-key cryptography – allows to further reduce the overhead of public-key cryptography in the context of constrained devices. A key goal of our contributions in Chapter 4 is to enable the computationally efficient and secure use of public-key cryptography in end-to-end IP security protocols for the IoT. Moreover, the handshake delegation architecture, which we will present in Chapter 5, enables secure communication in the IoT for constrained devices with insufficient memory resources for a complete protocol implementation and public-key cryptography.

3.2.3.1 Elliptic Curve Cryptography

ECC is based on the algebraic structure of elliptic curves over finite fields. For this algebraic structure, the best algorithms, which are known to solve the discrete logarithm problem as the underlying mathematical foundation of ECC, currently

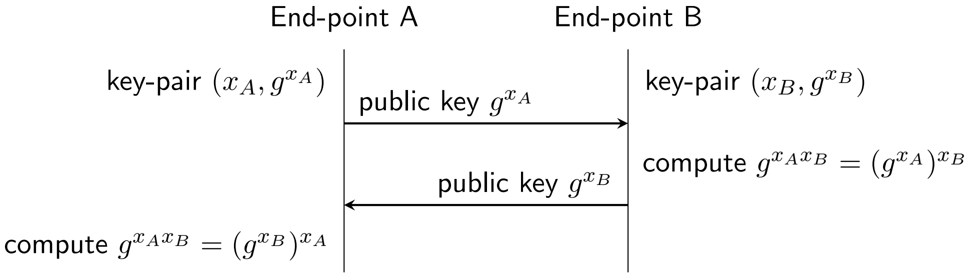

End-point A
End-point B
key-pair (x_A, g^{x_A})
public key g^{x_A}
key-pair (x_B, g^{x_B})
compute $g^{x_A x_B} = (g^{x_A})^{x_B}$
public key g^{x_B}
compute $g^{x_A x_B} = (g^{x_B})^{x_A}$

Figure 3.2 The DH key exchange. Both communication end-points *A* and *B* derive the same secret key $g^{x_A x_B}$. However, only public information g^{x_i} is revealed on the communication path.

have an exponential run-time [HMV04]. This stands in stark contrast to the sub-exponential run-time for the underlying mathematical problems of traditional public-key crypto-systems. As a result of this increased computational hardness, ECC-based primitives offer an equal level of security at shorter key sizes compared to traditional public-key primitives. A 224 bit ECC key, for example, is believed to be cryptographically as secure as a 2048 bit RSA key [NIST12a]. This characteristic of ECC allows to considerably reduce the computation and transmission overheads involved in using public-key cryptography for end-to-end security purposes.

Moreover, recent optimization techniques such as point addition and point doubling based on projective coordinate systems [LN08] afford a further reduction of the computation overhead of ECC primitives. These properties make ECC primitives such as the Elliptic Curve Digital Signature Algorithm (ECDSA) especially suitable for constrained devices [MWS04, LN08]. We note that the end-to-end IP security protocols considered in this thesis all support ECC in their protocol specifications.

3.2.3.2 Diffie-Hellman Key Exchange

The Diffie-Hellman (DH) key exchange [DH76] is a public-key protocol that enables two end-points to *securely* establish a shared symmetric key despite their interaction over untrusted network infrastructure. To this end, the DH key exchange does *not* require the communication end-points to have prior information about each other. Moreover, it resists passive network attacks. As such, it is often used as one of the underlying cryptographic primitives in the design of end-to-end IP security protocols.

As depicted in Figure 3.2, the DH key exchange requires both communication end-points to be in the possession of a public/private key-pair (x_i, g^{x_i}). This key-pair may either be freshly generated or static across multiple key exchanges. In the first step of the DH key exchange, the communication end-points exchange their public DH keys g^{x_i}. The end-points then establish a common secret key $g^{x_A x_B} = (g^{x_A})^{x_B} = (g^{x_B})^{x_A}$ based on their own private key x_i and the received public key g^{x_j}. At this point, the end-points can use the derived secret key to protect subsequent communication via symmetric-key primitives. Importantly, both the computation of the private key x_i from an overheard public key g^{x_i} and the calculation of the secret key $g^{x_A x_B}$ given the public keys g^{x_A} and g^{x_B} are assumed to be computationally infeasible [DK07]. This prevents an eavesdropping adversary from deriving the established secret key.

With respect to the applicability for constrained devices, the DH key exchange revolves around the same mathematic foundation as the previously discussed tradi-

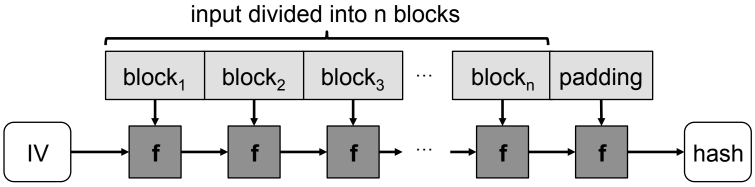


Figure 3.3 The Merkle-Damgård construction. The hash computation starts with a fixed initialization vector (IV) that is algorithm-specific. For each input block, the compression function f is applied to the current input block and the intermediate value of the previous iteration. The result of this operation is the intermediate value for the next iteration. The last input block contains padding information that encodes the length of all input blocks.

tional public-key primitives and, thus, similar considerations apply (see Section 3.2.3). Still, there also exists an ECC variant of the DH key exchange, i.e., the Elliptic Curve Diffie-Hellman (ECDH) key exchange [NIST13a]. This variant allows to significantly reduce the computation and transmission overheads that are caused by the DH key exchange. As a result, an infrequent application of the ECDH key exchange is often considered viable in the context of constrained devices [MWS04].

3.2.4 Cryptographic Hash Functions

Cryptographic hash functions constitute a third category of cryptographic primitives that is predominantly used in the context of network security to provide integrity protection of the transmitted information. As their main characteristic, hash functions map an arbitrary input space to a fixed-length output space. A *cryptographic* hash function then additionally must provide the following three properties:

1. **Pre-image resistance.** Pre-image resistance states that a cryptographic hash function is irreversible, i.e., one-way. Hence, given a hash value y, it must be computationally hard to find a pre-image x such that $H(x) = y$. Notably, we leverage the pre-image resistance property of cryptographic hash functions in the design of the content-chaining scheme, which we will present in Chapter 6.
2. **Second pre-image resistance.** Second pre-image resistance demands that no pre-image x' that differs from the original pre-image x can efficiently be computed. Hence, given x and $y = H(x)$, it must be computationally hard to find $x' \neq x$ such that $H(x') = H(x) = y$. Similar to pre-image resistance, this property also is important for the design of the content-chaining scheme.
3. **Collision resistance.** Collision resistance requires that finding two or more values with the same image is infeasible. Hence, it must be computationally hard to find x and x', such that $H(x) = H(x')$. Otherwise, public-key signatures[1], for instance, would not be able to provide the non-repudiation property as the signer could pretend to have signed a colliding value x' instead of x.

Many cryptographic hash functions used in practice employ the Merkle-Damgård construction [Mer90, Dam90] depicted in Figure 3.3 as their underlying foundation.

---

[1] In digital signature schemes such as ECDSA, the data to be signed is first hashed to a fixed-length representation. The private key of the signature scheme then is used to sign the hash.

Prominent examples include the Secure Hash Standard version 1 (SHA-1) and the Secure Hash Standard version 2 (SHA-2) [NIST12b]. The Merkle-Damgård construction is based on the observation that the arbitrary-length input domain of a hash function can effectively be handled by a function with a fixed-length input domain via an iterated application of this function to fixed-sized blocks of the original hash function input (see Figure 3.3). This observation allows to reduce the problem of building a cryptographic hash function to finding a collision resistant function f with finite input domain $\{0,1\}^{r+s}$ and fixed-length output domain $\{0,1\}^r$, where $s \in \mathbb{N}$ and $s > 0$. Such collision resistant functions f are called *compression functions*.

Hash functions based on the Merkle-Damgård construction typically use customized compression functions that are designed for high performance. Still, symmetric-key primitives such as AES can also be used as compression functions in the Merkle-Damgård construction [BÖS11]. In Chapter 6, we employ such an AES-based construction to leverage the AES hardware support of many constrained devices.

3.2.4.1 Hash Chains

Hash chains are based on the iterated application of a hash function to an initial seed value in order to generate a chain of *efficiently verifiable one-time authentication tokens*. Hash chains were first proposed by Lamport [Lam81] to protect remote password authentication against an eavesdropping or replaying adversary. Gennaro et al. [GR97] subsequently showed that hash chains can also be employed to efficiently sign digital streams of finite length. Since then, hash chains have been applied to various communication scenarios including broadcast and multicast authentication [PTSC00, PST+02] as well as end-to-middle authentication [HHK+09, Hee11].

As illustrated in Figure 3.4, the basic idea behind a hash chain is to provide an interlinked chain of one-time tokens that can be revealed sequentially for authentication purposes. To this end, a hash chain is created by first drawing a random *seed* value h_0. The hash function $H(\cdot)$ then is applied to this seed value to generate the first interlinked one-time token. Subsequently, the hash function is iteratively applied to the previous hash chain element until a hash chain of the desired length n is derived. The resulting hash chain then consists of the token sequence $(h_0, h_1, h_2, \ldots, h_{n-1})$.

The last element h_{n-1} of this sequence is also known as the *anchor element*. To use the created hash chain for authentication purposes, the sender, who is the creator of the hash chain, first has to securely transmit the anchor element of the hash chain to the verifier. To this end, public-key signatures over the anchor element are typically

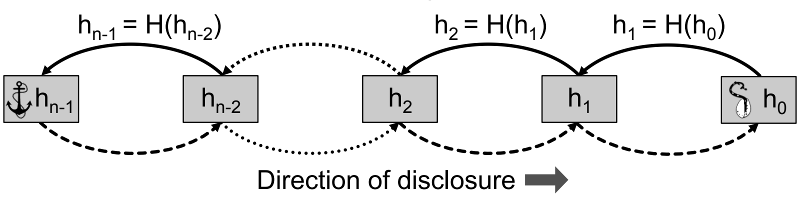


Figure 3.4 A hash chain consisting of n elements. The generation of the hash chain elements proceeds from the seed to the anchor element, whereas token disclosure advances from the anchor to the seed element in order to prevent token forgery by an adversary.

used to bind the created hash chain to the cryptographic identity of the sender. The remaining hash chain elements, however, have to be kept secret at this stage.

Once this setup phase has completed successfully, the sender traverses the hash chain in reverse order to authenticate itself to the verifier. That is, if the hash chain element h_i was revealed during the last authentication procedure, the sender selects h_{i-1} in order to re-authenticate itself to the verifier. When receiving h_{i-1}, the verifier can validate the authenticity of the sender due to the following two properties:

1. Given a previously verified hash chain element h_i, the verifier can efficiently validate that a newly received element h_{i-1} belongs to the same hash chain by performing a single hash calculation, i.e., $h_i \overset{?}{=} H(h_{i-1})$, as shown in Figure 3.4.
2. Given h_i, the verifier can also validate that an element h_{i-j} for $0 < j \leq i$ belongs to the same hash chain by iteratively applying the hash function $H(\cdot)$ to the received element j times: $h_i \overset{?}{=} H^j(h_{i-j})$. Hence, the iterative construction of the hash chain also enables the verifier to validate hash chain elements that are not immediate predecessors of a previously verified element.

An adversary cannot compute an undisclosed hash chain element h_{i-2} that is closer to the seed value than a just revealed element h_{i-1}. This is due to the pre-image resistance property of the underlying hash function (see Section 3.2.4). Moreover, the adversary also cannot forge a hash token h'_{i-1} that would validate against a previously disclosed hash chain element h_i, i.e., $H(h'_{i-1}) = H(h_{i-1}) = h_i$, as a result of the second pre-image resistance property of the employed hash function. These security guarantees along with the ability to efficiently generate and verify their comprising one-time tokens make hash chains an effective cryptographic primitive for data origin authentication in the context of constrained devices. We note that, for this reason, we leverage the basic concept of a hash chain as the underlying foundation for the content-chaining scheme, which we will present in Chapter 6.

3.2.5 Message Authentication and Integrity Protection

The main goal of Message Authentication Codes (MACs) is to enable data origin authentication as well as integrity protection of network transmissions [DK07]. MAC constructions are commonly based on symmetric-key primitives or cryptographic hash functions. Moreover, they employ a shared secret to prevent an adversary from modifying the packet payload without invalidating the appended MAC value.

An often-used MAC construction is the Hash-based Message Authentication Code (HMAC) that employs a cryptographic hash function such as SHA-1 as its main building block [BCK96, KBC97]. As an alternative to hash-based MACs, symmetric-key-based constructions such as the Cipher-based MAC (CMAC) [NIST05] can also be used for efficient authentication and integrity protection. Such symmetric-key-based approaches are increasingly demanded in the context of the IoT. This is for the following two reasons. First, constrained devices that are equipped, e.g., with IEEE 802.15.4 radio modules often provide hardware support for the AES block cipher. Thus, by relying on an AES-based MAC construction, the computation and memory overheads of the underlying cryptographic algorithm can significantly be reduced by

Threat **Security**	Eavesdropping	Paket Injection	Modification	Impersonation	Access Violation	DoS
Authentication		x		x		
Confidentiality	x					
Data Integrity			x			
Non-repudiation						
Authorization					x	
Availability						x
Freshness				x		

Table 3.2 Overview of main objectives of network security and the presented network attacks. By combining multiple cryptographic primitives and by incorporating additional security mechanisms, security protocols offer protection against a wide range of network attacks.

leveraging this hardware support. Second, even without such hardware support, the scarce memory resources of constrained devices can effectively be conserved by building the MAC primitive based on the same underlying symmetric-key algorithm as the encryption primitive. This shared implementation then allows to abate most of the memory overhead that would otherwise be caused by a separate hash algorithm.

3.3 Network Security in the IP-based IoT

The cryptographic primitives outlined in the previous section constitute basic building blocks to assure the authenticity, confidentiality, data integrity, and non-repudiation of transmitted information. A single cryptographic building block, however, often does not suffice to protect communication end-points against the wide range of network attacks that these end-points may be exposed to. Thus, security protocols typically combine multiple cryptographic primitives in their protocol design. Moreover, they incorporate additional security mechanisms to fulfill security objectives that *include* but extend on the fundamental objectives of cryptography [LRA09].

We now first present these additional objectives of network security in Section 3.3.1 and discuss their relevance in the context of the network attacks in Section 3.1 and our contributions in this thesis. We then provide an overview of the different types of network security that realize these objectives in Section 3.3.2. Our main goal thereby is to highlight the role of end-to-end security with respect to other network security solutions for the IP-based IoT. Following this overview, we introduce the DTLS, HIP DEX, and Minimal IKEv2 protocol profiles and variants for the IP-based IoT in Section 3.3.3 to Section 3.3.5. These end-to-end IP security protocol adaptations build the basis for our work in Chapters 4 and 5. Finally, we analytically compare these protocol adaptations and highlight their main commonalities and differences.

3.3.1 Additional Objectives of Network Security

As summarized in Table 3.2 (see last three rows), security protocols commonly realize additional objectives that extend on the main objectives of cryptography. These additional security objectives most notably are [WAR06]:

Authorization: The main objective of authorization is to enable an end-point to determine if its communication partner is allowed to access a requested resource. As such, the primary goal of authorization is to mitigate *access violations*. Importantly, while authentication is an essential pre-condition for authorization, it does not explicitly restrict network interactions to approved communication end-points. Instead, additional authorization mechanisms and policies are required to limit access to restricted resources based on the authenticated identity of the communication partner. We note that the handshake delegation architecture, which we will present in Chapter 5, efficiently provides such authentication and authorization in the context of constrained devices.

Availability: The central goal of availability is to protect communication end-points against attacks that aim at degrading its provided services, i.e., *DoS attacks*. Availability is a highly critical security objective in the IoT as constrained devices are commonly equipped with largely inferior hardware compared to what might be at the disposal of a potential adversary [RL09]. Hence, one of the key contributions in Chapter 4 is the introduction of a DoS protection mechanism for end-to-end IP security protocols that we specifically designed with the high resource asymmetry of the IoT in mind.

Freshness: The main objective of freshness is to assure that network packets are constructed at the time of interaction instead of being *replayed* packets of prior communication. Freshness is especially important regarding end-to-end IP security protocols as these protocols commonly verify the authenticity of the communicating end-points. If freshness was neglected by these protocols, an adversary could simply undermine the employed authentication mechanisms by replaying authentic recorded messages of prior communication. For our contributions in Chapters 4, 5, and 6, we require that freshness of packets can be verified whenever relying on the authenticity of transmitted information.

We now continue with a brief overview of security protocols for the IP-based IoT that realize the above security objectives. We note that the objective of authorization is typically provided by dedicated Authentication, Authorization, and Accounting (AAA) protocols such as Diameter [FALZ12], which are not discussed below.

3.3.2 Overview of Protocol-based Security Solutions

While several specialized network security approaches exist in the context of WSN research [WLSC07, ZV10], we focus our discussion in this section on IP security protocols that are currently considered or are already deployed within the scope of IoT network scenarios. These protocols denote widely accepted standards that are specifically designed to fit the IP networking paradigm. We note that the overview provided here is based on previous work published in [HGMH$^+$11, GMKK$^+$13a].

For our discussion, we categorize the different IP security protocols according to the lifecycle of a constrained device. This lifecycle starts with a bootstrapping phase and cycles through iterated operational and maintenance phases once the device is readily bootstrapped. The security protocols employed during the *bootstrapping phase* typically fulfill the following two purposes. On the one hand, they facilitate authentication of a joining device towards the constrained node network. On the other hand, they enable the network to provision the joining device with the necessary keying material to securely interact with other communication end-points.

The Extensible Authentication Protocol (EAP) [ABV+04] is a prominent example of a network admission and key provisioning protocol that is currently employed in constrained node networks, e.g., for the bootstrapping procedure in ZigBee IP [ZBIP13]. EAP is an authentication framework that supports a wide range of authentication methods including pre-shared keys. Important in the scope of constrained node networks, EAP directly operates at the data link layer and only supports unfragmented packet transmissions. Hence, when deployed in these networks, it typically needs to be encapsulated by the Protocol for Carrying Authentication for Network Access (PANA) [FOP+08], a network-layer transport for EAP. This encapsulation affords IP-based multi-hop forwarding and 6LoWPAN fragmentation of EAP messages in constrained node networks. Notably, EAP and PANA can also be used to re-bootstrap a constrained device after completing a *maintenance* phase, e.g., if a firmware update necessitates the re-provisioning of the network security material.

After bootstrapping or maintenance, the constrained device transitions to its *operational state*. Here, the device uses the provisioned keying material as well as potentially pre-provisioned security material to securely interact with other communication end-points. For our discussion, we further divide network security protocols in the operational phase into hop-by-hop and end-to-end approaches.

Hop-by-hop security is classically provided at the data link layer. The IEEE 802.15.4 standard, e.g., specifies dedicated security mechanisms that enable a constrained device to encrypt and to protect the integrity of the frame payload. Such protection then prevents a network-external adversary, who is within radio range of the targeted constrained node network, from eavesdropping on legitimate communication and from injecting malicious packets into that network. At the same time, hop-by-hop encryption still affords legitimate on-path devices, e.g., to forward packets based on IP header information. This is because each intermediate hop decrypts network packets upon reception and re-encrypts them before packet forwarding.

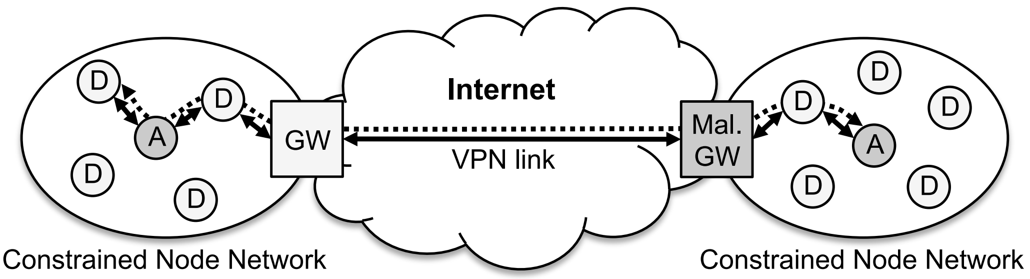


Figure 3.5 Network scenario consisting of two interconnected constrained node networks. Protection is based on link layer security between constrained devices (D) and a VPN link between the involved gateways (GW). Potential adversaries are marked in dark gray. Notably, on-path adversaries can eavesdrop or modify traversing network packets unnoticed as hop-by-hop security is applied on a per-hop basis. Moreover, hop-by-hop security cannot provide end-to-end security guarantees that sensitive information was protected on the entire packet forwarding path and does not allow the end-points to cryptographically authenticate each other.

While providing adequate protection inside a constrained node network with trustworthy network participants, hop-by-hop security often does not suffice to protect network interactions over untrusted network infrastructure or in network environments with malicious network participants. This is for the following three reasons (see Figure 3.5). First, payload decryption at each hop on the forwarding path exposes the packet content to potentially malicious on-path network entities. Second, as each hop strips and applies packet protection, there are no end-to-end guarantees that sensitive information was actually protected on the entire forwarding path. Third, the per-hop semantics of hop-by-hop security do not allow the end-points to cryptographically authenticate each other. Thus, hop-by-hop security requires the end-points to fully trust each hop on the forwarding path. This also includes potentially unknown network entities in case of communication across network domains.

The goal of *end-to-end IP security protocols* is to provide these missing end-to-end security guarantees to the communication end-points. This renders end-to-end security specifically important for IoT network scenarios that involve cross-domain interactions. End-points that belong to independent network domains, however, may be configured with different preferences regarding cryptographic primitives and protocol mechanisms as well as the pursued level of security. Hence, besides peer authentication and key agreement, end-to-end security protocols regularly also need to negotiate the protocol parameters that are mutually supported by the end-points.

Concerning the location in the network stack, end-to-end IP security protocols are commonly located above the network layer. This affords packet forwarding based on IP header information. At the same time, this circumstance also allows to protect protocol and application data in an end-to-end manner. The most prominent end-to-end IP security protocols that are situated on top of the network layer are HIPv2 [MHJH15] and IKEv2 [KHN$^+$14]. Similarly, TLS [DR08] provides end-to-end security above the transport layer, e.g., as the default security protocol in the world wide web. Importantly, HIPv2, IKEv2, and TLS all provide channel security guarantees that are largely independent from the protected applications. Hence, these protocols constitute *general-purpose solutions* for end-to-end security.

Similar to these general-purpose protocols, object security mechanisms such as JSON Object Signing and Encryption (JOSE) [Bar14] also allow to provide end-to-end security at the application layer, e.g., for the Extensible Messaging and Presence Protocol (XMPP) [SA11, MW14]. Authentication and key management for these object-based approaches, however, typically are highly application and scenario-specific. We, therefore, consider these application-specific approaches out-of-scope in this thesis. Instead, we investigate the applicability of general-purpose protocols such as TLS, HIPv2, and IKEv2. More precisely, we focus on their protocol profiles and variants, i.e., DTLS, HIP DEX, and Minimal IKEv2, that are currently devised at the IETF with the special device and network constraints of the IoT in mind. We now proceed with a detailed description of these IP security protocol adaptations.

3.3.3 Datagram Transport Layer Security

DTLS [RM12] is the datagram variant of the widely used TLS protocol. DTLS was originally developed to provide TLS-like network security for streaming applications

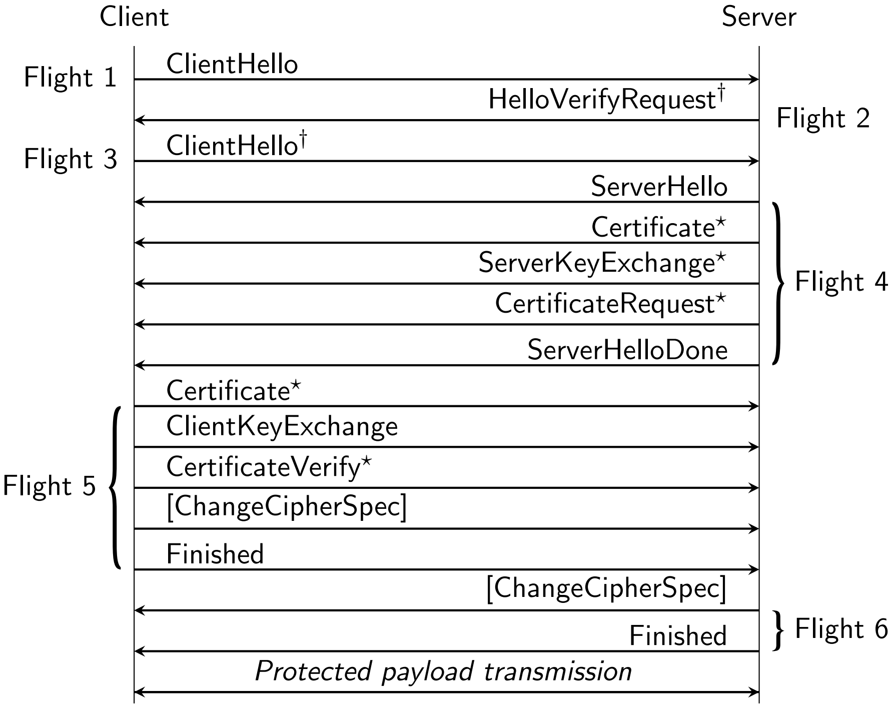


Figure 3.6 Sequence diagram of the DTLS handshake. Messages marked with a dagger (†) implement a return-routability test for DoS protection purposes. Starred messages (⋆) are optional and are only required depending on the negotiated cryptographic primitives.

such as VoIP [MR04]. To this end, the underlying reliable transport of TLS was replaced with an unreliable transport such as UDP. Besides this modification, one of the main goals of the DTLS specification is to preserve the protocol semantics and security guarantees of TLS. This allows to re-use most existing TLS protocol extensions for DTLS, e.g., session resumption without server-side state [SZET08].

The DTLS protocol specification assumes a connection-oriented client/server architecture. Hence, a client and a server must first perform a connection establishment handshake before securely transmitting application data. During this handshake, the peers most notably negotiate the mutually supported cryptographic primitives and protocol mechanisms, authenticate each other, and agree on secret keying material for the subsequent protection of application data. We now briefly outline the DTLS connection establishment handshake as the main protocol component under investigation in this thesis. An overview of this handshake is illustrated in Figure 3.6.

The DTLS handshake consists of 10 to 15 messages. These handshake messages are subsumed in six message flights. The handshake is triggered by the client with the `ClientHello` message. When receiving this message, the server responds with a `HelloVerifyRequest` that starts a return-routability test for DoS protection purposes. This test is designed to prevent spoofing-based DoS attacks against computationally or memory-wise expensive handshake operations at the DTLS server as well as an amplification-based DoS attack against the DTLS client [MR04]. Specifically, the return-routability test enables the DTLS server to verify that the source IP address of the first `ClientHello` message is actually reachable and that the end-point located behind this address indeed wants to establish a DTLS connection. Message flights 3, 4, and 5 then constitute the central part of the DTLS handshake. Here, the peers first negotiate the cryptographic primitives and key sizes for the subsequent cryptographic protocol operations. With respect to these operations, DTLS supports a wide range of public-key primitives and optionally provides for a purely symmetric-key-based handshake [ET05]. The peers then authenticate each other and carry out a key agreement for the protection of subsequent handshake messages.

The handshake concludes with the `Finished` messages in flights 5 and 6. These messages allow both peers to verify the correctness of the performed handshake. To this end, the peers compute a cryptographic hash over the exchanged handshake messages and transmit this information to the communication partner in encrypted form. Successful decryption and hash verification then validate the derived session keys and the integrity of the entire handshake. After the handshake has completed, the peers use the derived session keys for the protection of application data.

Unlike TLS, DTLS messages are transmitted over an unreliable transport channel. Hence, handshake messages that are lost on the communication path must be recovered by the DTLS protocol itself. To this end, DTLS defines a simple retransmission mechanism that operates on a *per-flight basis*. This per-flight semantic implies that the loss of a single handshake message leads to the retransmission of the entire corresponding message flight. To determine whether a handshake message was lost, the DTLS specification recommends the use of a retransmission timeout that should initially be set to 1 s. Hence, if a peer does not start receiving the expected response flight within 1 s after the transmission of its own message flight, it considers the handshake messages from its own message flight lost and must retransmit these handshake messages. Moreover, the peer doubles the timeout for each additional retransmission of the same message flight until it reaches a maximum timeout value of 60 s. This back-off mechanisms is designed to avoid network congestion [Jac88].

Importantly, the DTLS protocol currently receives significant attention at the IETF in the context of the IoT. The CoAP specification, for example, recommends DTLS as the preferred solution to secure the CoAP protocol [SHB14]. Moreover, the DTLS In Constrained Environments (DICE) working group at the IETF currently defines a DTLS profile for the IoT [TF15]. This profile prescribes or precludes certain cryptographic cipher suites and protocol extensions to decrease the memory overhead of a DTLS protocol implementation as well as to reduce the transmission overhead incurred by the protocol handshake. We note that we use the term “DTLS” as a reference to this DTLS profile in this thesis if not mentioned otherwise.

3.3.4 Host Identity Protocol Diet EXchange

HIP DEX [MH14] is a protocol variant of HIPv2 and is specifically developed with constrained node networks in mind. As such, its protocol specification makes two key concessions that are exclusively motivated by IoT device and network constraints. First, while the original HIPv2 protocol design employs public-key signatures, an ephemeral DH key agreement[2], and a cryptographic hash function, the HIP DEX protocol specification forfeits these cryptographic primitives to reduce the overhead

---

[2] The ephemeral DH key agreement is based on a temporary DH key-pair that is freshly generated for each handshake. As a result, a compromise of the DH key-pair does not compromise the security of previous connections (perfect forward secrecy) [Shi07]. The repeated generation of DH key-pairs, however, causes a considerable computation overhead in the context of constrained devices [LN08].

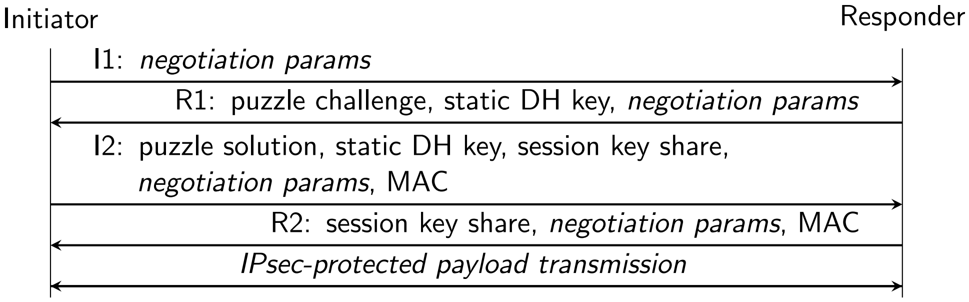

Initiator
Responder
I1: negotiation params
R1: puzzle challenge, static DH key, negotiation params
I2: puzzle solution, static DH key, session key share, negotiation params, MAC
R2: session key share, negotiation params, MAC
IPsec-protected payload transmission

Figure 3.7 Sequence diagram of the HIP DEX handshake. Cipher suite negotiation, peer authentication, and key agreement are performed in a four-way message exchange.

incurred by the protocol handshake. Instead, HIP DEX defines an *adapted connection establishment handshake* that is based on static DH keys for mutual peer authentication and key agreement purposes (see Figure 3.7). Moreover, the protocol design restricts itself to purely symmetric-key-based primitives for confidentiality and integrity protection. Second, the HIP DEX specification defines an *aggressive retransmission mechanism* that is intended to handle the increased packet loss in constrained node networks when compared to conventional IP networks [Mos12]. This retransmission mechanism requires the Initiator to continually send I1 or I2 messages in short succession until the reception of the corresponding R1 or R2 response, respectively. In contrast, the Responder does not trigger retransmissions and simply must reply to each received Initiator message. Beyond these modifications, the HIP DEX specification aims at preserving the general HIPv2 protocol semantics, similar to DTLS. We now briefly describe the HIP DEX connection establishment handshake as the most important protocol component in scope of this thesis.

As shown in Figure 3.7, the HIP DEX protocol handshake consists of four messages and is triggered by the Initiator via the I1 message. The Responder replies to this initial handshake message with an R1 response that contains a cryptographic challenge among other information. This challenge is part of a cryptographic puzzle-based DoS protection mechanism that requires the Initiator to spent a Responder-defined amount of computational resources before proceeding with the protocol handshake. Consequently, the reception of a correct puzzle solution in the I2 message allows the Responder to deduce that the Initiator is sincere in performing the protocol handshake because it churned the requested amount of computational resources.

In addition to this DoS protection mechanism, the Initiator and the Responder also perform a DH key exchange (see Section 3.2.3.2). To this end, the peers exchange their public static DH keys in the R1 and I2 messages. Afterwards, the peers use the derived secret for a MAC-based integrity protection of the I2 and R2 messages. The successful verification of these MAC values then not only confirms the integrity of the corresponding handshake messages, but also authenticates the communication partner. This is because only the legitimate communication end-points are in the possession of the private DH keys that correspond to the previously exchanged public DH keys. Hence, only these end-points can derive the secret keying material that is required to generate a valid MAC value for a given handshake message.

Finally, the peers leverage the derived secret to encrypt randomly chosen key shares for each other. The concatenation of these key shares then is the basis for the *session key* that is used for the subsequent protection of application data. HIP DEX thereby does not define its own protection mechanism for application data but employs an IPsec-protected payload channel similar to the original HIPv2 protocol [JMM15].

3.3.4.1 Remarks

The HIP DEX protocol is currently proposed for standardization at the IETF and is still under active development. As a result, an updated protocol revision has been released since our work in Chapter 4. The protocol revision employed in this thesis [Mos12] primarily differs from the most recent protocol specification [MH14] with respect to the following two modifications. First, we identified a potential downgrading attack targeting the negotiation of the supported cipher suites and of the application data protection mechanisms in the unprotected R1 message. To mitigate this attack, we proposed to re-include the affected negotiation parameters in the integrity-protected R2 message. This change enables the communication endpoints – even though with potentially weaker cryptographic primitives than mutually desired – to verify if these negotiation parameters were altered during transmission of the R1 message. Second, the refined retransmission mechanism, which we will present in Chapter 4, recently replaced the original aggressive retransmission strategy of HIP DEX due to the improved retransmission properties of our approach.

3.3.5 Minimal Internet Key Exchange Protocol Version 2

Minimal IKEv2 [Kiv15] is a profile of the IKEv2 protocol for the IoT that has recently been proposed for standardization at the IETF. The main goal of this profile is to reduce the complexities involved in implementing the IKEv2 protocol. To this end, it excludes most optional functionality of IKEv2, e.g., the creation of multiple child security associations within a single session context. This allows to limit the Minimal IKEv2 handshake to the first four messages of the IKEv2 handshake (see Figure 3.8). Moreover, the Minimal IKEv2 specification focuses on the use of symmetric-key instead of public-key cryptography for peer authentication purposes. Still, Minimal IKEv2 does not completely eliminate the need for public-key cryptography as it continues to rely on an ephemeral DH key agreement for the establishment of a confidentiality- and integrity-protected channel. We now continue with an overview of the Minimal IKEv2 connection establishment handshake.

Similar to HIP DEX, the Minimal IKEv2 protocol handshake consists of a four-way message exchange that is triggered by the Initiator. The handshake proceeds in

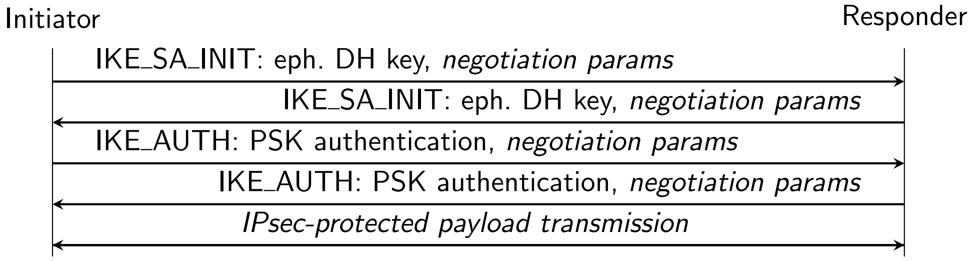

Initiator
Responder
IKE_SA_INIT: eph. DH key, *negotiation params*
IKE_SA_INIT: eph. DH key, *negotiation params*
IKE_AUTH: PSK authentication, *negotiation params*
IKE_AUTH: PSK authentication, *negotiation params*
IPsec-protected payload transmission

Figure 3.8 Sequence diagram of the Minimal IKEv2 handshake. The IKE_AUTH exchange and the IPsec payload channel are protected based on an ephemeral DH key agreement. The peer authentication performed during the IKE_AUTH exchange is based on Pre-Shared Keys (PSKs).

two message pairs, i.e., the IKE_SA_INIT followed by the IKE_AUTH exchange (see Figure 3.8). During the IKE_SA_INIT, the Initiator and the Responder primarily negotiate protocol parameters and exchange their ephemeral DH keys. These keys allow the peers to establish a secure channel for the following IKE_AUTH message exchange as well as to derive secret keying material for the subsequent establishment of an IPsec-protected payload channel. The peers then authenticate each other during the IKE_AUTH exchange and finally setup the aforementioned IPsec channel for the secure transmission of application data. In this context, it is worth noting that the communication end-points are expected to use symmetric-key cryptography for peer authentication purposes. Hence, the end-points have to be in the possession of a common secret prior to the establishment of a secure connection via Minimal IKEv2.

Like DTLS and HIP DEX, the Minimal IKEv2 protocol design incorporates a retransmission mechanism for lost handshake messages. This retransmission mechanism is a hybrid solution of the strategies employed by DTLS and HIP DEX. Specifically, retransmissions in Minimal IKEv2 are triggered by the Initiator whereas the Responder reacts to retransmissions, equal to HIP DEX. Similar to DTLS, repeated message loss leads to an exponential back-off of the retransmission timeout. Moreover, while IKEv2 provides a return-routability test for DoS protection comparable to DTLS, Minimal IKEv2 forgoes this optional mechanism in its protocol design.

3.3.6 Protocol Comparison

We now analytically compare the main handshake properties of the DTLS, HIP DEX, and Minimal IKEv2 protocols. By doing so, our goal is to determine the strengths and weaknesses of the considered protocols with respect to the special device and network characteristics in the IoT. We then leverage this gained understanding in order to guide our research in Chapters 4 and 5. We refer to Table 3.3 for an overview of the main results of the following brief analytical protocol comparison.

The most prominent difference between the DTLS, HIP DEX, and Minimal IKEv2 protocols is the required number of handshake messages. More precisely, while DTLS commonly requires the transmission of 10 to 15 handshake messages, the HIP DEX and Minimal IKEv2 handshakes only consist of a total of four messages. Still, it is worth noting that all three protocols have to transfer similar handshake information. Moreover, DTLS allows multiple handshake messages to be sent in a single network packet. Thus, the overall transmission overheads of the different handshakes do not differ to the significant extend indicated by the diverging number of handshake messages. Similarly, the retransmission of a DTLS message flight resembles the retransmission of a HIP DEX or Minimal IKEv2 handshake message when sending entire DTLS message flights in a single packet. Still, DTLS end-points also have to be able to process message flights that are transmitted via multiple network packets as this likewise constitutes protocol-compliant transmission behavior. The DTLS handshake, therefore, exhibits an increased message processing complexity and may incur higher transmission overheads compared to HIP DEX and Minimal IKEv2.

The main strength of the DTLS protocol is its agility with respect to the supported cryptographic primitives for peer authentication and key agreement purposes. Here, the DTLS protocol specification allows to either use public-key or symmetric-key cryptography. As a result, the DTLS protocol design provides for an efficient

Protocol / Property	DTLS	HIP DEX	Minimal IKEv2
Handshake messages	10 - 15	4	4
Retransmissions	Flight-based	Packet-based	Packet-based
Public-key primitives	Recommended	Mandatory	Mandatory
Other primitives	Hash, Sym.-key	Sym.-key crypto	Sym.-key crypto
DoS protection	Return routability	Cryptogr. puzzle	–
Additional protocol mechanisms	Fragmentation, Compression	Mobility, Multi-homing	–

Table 3.3 Comparison results for the main handshake properties of the DTLS, HIP DEX, and Minimal IKEv2 protocols. HIP DEX and Minimal IKEv2 mandate the use of public-key cryptography during the protocol handshake, whereas DTLS also provides for a purely symmetric-key-based handshake. The extensive handshake length and the per-flight retransmission semantics, however, render DTLS more complex than HIP DEX and Minimal IKEv2.

symmetric-key-based handshake if the communication end-points already possess a common secret *prior* to the actual handshake. However, such pre-shared secret keys often are not readily available, e.g., when constrained devices communicate with other constrained devices, hosts, or services in remote network domains. For such scenarios, the DTLS protocol affords the use of public-key cryptography for authentication and key agreement across potentially untrusted network infrastructure.

In contrast to the DTLS protocol, HIP DEX and Minimal IKEv2 both mandate the use of public-key cryptography for a DH key exchange. In doing so, Minimal IKEv2 is computationally more demanding than HIP DEX as it requires an *ephemeral* DH key agreement that – compared to the static DH keys in HIP DEX – additionally necessitates the peers to generate new DH keys for each protocol handshake. Despite their mandatory use of public-key cryptography, it is worth noting that HIP DEX and Minimal IKEv2 eliminate the need for a cryptographic hash function in their protocol design. This allows to slightly reduce the memory overhead of the employed cryptographic primitives. Still, we note that this overhead reduction typically does not suffice to compensate for the memory requirements of public-key cryptography.

Concerning DoS attacks against the protocol handshake, HIP DEX incorporates the most sophisticated protection mechanism of the considered protocols. The employed puzzle-based mechanism enables a resource-constrained Responder to demand a resource commitment from the Initiator before proceeding with expensive handshake operations. This mechanisms also implicitly incorporates the challenge/response-based return-routability test that is employed in the DTLS protocol design. Interestingly, Minimal IKEv2 does not include a DoS protection mechanism in its protocol design. Thus, with Minimal IKEv2, constrained devices are exposed to DoS attacks during the protocol handshake that HIP DEX and DTLS can protect against.

In addition, DTLS and HIP DEX also provide further protocol mechanisms that are not strictly required for end-to-end security purposes. For DTLS, these mechanisms include message compression and fragmentation. HIP DEX additionally supports end-point mobility and multi-homing. We note that we do not specifically consider these protocol mechanisms in this thesis for the following reasons. Content compression before encryption has been shown to leak information about the uncompressed packet content [Kel02] and, thus, is no longer considered secure [SHSA15]. Moreover, message fragmentation at the DTLS layer is less efficient than fragmentation at the 6LoWPAN layer as each DTLS fragment contains IP and UDP header information [Har14a]. Finally, mobility and multi-home are out-of-scope of our work.

3.4 Problem Statement and Research Challenges

The main goal of this thesis is to contribute to the efficient and secure operation of end-to-end security protocols in the IP-based IoT. To achieve this goal, a suitable network security solution must account for the special device and network characteristics in the embedded domain as well as for the high resource asymmetry in the IoT (see Section 2.5). The DTLS, HIP DEX, and Minimal IKEv2 protocol adaptations constitute candidate solutions that should already satisfy the above requirements as they are intended to provide secure end-to-end communication between constrained devices, hosts, and services. Based on our brief analytical protocol comparison and our practical experimentation experience, we, however, identify three open research challenges (RCs) with respect to the secure and efficient operation of these end-to-end IP security protocol adaptations in the context of constrained devices:

RC1 - High handshake computation and transmission overheads

DTLS, HIP DEX, and Minimal IKEv2 all mandate or recommend the use of public-key cryptography in their protocol design. As we will show in Chapter 4, this use of public-key cryptography, however, causes several design-level security and performance issues in the context of constrained devices. Most importantly, we find that the long processing times of public-key operations significantly hamper the availability and response time of constrained devices during the protocol handshake. Hence, the design of end-to-end security protocols for the IP-based IoT must be adapted to reduce the need for computationally expensive cryptographic operations, to protect networked embedded devices against DoS attacks targeting these operations, and to account for the varying processing times of the different handshake messages.

In addition, a key design goal of the DTLS, HIP DEX, and Minimal IKEv2 specifications is to preserve the original protocol semantics of TLS, HIPv2, and IKEv2, respectively. TLS, HIPv2, and IKEv2, however, were designed for extensibility and flexibility, whereas protocol conciseness only was a secondary goal. As a result, certain design decisions in the DTLS, HIP DEX, and Minimal IKEv2 specifications, e.g., the conservatively large size of fixed-length header fields, stand in direct conflict with the requirement for protocol conciseness in the IoT. This requirement stems from the fact that the radio modules constitute a major energy consumer on constrained devices [PK00].

In Chapter 4, we will provide a detailed analysis of the above protocol efficiency and security issues as well as quantify their impact based on the HIP DEX protocol. Moreover, we will present the design of our four complementary, lightweight protocol extensions that address the identified protocol issues.

RC2 - Prohibitive code size for memory-constrained devices

The contributions in research challenge RC1 account for the extensive run-time requirements, i.e., the high handshake computation and transmission overheads, of the DTLS, HIP DEX, and Minimal IKEv2 protocols in the context of constrained devices. As we will show in Chapter 5, these protocols, however, also cause significant RAM and ROM overheads when employing public-key cryptography in the protocol handshake. These memory requirements render the use of public-key cryptography in the design of the above end-to-end IP security protocols infeasible for a wide range of memory-constrained devices.

To still enable such memory-constrained devices to employ standard end-to-end IP security protocols for secure communication in the IoT, we will introduce the handshake delegation architecture in Chapter 5. The key idea behind this architecture is to physically separate the connection establishment phase from the subsequent protection of application data and to offload the connection establishment handshake to an off-path, trusted delegation server.

RC3 - Vulnerable 6LoWPAN fragmentation of handshake messages

Our contributions in research challenges RC1 and RC2 enable the secure and efficient operation of DTLS, HIP DEX, and Minimal IKEv2 at the security protocol level. However, we observe that the packet processing at the lower layers in the network stack for constrained devices also needs to be considered in order to fully provide for the secure operation of the DTLS, HIP DEX, and Minimal IKEv2 protocols in the embedded domain. Specifically, we find that select handshake messages need to be fragmented at the 6LoWPAN layer for transmission inside a constrained node network. As we will show in Chapter 6, this fragmentation mechanism, however, is vulnerable to potential DoS attacks. An adversary can exploit these fragmentation-based DoS attacks to block the correct reassembly of fragmented handshake messages and, thus, can prevent the DTLS, HIP DEX, and Minimal IKEv2 handshakes from completing.

In Chapter 6, we will provide a detailed security analysis of the 6LoWPAN fragmentation mechanism and we will discuss the identified fragmentation attacks. Moreover, we will present two complementary defense mechanisms that protect constrained devices against these attacks. The devised defense mechanisms then afford the missing protection of fragmented handshake messages.

We now continue with the presentation of our lightweight security protocol extensions and architectural considerations that address the above research challenges. Combined, these contributions provide a comprehensive, yet efficient solution for authentication, authorization, and secure communication in the IP-based IoT.

4

Tailoring the Protocol Design

The DTLS, HIP DEX, and Minimal IKEv2 protocol adaptations (see Sections 3.3.3 to 3.3.5) are proposed for standardization at the IETF with the intention to adapt standard end-to-end security to the IP-based IoT. In principle, these protocol adaptations should already consider the special device and network characteristics in the embedded domain as well as the high resource asymmetry in the IoT. However, based on our analysis, we identify several design-level efficiency and security issues that render the deployment of these protocols in the IoT inefficient and insecure.

Our contributions in this chapter are twofold. First, we present the results of our protocol analyses and quantify the identified design-level protocol issues for the HIP DEX protocol. We chose HIP DEX for our detailed protocol analysis as it already features a highly adapted protocol handshake (see Section 3.3.6). As a second contribution, we then present three complementary, lightweight protocol extensions that address the identified design-level efficiency and security issues. These mechanisms adapt the HIP DEX protocol design to the *extensive computation requirements* on constrained devices. Moreover, we show how the core ideas of these protocol extensions also translate to DTLS and Minimal IKEv2. We additionally introduce an evolvable compression layer called Slimfit that we specifically design to reduce the *high transmission overhead* of the HIP DEX protocol. Combined, these contributions tailor the protocol design of HIP DEX in particular and of standard end-to-end IP security in general to the special device and network characteristics in the embedded domain as well as to the high resource asymmetry in the IoT.

The remainder of this chapter is structured as follows. In Section 4.1, we address the protocol handshake issues that are caused by the high computation overhead in the context of constrained devices. Section 4.2 then focuses on the transmission overhead of the HIP DEX handshake and presents the design of the Slimfit compression layer. In both sections, we provide a detailed analysis of the problem space, describe the design of the devised solutions, discuss their security considerations as well as their related work, present the evaluation results, and review the main contributions. Finally, we conclude this chapter in Section 4.3 with a brief summary. We note that the contents of this chapter is based on our published work in [HHW11, HWZ^{+}13, HHHW13] and our on-going standardization efforts in [HGS13, HHH13, MH14].

4.1 Designing for High Computation Requirements

Concerning the computation overhead of DTLS, HIP DEX, and Minimal IKEv2, the cryptographic operations that constrained devices must perform during the protocol handshake typically outweigh the remaining non-cryptographic message processing costs. This is especially true in case of public-key cryptography as discussed in Section 3.2. Public-key cryptography, however, plays a vital role for peer authentication and key agreement purposes across independent network domains. This is predominantly due to its high scalability – a single public/private key-pair suffices to authenticate a communication end-point to multiple communication partners – and its unchallenging key management properties. More precisely, public-key cryptography allows to exchange the *public* portion of a public/private key-pair *over untrusted network infrastructure* without revealing sensitive information to a third party (see Section 3.2.3 for more information). Based on these credentials, a communication end-point then can securely authenticate the owner of the corresponding private key.

Moreover, by further augmenting the public key with additional information such as a domain name via certificates, the public key can also be bound to a specific device or network domain. Contrary to symmetric-key cryptography, public-key primitives, therefore, do not require the communication end-points to be in the possession of pre-shared keying material *prior* to the actual protocol handshake. In fact, as noted in Section 3.2.2, public-key cryptography commonly is used in network security protocols to establish the keying material that is required for the efficient protection of handshake messages and application data via symmetric-key primitives.

The important role of public-key cryptography also manifests in the design of the DTLS, HIP DEX, and Minimal IKEv2 protocols. As discussed in Section 3.3.6, their protocol specifications all mandate or recommend the use of public-key primitives. However, we identify several performance and security issues that are directly attributable to these public-key operations during the protocol handshake. Most importantly, we find that public-key operations significantly hamper the availability and response time of constrained devices. The design of DTLS, HIP DEX, and Minimal IKEv2, therefore, must be tailored to reduce the need for these expensive handshake operations and to account for the high processing time of the individual handshake messages in order to fully adapt these protocols to IoT requirements.

4.1.1 Impact on the Protocol Handshake

We now discuss the identified protocol performance and security issues with respect to the computation requirements of the DTLS, HIP DEX, and Minimal IKEv2 protocols. We quantify their impact for HIP DEX and refer to Section 4.1.6 for detailed information about the evaluation setup. To structure our discussion, we distinguish between the following three aspects: (i) the choice and strength of the employed cryptographic primitives, (ii) the consideration of the high resource asymmetry in the

ECDSA (sign)	ECDSA (verify)	ECDH (generate)	ECDH (compute)	AES	SHA-256
399.38 ms	1010.70 ms	311.17 ms	656.25 ms	0.23 ms	10.49 ms

Table 4.1 Comparison of the processing time for a single block operation of various cryptographic primitives. The results denote the average over 100 measurements on the Zolertia Z1 platform. AES operations were performed with 128 bit keys in hardware. All other operations were based on the *relic toolkit* [relic]. Concerning ECDSA and ECDH, we employed elliptic curve SECP160R1. Notably, the overheads of ECDSA and ECDH are prone to grow even further as NIST currently recommends elliptic curves with a field size of at least 224 bit [NIST12a].

specified DoS protection mechanisms, and (iii) the adaptability of the retransmission strategies to the varying processing times of the different handshake messages.

4.1.1.1 Choice and Strength of the Cryptographic Primitives

As shown in Table 4.1, the computation overhead caused by public-key-based primitives on constrained devices dwarfs the overheads of symmetric-key-based primitives as well as cryptographic hash functions. The HIP DEX protocol therefore forfeits the use of most public-key primitives in its protocol design. These public-key primitives include public-key signatures and the ephemeral ECDH key agreement. Instead, HIP DEX employs a *static* ECDH key agreement for both peer authentication and key agreement purposes (see Section 3.3.4). As a result, the HIP DEX handshake only involves a single ECDH computation per communication end-point. This stands in stark contrast to DTLS, especially when considering recent cipher suite recommendations for the DTLS-based protection of the CoAP protocol[1]. Here, current cipher suite recommendations for public-key cryptography require *each communication end-point* to perform two ECDSA operations for signature generation and verification and two ECDH operations for the generation of an ephemeral ECDH key-pair and the actual ECDH key agreement. Similar to DTLS, the Minimal IKEv2 protocol handshake also involves the computation overhead of an ephemeral ECDH key agreement, but forfeits the use of public-key signatures (see Section 3.3.6).

To illustrate the computation overhead on a constrained device that stems from this use of public-key primitives during the protocol handshake, we now highlight the resulting computation times for the elliptic curve with the smallest field size supported by HIP DEX, i.e., SECP160R1. For this curve, the HIP DEX handshake incurs a public-key-related computation overhead of about 1.31 s for both communication end-points combined (see fourth column in Table 4.1). This overhead increases to 1.93 s for Minimal IKEv2 and to 4.76 s for DTLS. Importantly, these substantial overheads are about to grow even further as protocol implementations migrate to elliptic curves with larger field sizes to adopt higher levels of security. Such a migration is highly advisable as the National Institute of Standards and Technology (NIST) currently recommends curves with a field size of at least 224 bit for the protection

---

[1] The CoAP protocol specification recommends the use of an ephemeral ECDH key agreement, ECDSA signatures, and the AES block cipher with the Counter with CBC-MAC (CCM) mode of operation, with 128 bit keys, and with 8 byte authentication tags (i.e., cipher suite TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8) [SHB14] during the DTLS protocol handshake.

of sensitive information [NIST12a]. NIST P-256 is the elliptic curve with the smallest field size that satisfies this recommendation and that also is supported by the HIP DEX protocol. With this curve, already the static ECDH key agreement of HIP DEX, however, requires about 1.86 s *per communication end-point*.

Identified protocol issue:

As shown above, public-key operations block the MCU of constrained devices for extended periods of time during the protocol handshake. This, however, inevitably leads to situations where these devices cannot fulfill their original tasks, e.g., the processing of sensed information or packet forwarding, in a reliable manner due to their intermittent availability. In addition, sleep deprivation considerably decreases the lifetime of energy-constrained devices if they have to perform expensive cryptographic operations on a regular basis. These adverse effects of public-key cryptography are even further aggravated when taking current security recommendations and increased levels of security into account. Hence, to preserve the scarce resources of constrained devices for their original tasks, it is important to reduce the amount of public-key operations during the DTLS, HIP DEX, or Minimal IKEv2 handshake.

4.1.1.2 DoS Protection and the Resource Asymmetry in the IoT

The high resource asymmetry in the IoT enables a even *single* adversary such as a conventional Internet host to mount a flooding-based DoS attack against a constrained device. To amplify this attack, the adversary can exploit asymmetries in the design of the DTLS, HIP DEX, or Minimal IKEv2 handshake. For example, the adversary may initiate multiple handshakes in parallel, thus exploiting the expensive public-key operations during the protocol handshake with the goal to exhaust the target's scarce computation resources. Similarly, the adversary may initiate multiple handshakes in short succession to continually occupy these scarce resources.

DTLS and HIP DEX both employ protection mechanisms that aim at countering such DoS attacks against the protocol handshake. The DTLS protocol design integrates a *cookie-based return-routability test* that requires the Initiator of a protocol handshake to echo back a Responder-defined nonce before the Responder invests computation or memory resources into the continuation of the handshake. This allows to identify spoofed handshake messages and to blacklist communication end-points that behave maliciously if they can uniquely be identified by their IP address. Return-routability tests, however, often do not suffice to protect against DoS attacks for IPv6-based network communication as typically encountered in the IP-based IoT (see Section 2.3.1). This is because the address assignment with IPv6 for an individual site is based on a large address space of at least 64 bit, even for small-scale home networks [NHR11]. As a result, a single IPv6 host can be equipped with multiple globally routable addresses or can change its address over time, e.g., for privacy reasons [dVPC$^+$08]. An adversary can exploit this fact to pretend to represent multiple end-points, thus thwarting the blacklisting capabilities of the return-routability test.

In contrast, *puzzle-based DoS protection mechanisms* as, e.g., used by the HIP DEX protocol, require the Initiator of a connection establishment handshake to solve a Responder-defined cryptographic challenge. This enables the Responder to demand an *adjustable* resource commitment from the Initiator before performing expensive handshake operations. Specifically, as DoS protection is no longer based on virtual information that can be tampered with but instead relies on the actual physical resources of the adversary, the Responder can protect against DoS attacks from a single adversary by rendering the connection establishment expensive.

Identified protocol issue:

As discussed above, mere return-routability tests often do not suffice to thoroughly protect constrained devices against DoS attacks that target the protocol handshake. Puzzle-based DoS protection mechanisms, however, invariably impact legitimate Initiators that aim at establishing a secure connection with a Responder who is currently under attack. More precisely, the Responder is required to demand the same resource commitments from both legitimate and malicious Initiators as it cannot distinguish between them during the protocol handshake. While puzzles with a high difficulty may eventually be solved by unconstrained legitimate Initiators, these puzzle, however, quickly become impractical for constrained Initiators. As a result, handshakes with resource-constrained Initiators are prone to fail. Vice versa, low puzzle difficulties account for resource-constrained Initiators but do not protect the Responder against DoS attacks mounted by an unconstrained adversary. Thus, DoS protection in the IoT must require a small resource commitment from constrained devices, while demanding a high resource commitment from unconstrained communication end-points. Cipher suites and other HIP DEX protocol information, however, do not carry sufficient semantic meaning to make an informed decision about the Initiator's available resources and consequently an appropriate puzzle difficulty. This renders the protection against DoS attacks in the IoT challenging.

4.1.1.3 Retransmissions and Varying Message Processing Times

The DTLS, HIP DEX, and Minimal IKEv2 specifications all define retransmission strategies to handle lost handshake messages. These strategies aim for the opposing goals of minimizing premature retransmissions as well as reducing the handshake delay that is caused by the loss of a handshake message. Notably, premature retransmissions are highly undesirable in constrained node networks as they waste scarce energy resources on the forwarding path and needlessly occupy the wireless medium. The processing time of individual handshake messages, however, varies significantly depending on the triggered handshake operations and the device that performs these operations. For example, our evaluation shows that the processing times of the I1 and I2 messages differ by about 0.7 s on a constrained device. Similarly, a static ECDH key agreement on a desktop-class communication end-point takes less than 0.02 s, whereas the same operation requires about 0.66 s on a constrained device. Consequently, response messages, which also signal message reception, may exhibit delays that exceed purely network delay-based retransmission timeouts.

Identified protocol issue:

The retransmission mechanisms specified for DTLS, HIP DEX, and Minimal IKEv2 predominantly employ *fixed, message-independent retransmission strategies* that do not account for the computation time of public-key operations. HIP DEX, for instance, specifies an aggressive retransmission strategy that triggers retransmissions in the order of a few milliseconds. Yet, public-key operations may take seconds to complete on a constrained device. As a result, overly aggressive retransmission strategies inevitably cause premature retransmissions despite the successful reception of a given message at the handshake peer. The DTLS specification, in turn, recommends a retransmission timeout of 1 s that the communication end-point should double for each repeated retransmission of the same message flight. However, while this strategy causes less premature retransmissions, it quickly delays the conclusion of a handshake for a significant amount of time if consecutive retransmissions are lost. Moreover, it is important to note that public-key operations based on elliptic curves such as NIST P-256 cause computation times on constrained devices that are well above 1 s. Thus, the retransmission mechanism defined for DTLS also provokes premature retransmissions when following current security recommendations. In contrast, the Minimal IKEv2 specification does not recommend specific timeout values for retransmission purposes. Still, it presumes retransmissions to only employ local knowledge. Consequently, these existing retransmission strategies are unable to take the message processing time at the handshake peer into account and, thus, are prone to cause premature retransmissions in the context of constrained devices.

We now continue with the presentation of our three complementary protocol extensions that alleviate the identified protocol issues. We first introduce a flexible session resumption mechanism that allows to amortize the expensive public-key operations of an initial protocol handshake across repeated connection establishments. We then show how to extend the puzzle-based DoS protection mechanism of HIP DEX for collaborative protection in network scenarios with high resource asymmetries. Finally, we present an adaptive retransmission mechanism that accounts for the varying processing times of the different handshake messages on constrained devices.

4.1.2 Reducing the Cost of Repeated Connection Establishments

The high computation overhead of public-key operations during the DTLS, HIP DEX, and Minimal IKEv2 handshakes leads to the inclination of using an established connection over extended periods of time. To this end, a constrained device may, e.g., set up *long-lived* connections during its bootstrapping phase (see Section 3.3.2). These connections then are anticipated to last for the remaining device lifetime.

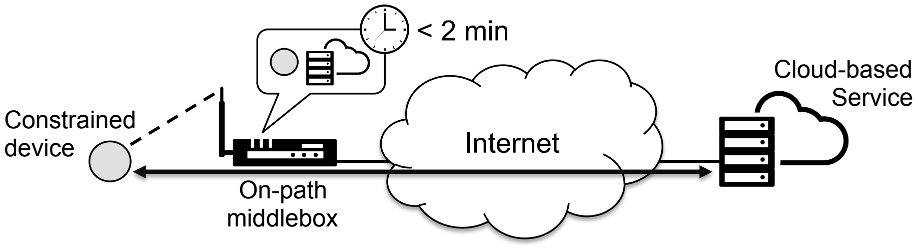


Figure 4.1 Network scenario with an on-path middlebox that employs a soft-state approach for maintaining information about active connections. The arrow indicates periodic keep-alive messages sent by the communication end-points to keep the connection open.

Such an approach, however, has several drawbacks that often necessitate *repeated connection establishment handshakes.* First, the cryptographic keys that are used for the protection of bulk data must occasionally be refreshed to prevent overuse and degradation of the security properties of the underlying symmetric-key algorithm [BH05]. Second, on-path middleboxes such as firewalls typically employ soft-state approaches to store information about traversing data flows [Woo11] (see Figure 4.1). As a result, communication end-points must transmit periodic keep-alive messages to ensure uninterrupted forwarding of bidirectional data flows. These keep-alive messages must be sent at least once every two minutes according to recent recommendations for on-path middleboxes [Woo11]. Thus, the transmission overhead required for keeping a connection alive eventually exceeds the overhead for a new connection establishment with respect to the resulting energy expenditure. This is especially true for application scenarios that only involve sporadic data transmissions. Third, constrained devices may only be able to maintain very few connections in parallel due to the memory requirements of the corresponding session states. As a result, constrained devices may need to discard session state for completed requests and perform a new protocol handshake for repeated interactions with prior communication partners if the available memory space for session state is fully occupied.

4.1.2.1 Handling Repeated Handshakes with Session Resumption

Our main goal in this section is to reduce the computation overhead caused by such repeated connection establishment handshakes. To this end, we introduce a novel session resumption mechanism for HIP DEX that is inspired by similar mechanisms for TLS [DR08] and IKEv2 [ST10]. The basic idea behind session resumption is to enable the handshake peers to reuse their session state from an *initial connection establishment handshake* with expensive public-key operations *across* subsequent handshakes. The handshake peers, therefore, keep their established session state even after the connection is torn down. The peers then efficiently re-authenticate each other and re-establish the previous session context based on their stored session state in an *abbreviated session resumption handshake.* Moreover, a variant of this basic session resumption type additionally allows the Responder to securely offload its session state to the Initiator for safe-keeping purposes [SZET08, ST10]. This enables the Responder to remain stateless while the connection is inactive.

We observe that such *Responder-side state-offloading* is motivated by the fact that traditional client-server systems commonly involve a small number of powerful servers that handle a multitude of clients. Hence, Responder-side state-offloading improves the scalability of the overall system by unburdening the Responder at the cost of its connected Initiators. In the IoT, this mindset, however, puts resource-constrained Initiators at a considerable disadvantage. For example, in case of a connection between a constrained Initiator and a powerful Responder such as a Cloud-based service, offloading the Responder's session state to the Initiator further draws on the Initiator's scarce memory resources instead of unburdening them. Moreover, the role of a constrained device typically is not pre-determined in M2M scenarios, where this device can act as the Initiator *or* the Responder during a connection establishment handshake. State-offloading, therefore, must be based on the available resources at the handshake peers instead of their role during a given handshake.

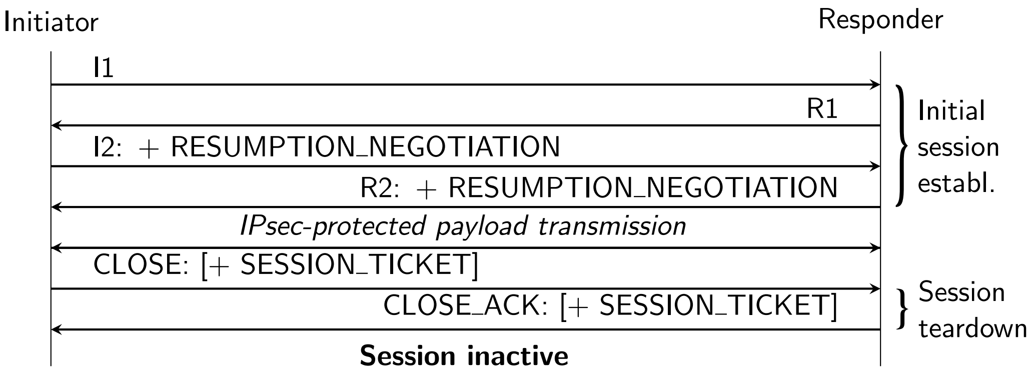

Initiator
Responder
I1
R1
I2: + RESUMPTION_NEGOTIATION
R2: + RESUMPTION_NEGOTIATION
IPsec-protected payload transmission
CLOSE: [+ SESSION_TICKET]
CLOSE_ACK: [+ SESSION_TICKET]
Session inactive
Initial session establ.
Session teardown

Figure 4.2 Initial HIP DEX connection establishment with session resumption type negotiation. The session state can be offloaded during the connection teardown. Brackets denote optional parameters that are included depending on the actual session resumption type.

To address the above observations, we now present a the design of our flexible session resumption mechanism that features an additional session resumption type compared to the existing session resumption mechanisms. This additional session resumption type enables a resource-constrained Initiator to securely offload its session state to the Responder. Moreover, we introduce an *explicit* session resumption type negotiation mechanism that enables the communication end-points to agree on a mutually preferred session resumption type for the next protocol handshake.

4.1.2.2 Initial connection Establishment

As shown in Figure 4.2, the Initiator and the Responder leverage the initial HIP DEX handshake to negotiate the mutually preferred session resumption type for subsequent connection establishments. Our flexible session resumption mechanisms thereby differentiates between the following three session resumption types: (i) *state compression*, where both communication end-points store their own session state across connections, (ii) *Initiator-side state-offloading*, where the Responder stores the session state on behalf of the Initiator, and (iii) *Responder-side state-offloading*, where the Initiator stores the session state on behalf of the Responder.

Concerning the actual message exchange, the Initiator signals its preferred session resumption types to the Responder in the I2 handshake message. To this end, the Initiator includes a RESUMPTION_NEGOTIATION parameter that lists its supported session resumption types in their order of preference. This order can be based on the Initiator's memory resources at the time when the handshake takes place or can be configured statically. Static configuration thereby especially applies for highly memory-constrained devices and unconstrained communication end-points. Specifically, while the former preferably offload their session state, we expect the latter to graciously accept storing session state on behalf of their communication partners.

After receiving and processing the I2 message, the Responder selects and indicates the negotiated session resumption type in the RESUMPTION_NEGOTIATION parameter of the R2 message. In this, the Responder takes its own as well as the Initiator's preferences into account. Importantly, *no* session resumption is performed during the subsequent handshakes if at least one communication end-point omits the negotiation parameter during the initial connection establishment. In this case, the initial handshake concludes without modification to the standard HIP DEX protocol specification. This specific design trait allows to probe the communication partner for session resumption support and to perform a standard HIP DEX handshake with *legacy end-points* that are unaware of our new protocol extension.

When the handshake phase completed successfully, the end-points communicate securely via an IPsec-protected payload channel, i.e., equal to the standard HIP DEX protocol. The connection teardown exchange then deviates again from the standard protocol behavior depending on the negotiated session resumption type. This is because the Initiator or the Responder use the connection teardown to offload their protected session states (see Figure 4.2). Specifically, if both end-points previously agreed on Initiator-side state-offloading, the Initiator compresses and encrypts its active session state when triggering the connection teardown exchange. The Initiator then includes this state in the SESSION_TICKET parameter of the CLOSE message and discards its active session state. On receipt of the CLOSE message, the Responder compresses its own session state and stores it along with the received session state from the Initiator. In contrast, with Responder-side state-offloading, the CLOSE message remains unaltered. Instead, the Responder includes its session state in the SESSION_TICKET parameter of the CLOSE_ACK message. As a third option, both end-points may also agree on session resumption with state compression. In this case, no session tickets are exchanged during connection teardown and both end-points store their own compressed states while the session is inactive.

4.1.2.3 Remarks concerning the Delayed State-Offloading

We note that we intentionally delay state-offloading until the end of the connection lifecycle in order to relieve the communication end-points from storing a peer's offloaded session state while the session is still active. This design decision enables even constrained devices to store a limited amount of session state for constrained communication partners as the compressed and the offloaded session states combined are smaller than the state of an active session (see Section 4.1.6.1). Still, an end-point may temporarily loose connectivity, thus preventing the communication partners from performing the negotiated state-offloading during the connection teardown exchange. In case of an intermittent connection failure, the end-points, therefore, always employ session resumption with state compression independent from the negotiated session resumption type in order to efficiently re-establish and re-synchronize the session context without the need for public-key cryptography.

4.1.2.4 Abbreviated Session Resumption Handshake

The abbreviated session resumption handshake enables the communication end-points to leverage the stored session state from a previous connection to efficiently re-authenticate each other and to establish a fresh session key for payload protection. The abbreviated handshake thereby differs depending on which end-point stores the session states and which end-point triggers the connection re-establishment. We now first introduce the abbreviated handshake with state compression. The two handshakes involving state-offloading then constitute variants of this basic handshake.

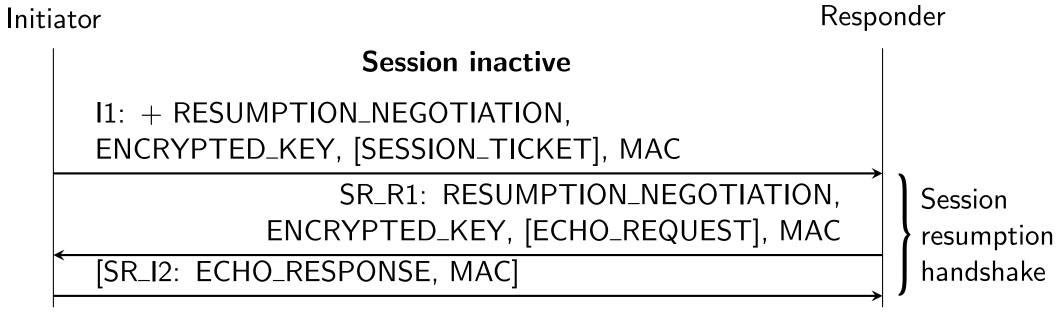

Initiator
Responder
Session inactive
I1: + RESUMPTION_NEGOTIATION,
ENCRYPTED_KEY, [SESSION_TICKET], MAC
SR_R1: RESUMPTION_NEGOTIATION,
ENCRYPTED_KEY, [ECHO_REQUEST], MAC
[SR_I2: ECHO_RESPONSE, MAC]
Session
resumption
handshake

Figure 4.3 The abbreviated session resumption handshake for HIP DEX. Brackets denote optional messages and parameters that depend on the actual session resumption type.

The abbreviated handshake with *state compression* consists of the I1 and SR_R1 messages as well as their mandatory parameters depicted in Figure 4.3. The I1 message most notably contains a RESUMPTION_NEGOTIATION parameter. This parameter allows the end-points to negotiate the session resumption type for the next session resumption handshake and indicates the current session resumption type to the Responder. Moreover, this message includes a fresh encrypted key share that is part of a new session key for the subsequent protection of application data (see Section 3.3.4). Finally, the I1 message contains a MAC parameter that protects the message integrity and allows the Responder to authenticate the Initiator.

When receiving the I1 message, the Responder inspects the included session resumption type negotiation parameter and looks up its own compressed session state. The Responder then uses this state to authenticate the Initiator and to verify the integrity of the I1 message. To this end, the Responder leverages the fact that the Initiator is the only other party in possession of the necessary keying material that allows to compute a correctly verifiable MAC value. Specifically, both end-points previously derived this keying material during the initial ECDH key agreement and store it as part of their compressed session states while the session is inactive. In case of a successful verification, the Responder then decrypts the received Initiator-side key share and concludes the session resumption handshake with a new SR_R1 message. This message confirms the subsequent session resumption type in the RESUMPTION_NEGOTIATION parameter and includes the Responder-side key share in the ENCRYPTED_KEY parameter. Combined, both key shares build the basis for a new session key. Hence, each session resumption handshake also includes a rekeying operation that updates the session key for the protection of application data. Moreover, the SR_R1 message contains a MAC parameter that enables the Initiator to authenticate the Responder and to verify the SR_R1 message integrity.

We note that we introduce the dedicated SR_R1 message type for the abbreviated session resumption handshake in order to further reduce the transmission overhead for repeated connection establishments. This is because the specification of the standard R1 message contains mandatory parameters that are not required in the abbreviated session resumption handshake, e.g., the Responder's public ECDH key.

4.1.2.5 Session Resumption with State Offloading

Compared to the basic abbreviated handshake described above, *Responder-side state-offloading* also requires the Initiator to include the Responder's offloaded session state in the SESSION_TICKET parameter of the I1 message. Upon reception, the Responder verifies the received session state as discussed in the next section and uses the included keying material to authenticate the Initiator and to verify the integrity of the received message. The Responder then generates an SR_R1 response with an ECHO_REQUEST parameter. This parameter allows the Responder to determine the freshness of the abbreviated handshake by requiring the Initiator to echo back a Responder-defined challenge value in the ECHO_RESPONSE parameter of the new SR_I2 message. The successful verification of the MAC value as well as of the re-included challenge value in the received SR_I2 message then assures the Responder that the handshake is not a replay of a previously recorded session resumption handshake. This assurance is based on the fact that only the legitimate Initiator can compute correct MAC values for messages with Responder-defined content.

Finally, in case of session resumption with *Initiator-side state-offloading*, the Initiator sends a standard I1 message to the Responder as it is unaware of the previously established session state. The Responder then looks up its own compressed and the Initiator's offloaded session states. For look-up purposes, the Responder leverages the fixed-length representation of the sender's public ECDH key, i.e., the Host Identity Tag (HIT), that is included in the header of all HIP DEX protocol messages. In case of a successful session state look-up, the Responder then takes over the role of the Initiator for the subsequent handshake messages and sends an I1 message that contains the same parameters as the I1 message in the session resumption handshake with Responder-side state-offloading. When receiving this message, the original Initiator also changes its role and the handshake concludes as described above.

4.1.2.6 Required Session State Information

While the HIP DEX session state is implementation and scenario-specific, it must meet a few requirements to ensure the secure operation of the introduced session resumption mechanism. Most importantly, it has to include a locally unique identifier of the communication partner. Without such an identifier, an adversary, who previously established session state with the target device, could impersonate another end-point during a subsequent abbreviated session resumption handshake, thus potentially escalating its access privileges at the target device. This privilege escalation attack is possible because, in contrast to the public-key-based peer authentication during the standard HIP DEX handshake, the communication end-points re-authenticate each other in the session resumption handshake based on their common knowledge of the previously established session state. This session state, however, is not bound to a specific end-point identity per se. Hence, to protect against this attack, we require the HIT of the state-receiving peer to be part of the stored or the offloaded session state. This inclusion of the peer's HIT explicitly binds the maintained session state to a specific HIP DEX connection. In addition, the session state must include the derived ECDH keying material and the negotiated cipher suites of the initial connection establishment in order to be able to compute valid MAC values and to correctly encrypt the key shares during the abbreviated handshake.

In case of state offloading, the transferred session state first must be encrypted as well as integrity protected and the corresponding session-state protection key must only be known to the offloading communication end-point. Otherwise, an adversary could modify existing or create forged offloaded session state. As a result, the adversary could impersonate other communication end-points towards the offloading end-point by pretending to resume a previously established connection. To protect state offloading against such attacks, we recommend the use of AES in combination with the CCM mode of operation [WHF03] for the protection of offloaded session state. This authenticated encryption mode allows to leverage potential AES hardware support on constrained devices. Furthermore, this mode also requires the offloading communication end-point to only store a single cryptographic key for both the encryption and the integrity protection of its offloaded session state.

Finally, offloaded session state may need to be revoked, e.g., when the communication partner is compromised. In a naïve approach, such revocation can be achieved by simply changing the session-state protection key at the offloading end-point. Revocation then, however, also implies that all other inactive connections have to be re-established with a standard protocol handshake. We refer to Section 5.2.3.3 for a more efficient approach to handle the revocation of individual session state.

4.1.2.7 Integration with DTLS and Minimal IKEv2

The existing session resumption mechanisms for DTLS and Minimal IKEv2 currently lack the possibility for a resource-constrained Initiator to offload its session state to the Responder for safe-keeping purposes. Similarly, these approaches do not provide the necessary protocol functionality to explicitly negotiate the mutually preferred session resumption type. Hence, we propose to integrate these two additional aspects of our approach with the existing mechanisms for DTLS and Minimal IKEv2.

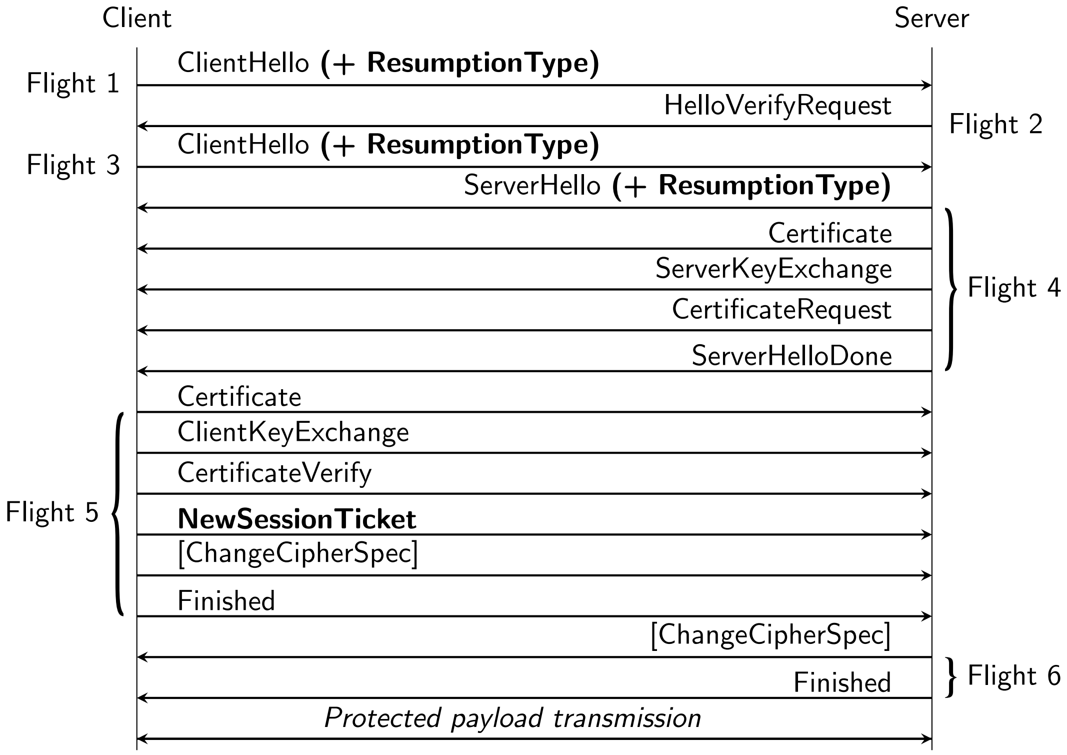


Figure 4.4 Session resumption type negotiation and client-side state-offloading during the initial connection establishment handshake of the DTLS protocol. Session resumption-related information is marked bold. Brackets denote DTLS protocol extensions.

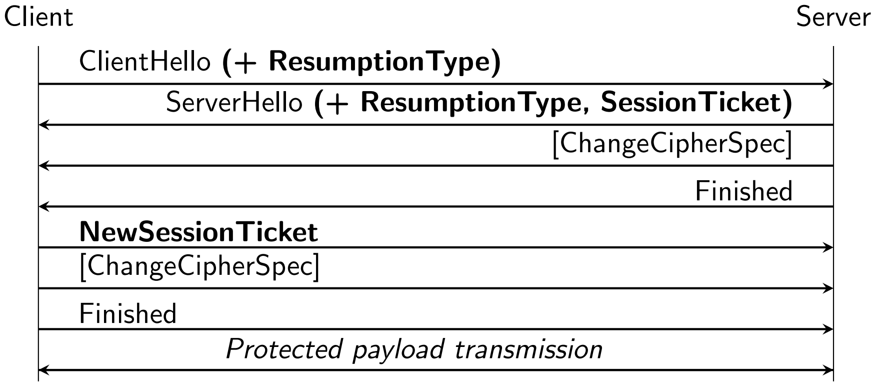

Client
Server
ClientHello (+ ResumptionType)
ServerHello (+ ResumptionType, SessionTicket)
[ChangeCipherSpec]
Finished
NewSessionTicket
[ChangeCipherSpec]
Finished
Protected payload transmission

Figure 4.5 Abbreviated handshake with client-side state-offloading for DTLS. Session resumption-related information is marked bold. Brackets denote DTLS protocol extensions.

For DTLS, session resumption type negotiation can be realized as an extension of the ClientHello and the ServerHello handshake messages (see Figure 4.4). This enables the handshake peers to explicitly agree on one of the existing session resumption types, i.e., state compression or server-side state-offloading, as well as on our novel client-side state-offloading. If the peers agree on one of the existing types, the handshake concludes as specified in the corresponding protocol specification [RM12, SZET08]. In contrast, for client-side state-offloading, we propose that the client transfers its protected session state to the server via the NewSessionTicket message as illustrated in Figure 4.4. To resume the session, the server then sends the offloaded session state back to the client. To this end, it uses the SessionTicket extension of the ServerHello message in a newly introduced abbreviated DTLS session resumption handshake (see Figure 4.5). For more detailed information about this new handshake and about our above session resumption extensions, we refer to the proposed protocol specifications in our corresponding Internet-Draft at the IETF [HGS13].

In case of Minimal IKEv2, session resumption type negotiation can be implemented in the first two messages of the handshake, i.e., the IKE_SA_INIT exchange, as depicted in Figure 4.6. Moreover, Initiator-side state-offloading can be realized based

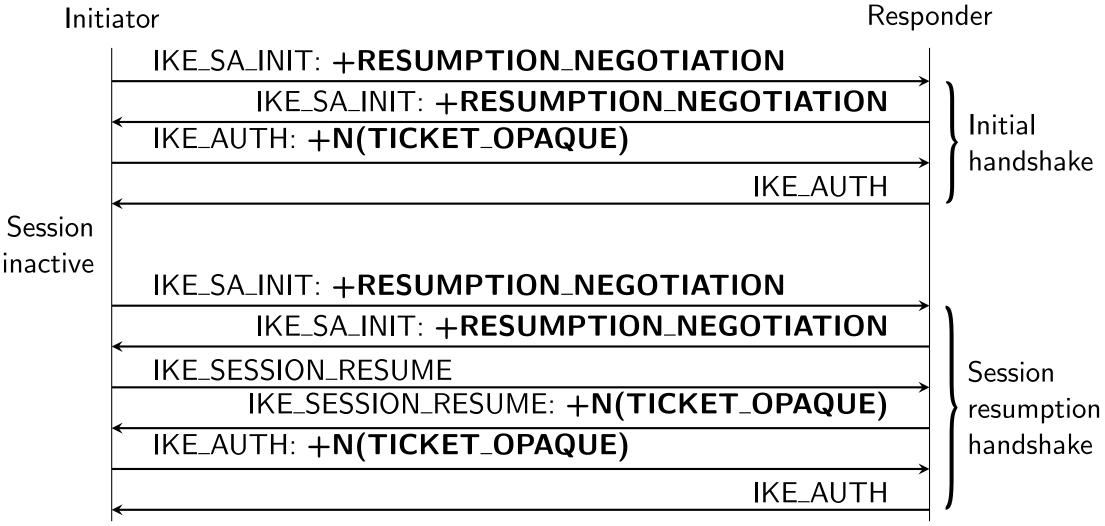

Initiator
Responder
IKE_SA_INIT: +RESUMPTION_NEGOTIATION
IKE_SA_INIT: +RESUMPTION_NEGOTIATION
IKE_AUTH: +N(TICKET_OPAQUE)
IKE_AUTH
Initial handshake
Session inactive
IKE_SA_INIT: +RESUMPTION_NEGOTIATION
IKE_SA_INIT: +RESUMPTION_NEGOTIATION
IKE_SESSION_RESUME
IKE_SESSION_RESUME: +N(TICKET_OPAQUE)
IKE_AUTH: +N(TICKET_OPAQUE)
IKE_AUTH
Session resumption handshake

Figure 4.6 Sequence diagram of the session resumption type negotiation and Initiator-side state-offloading functionalities for Minimal IKEv2. N(·) denotes an IKEv2 notification payload.

on the IKE_SESSION_RESUME exchange and the TICKET_OPAQUE notification payload of the existing IKEv2 session resumption mechanism. Specifically, we propose that the Initiator offloads its protected session state to the Responder in the ticket notification payload in the first message of the initial IKE_AUTH exchange (see Figure 4.6). When resuming a session, the stateless Initiator then first sends a regular IKE_SA_INIT message as it is unaware of its offloaded session state. This message also contains a session resumption type negotiation parameter that informs the Responder about the Initiator's supported session resumption types. The Responder's reply also includes a negotiation parameter notifying the Initiator about the stored session state. As a result, the Initiator starts the existing IKE_SESSION_RESUME exchange and the Responder transfers the session state in the corresponding response message as shown Figure 4.6. Moreover, the Initiator offloads its session state in the IKE_AUTH exchange for use during the next session resumption handshake. We note that, besides these changes, the existing IKEv2 session resumption mechanism is unaffected by our newly introduced session resumption functionality.

4.1.3 DoS Protection Despite Resource Asymmetries

In Section 4.1.2, we showed how our flexible session resumption mechanism unburdens constrained devices from the expensive public-key-related protocol operations during repeated connection establishments. More precisely, the presented mechanism enables constrained devices to purely rely on efficient symmetric-key cryptography during a session resumption handshake. This circumstance significantly reduces amplification effects that an adversary can exploit in a DoS attack against *repeated* connection establishments. The *initial* connection establishment as well as the standard HIP DEX handshake, however, still involve computationally expensive public-key operations. These operations enable even a single unconstrained adversary to mount a DoS attack against a constrained device (see Section 4.1.1.2). Hence, to protect constrained devices against such DoS attacks, we now introduce a *collaborative puzzle-based DoS protection mechanism* that is tailored to the device constraints in the embedded domain and the high resource asymmetry in the IoT.

4.1.3.1 DoS Attacks Against the HIP DEX Protocol Handshake

As a first step towards developing this tailored DoS protection mechanism, we observe that DoS attacks may target two primary operations during the standard HIP DEX protocol handshake. On the one hand, an adversary may target the expensive public-key operations with the goal to maliciously occupy the scarce computation resources of a constrained device. On the other hand, the adversary may target the device's state creation process, thereby aiming to exhaust its scarce memory resources [AN97]. As illustrated in Figure 4.7, the design of the HIP DEX protocol already mitigates the latter attack by allowing the Responder to remain stateless until the Initiator's identity is confirmed during the I2 message processing. This enables the Responder to only establish session state for successfully *authenticated and authorized* communication end-points. Likewise, the Initiator's state creation process is commonly not considered a viable target for a DoS attack as the Initiator only creates session state when *actively* triggering a handshake (see Figure 4.7).

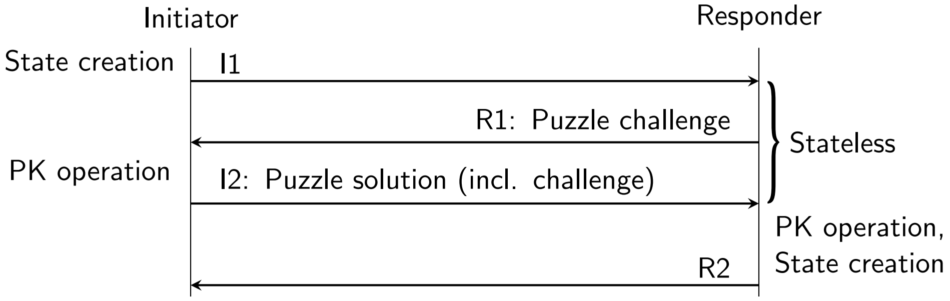


Figure 4.7 The standard HIP DEX handshake with a specific focus on the expensive protocol operations and the existing DoS protection mechanisms. PK denotes a public-key operation.

Hence, an adversary would need to force the Initiator into *unwillingly* establishing multiple new HIP DEX connections in order to exhaust its memory resources.

An adversary, however, would still be able to attack the peer authentication process that is performed during the HIP DEX handshake. This is because the peer authentication process employs a computationally expensive public-key operation. Moreover, this expensive operation is imperative for the verification of the peer's authenticity and, thus, its authorization to communicate with the constrained device.

We note that the Initiator can effectively prevent DoS attacks against its peer authentication process by simply limiting the number of concurrently triggered protocol handshakes. Likewise, the Initiator can drop *unsolicited* R1 messages based on the lack of associated session state (see Figure 4.7). The Responder, however, remains stateless until the Initiator has been authenticated successfully. As a result, the Responder has to process all received Initiator-side handshake messages prior to the creation of the associated session state. This, however, also includes I2 messages that trigger an expensive peer authentication process at the Responder.

To protect the Responder against DoS attacks that exploit this circumstance, the HIP DEX protocol employs a puzzle-based DoS protection mechanism that previously has similarly been used, e.g., to defend against spam e-mail [DN93] or TCP SYN flooding attacks [JB99]. The key idea behind this puzzle mechanism is to enable the Responder to demand an adjustable resource commitment from the Initiator *before* performing expensive protocol operations during I2 message processing. By default, this resource commitment should be insignificant to prevent the Initiator from unnecessarily churning its resources during normal operation. Yet, once the Responder suspects to be under attack, the resource commitment must incur non-negligible costs at the adversary in order to frustrate a potential DoS attack. At the same time, such non-negligible resource commitments, however, should still allow for a limited number of legitimate handshakes to conclude successfully. Thus, DoS protection in the IoT must require a small resource commitment from constrained devices, while demanding a high resource commitment from unconstrained communication end-points. The required detection of a DoS attack against the protocol handshake and the adjustment of the resource commitment depending on the Initiator's computational resources, however, denote open issues of the puzzle mechanism.

We now address these missing building blocks by introducing two complementary, low-overhead puzzle extensions. Specifically, we first present an *attack detection mechanism* that allows the Responder to follow a default-off policy for the puzzle

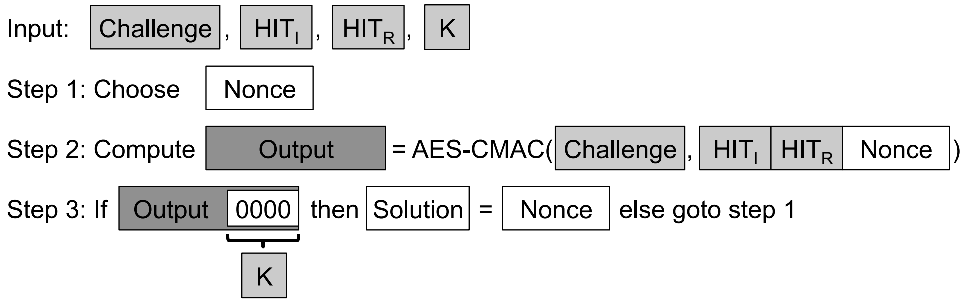

Input: Challenge , HIT_I , HIT_R , K
Step 1: Choose Nonce
Step 2: Compute Output = AES-CMAC(Challenge , HIT_I HIT_R Nonce)
Step 3: If Output 0000 then Solution = Nonce else goto step 1
K

Figure 4.8 Abstract algorithm that the Initiator employs to solve a Responder-issued puzzle challenge with difficulty K. HIT_I and HIT_R denote the Initiator's and Responder's HITs.

mechanism. We then devise a *collaborative difficulty selection strategy* that enables the Responder to adjust the puzzle difficulty based on additional information from an on-path gateway. In combination, these extensions allow to employ the HIP DEX puzzle mechanism as an effective means against DoS attacks in the IoT.

4.1.3.2 Attack Detection Mechanism

As highlighted above, the HIP DEX protocol design exposes the Responder's computational resources as the primary target of a DoS attack against the protocol handshake. We, therefore, employ the Responder's computational load as the key metric for our attack detection mechanism. More precisely, we use the number of recently performed peer authentications as an estimator for the Responder's computational load. This affords to implement our attack detection via a simple *sliding window mechanism.* The sliding window thereby counts the number of peer authentications during a fixed-length window period of, e.g., 1 min. For each window period, the Responder then graciously accepts a device- and scenario-specific number of protocol handshakes without demanding a resource commitment from the Initiator. We call this number of accepted handshakes the *puzzle issuing threshold.*

More concretely, as long as the puzzle issuing threshold has not been reached for a given window period, the Responder demands a puzzle difficulty of zero in the R1 message. The puzzle mechanism then effectively turns into a return-routability test as, e.g., employed by DTLS. Thus, the Initiator only needs to echo back the Responder's puzzle challenge along with an arbitrary puzzle solution in the I2 message (see Figure 4.7). However, once the puzzle issuing threshold is exceeded, the Responder starts demanding a non-zero puzzle difficulty K from the Initiator. As illustrated in Figure 4.8, this requires the Initiator to find a puzzle solution such that an AES-based CMAC operation over the Responder-defined puzzle challenge, the concatenation of the peers' HITs, and an Initiator-chosen puzzle solution generates an output where at least the K lowest order bits are zero. In other words, the Initiator has to find a pre-image that results in a partial output match of length K. For a hash function[2], this search for a pre-image requires the Initiator to perform about 2^K hash operations[3] due to the underlying pre-image resistance property.

---

[2] The expected work factor of CMAC-based puzzles in HIP DEX is still under active investigation. If it should prove to be too low, the CMAC algorithm could, e.g., be replaced with SHA-256 or an AES-based hash function at the expensive of increased computation and memory overheads.

[3] A pre-image attack against a 128 bit hash digest is assumed to require about 2^{128} hash operations and, thus, is considered infeasible with today's compute power [DK07]. However, finding pre-images becomes increasingly practical when reducing the length of the required output match.

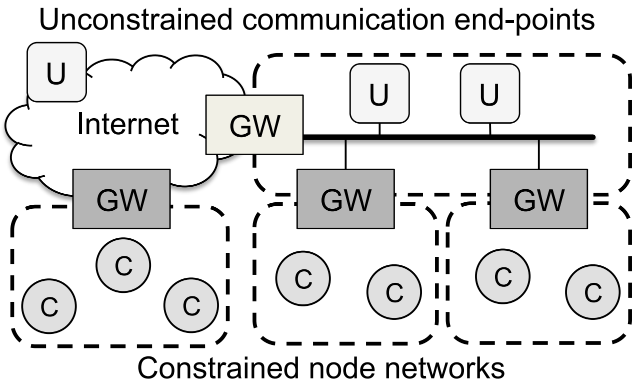

Unconstrained communication end-points
U
Internet
GW
U
U
GW
GW
GW
C
C
C
C
C
C
C
Constrained node networks

Figure 4.9 Example network scenario depicting the interconnection of constrained (C) and unconstrained (U) communication end-points in the IoT. Gateways (GW) that interconnect a constrained node network with adjacent network domains are marked in dark gray.

4.1.3.3 Collaborative Puzzle Difficulty Selection

We recollect that the adjustable puzzle difficulty K allows the Responder to demand a non-negligible resource commitment from the Initiator in case of an attack. The intention thereby is to render the execution of *multiple* protocol handshakes exceedingly expensive for a single adversary, while still allowing a benign Initiator to eventually conclude a legitimate protocol handshake. We note that both properties require the Responder to be aware of the Initiator's computational capabilities. This is because finding a partial output match for a given puzzle difficulty K requires highly differing computation times for constrained and unconstrained Initiators. Cipher suites and other HIP DEX protocol information, however, typically do not convey sufficient semantic meaning for the Responder to determine the Initiator's computational capabilities. Instead, we observe that a gateway that interconnects a constrained node network with adjacent network domains (see Figure 4.9) commonly possesses additional information about the interacting end-points. Specifically, this gateway can identify handshakes as originating from outside the Responder's network domain based on the mere fact that it is located on the forwarding path of the corresponding handshake messages. In addition, the gateway may also be able to further classify these handshakes as originating from other constrained node networks, e.g., based on the included Virtual LAN (VLAN) tags [IEEE11a] or due to authenticated tunnels from remote network domains that terminate at the gateway.

To leverage this additional information at the gateway in our collaborative difficulty selection mechanism, we extend the HIP DEX handshake with a new signaling pa-

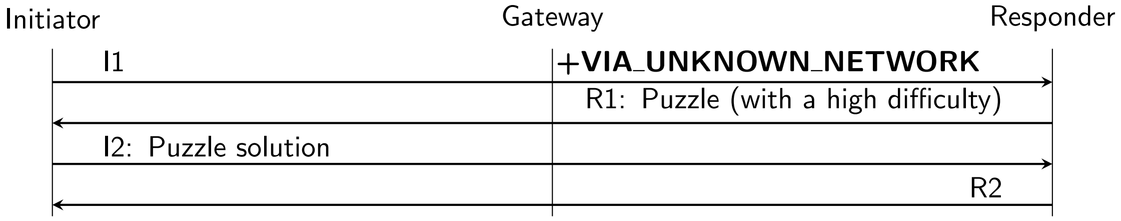


Figure 4.10 The gateway can notify the Responder about an unknown, network-external Initiator by adding the new VIA_UNKNOWN_NETWORK parameter to a traversing I1 message.

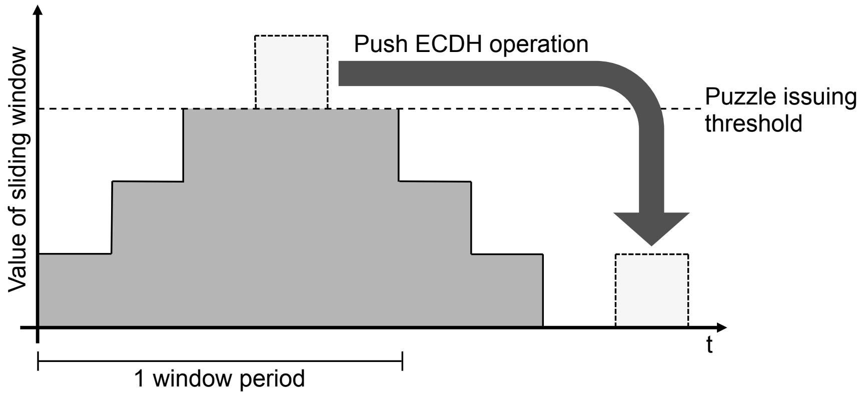


Figure 4.11 By demanding a resource commitment of about one window period, the Responder can push its ECDH operation to a period with potentially less computational load.

rameter. The basic idea behind this parameter is to allow the gateway to inform the Responder about an unknown and, therefore, potentially untrustworthy Initiator. To this end, the gateway adds the new VIA_UNKNOWN_NETWORK parameter to a traversing I1 handshake message (see Figure 4.10). The absence of this new parameter then either indicates a handshake with a co-located constrained device (i.e., no gateway involvement) or a positive identification by the on-path gateway.

The Responder reacts to the absence or the inclusion of the notification parameter from the gateway as follows. If a received I1 message does *not* include the notification parameter, the resource-constrained Responder issues the puzzle based on its own computational resources. More precisely, the Responder sets the puzzle difficulty such that the Responder itself would require about one window period to solve the issued puzzle. In addition to protecting the Responder against other computationally constrained devices, this selection of the puzzle difficulty also allows the Responder to push its public-key-related computational load to a later window period as depicted in Figure 4.11. In contrast, the Responder issues a high puzzle difficulty if the I1 message contains the VIA_UNKNOWN_NETWORK parameter. This high puzzle difficulty similarly should incur a resource commitment of about one window period from an unconstrained adversary. We propose to select this puzzle difficulty based on the computation power of today's high-end CPUs. Moreover, to guarantee that the Responder's puzzle difficulty reflects the continuing increase in computation power, we require this configuration parameter of the difficulty selection strategy to be updated periodically as part of a Responder's maintenance cycle (see Section 3.3.2).

4.1.3.4 Handling Exceedingly High Puzzle Difficulties

The conservatism of the difficulty selection mechanism may lead to situations where the Responder issues an exceedingly high puzzle difficulty for a legitimate Initiator. These situations primarily involve: (i) a resource-constrained Initiator that belongs to an unknown foreign network domain or (ii) a conventional Initiator that is equipped with a low-end or otherwise computationally weak CPU. In either case, the issued puzzles are prone to prevent the Initiator from delivering a puzzle solution in a timely manner. To still afford a successful conclusion of such legitimate handshakes despite an on-going attack, the Responder includes the maximum validity period of the puzzle, which corresponds to the length of a window period, in the

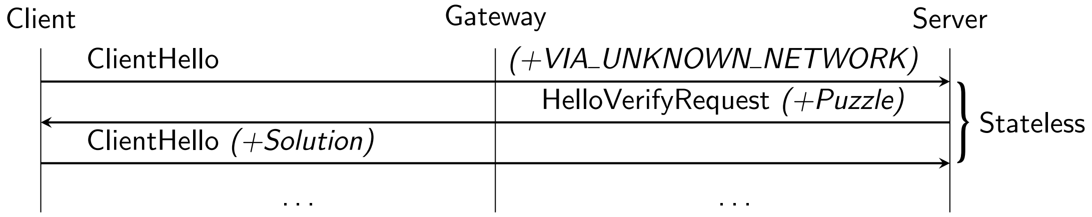


Figure 4.12 First part of the DTLS handshake when replacing the cookie-based return-routability test with our collaborative puzzle-based DoS protection mechanism. Brackets denote new DTLS protocol extensions.

puzzle parameter. The Initiator drops puzzles that it cannot solve within the validity period and requests a new puzzle from the Responder via a repeated I1 message. This repeated I1 message effectively probes the Responder for a window period with low load and, thus, a puzzle difficulty of zero. These window periods typically occur during an attack when the adversary is busy solving a puzzle. We note that this puzzle solving strategy denotes a race with an adversary for a low sliding window value as the adversary may also employ such probing in its DoS attack.

4.1.3.5 Integration with DTLS and Minimal IKEv2

As described in Section 3.3.3, the DTLS protocol provides a cookie-based return-routability test in its protocol design. In addition, it affords the Responder to remain stateless until receiving the echoed cookie value in the second ClientHello message. An adversary, however, may often be able to circumvent this return-routability test as discussed in Section 4.1.1.2. Hence, we propose to improve the DoS protection capabilities of the DTLS protocol by replacing the current cookie-based approach with our collaborative puzzle-based DoS protection mechanism. To this end, an on-path gateway informs the Server about a potentially untrustworthy Client via the new VIA_UNKNOWN_NETWORK parameter in the initial ClientHello message as depicted in Figure 4.12. The Server then issues a cryptographic puzzle according to our descriptions in Section 4.1.3.3. To convey this puzzle, we propose to use the puzzle parameters of a recent standardization proposal that aims at integrating a puzzle-based DoS protection mechanism with TLS [Nir14]. Notably, similar to HIP DEX, this proposal does not define an attack detection or a difficulty selection mechanism. Hence, it effectively complements our work in this section.

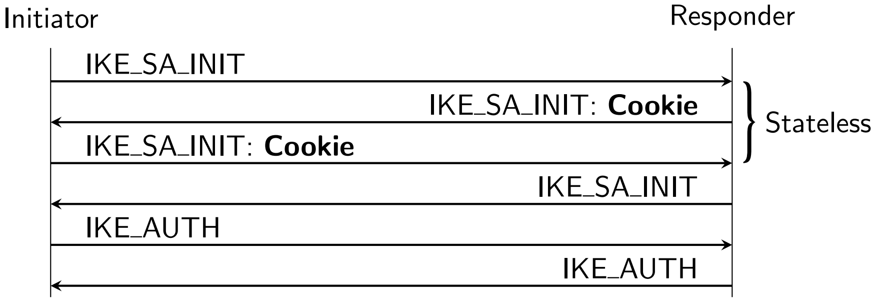


Figure 4.13 The cookie-based DoS protection mechanism of the original IKEv2 protocol. The Responder includes a cookie notification payload in its IKE_SA_INIT response. The Initiator then has to repeat the IKE_SA_INIT exchange and echo back the received cookie value.

In contrast to DTLS and HIP DEX, Minimal IKEv2 currently does not include a DoS protection mechanism in its protocol design. This renders resource-constrained Responders vulnerable to DoS attacks that HIP DEX and DTLS can protect against. Still, we note that the original IKEv2 protocol features a cookie-based DoS protection mechanism. As depicted in Figure 4.13, this mechanism is realized via a repetition of the initial part of the IKEv2 handshake and the inclusion of a Responder-defined cookie value. Moreover, the IP Security Maintenance and Extensions (IPSecME) working group at the IETF recently started standardizing a signaling extension that replaces the current cookie-based approach with a puzzle-based DoS protection mechanism [NS15]. We propose to integrate our presented attack detection and difficulty selection mechanisms with this standardization effort. The resulting combined approach then could also be used to provide DoS protection with Minimal IKEv2.

4.1.4 Accounting for Varying Message Processing Times

As discussed in Section 4.1.1.3, the high computation overhead of the public-key-based peer authentication process also has significant implications on the retransmission of handshake messages. More precisely, retransmissions in the IoT must account for the varying processing times of the exchanged handshake messages. This is to prevent premature retransmissions in case of computationally expensive handshake messages while still enabling the handshake peers to quickly recover from packet loss (see Section 4.1.1.3). DTLS, HIP DEX, and Minimal IKEv2, however, employ fixed, message-independent retransmission strategies. Hence, to tailor retransmissions to the special requirements in the IoT, we now present the *adaptive retransmission mechanism* that employs multiple worst-case estimates and feedback information from the handshake peer in order to derive message-specific retransmission timeouts. In doing so, our adaptive retransmission mechanism allows to significantly reduce the transmission overhead that is caused by premature retransmissions.

4.1.4.1 Adaptive Retransmission of Handshake Messages

We first observe that handshake messages commonly have a *dual role*. More precisely, they typically act as information carrier *and* reception acknowledgment. The R1 message of the HIP DEX protocol, for example, contains the Responder's static ECDH key as part of its carried protocol information (see Figure 3.7). In addition, this message, however, also *implicitly* informs the Initiator about the successful reception of the preceding I1 message at the Responder. This allows the Initiator, as the handshake peer that is responsible for retransmissions in case of HIP DEX (see Section 3.3.4), to limit the transmission overhead of retransmissions by waiting until the *expected* arrival time of the reception acknowledgement is exceeded.

Notably, the aggressive retransmission mechanism of the HIP DEX protocol[4] does not comply with the above observation. Instead, it employs a fixed-length retransmission timeout that is in the order of a few milliseconds, regardless of the specific network topology or the current network conditions. This particularly low

---

[4] We note that our adaptive retransmission mechanism recently replaced the aggressive retransmission strategy in the HIP DEX specification due to its improved transmission properties [MH14].

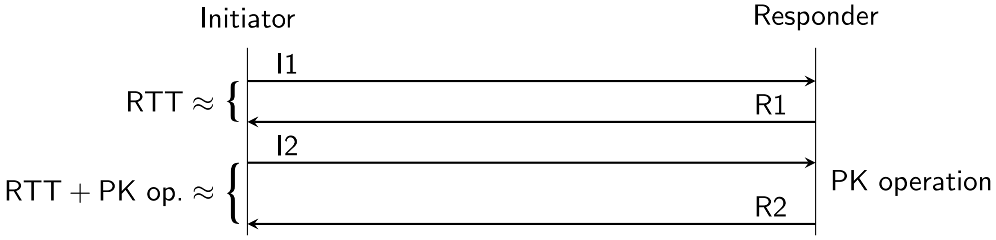


Figure 4.14 Expected delay between the transmission of the Initiator's and the reception of the Responder's handshake messages depending on the message type. The combined transmission delays of the I2 and the R2 messages along with the I2 processing time at the Responder quickly exceed a purely RTT-based retransmission timeout.

retransmission timeout, however, inevitably causes premature retransmissions if the network cannot guarantee an equally low Round-Trip Time (RTT). We therefore employ a more conservative retransmission strategy for our adaptive retransmission mechanism. Specifically, we adopt the *worst-case anticipated RTT* from the HIPv2 protocol [MHJH15]. To derive this worst-case estimate, we propose to sporadically sample the network via *ping* messages and to conservatively choose the retransmission timeout according to the observed RTTs while also accounting for a small message processing overhead. The Initiator then employs this primarily network delay-based timeout for I1 message retransmissions as this handshake message only trigger computationally inexpensive protocol operations at the Responder.

In contrast to the I1 message, the I2 message causes a non-negligible computation overhead at the Responder. This is because it triggers a public-key-based peer authentication process. As a result, the combined transmission delays of the I2 and the R2 messages along with the Responder's I2 processing time then quickly exceed the RTT-based retransmission timeout (see Figure 4.14). Thus, the Initiator additionally has to account for the Responder's computation time in our adaptive retransmission mechanism to prevent premature retransmissions in case of computationally expensive handshake messages. We, therefore, split the dual role of the R2 message and introduce an intermediate NOTIFY message that the Responder transmits *before* performing any computationally expensive handshake operations (see Figure 4.15). In doing so, the main intention is to enable the Responder to notify the Initiator about its successful I2 message reception. In addition, this NOTIFY message also enables the Responder to inform the Initiator about its *worst-case an-*

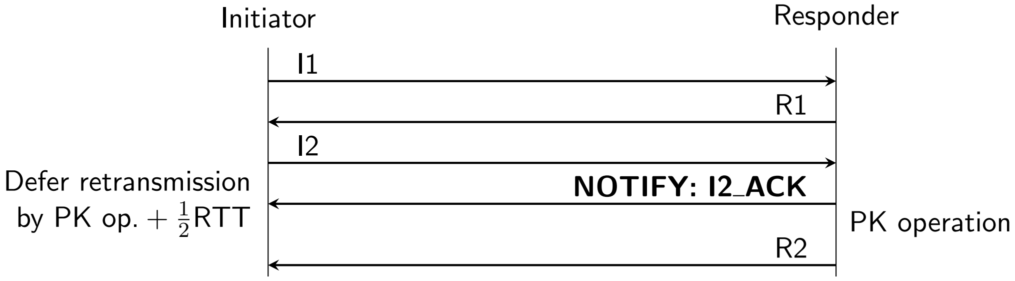


Figure 4.15 The Responder indicates its anticipated I2 message processing time in the I2_ACK parameter of our newly introduced intermediate NOTIFY message. When receiving this reception acknowledgement, the Initiator adapts its retransmission timeout to the indicated I2 processing time plus half the worst-case anticipated RTT for the transmission of the R2.

ticipated computation time for the received I2 message. For the derivation of this second worst-case estimate, the Responder, e.g., observes the maximum I2 processing time of its previous handshakes that involved the same cryptographic primitives. The Responder then adds this information to the I2_ACK parameter of the NOTIFY message as illustrated in Figure 4.15. Finally, the Responder concludes the handshake with a regular R2 message *after* it successfully processed the I2 message.

The introduction of a separate I2 reception acknowledgement in our adaptive retransmission mechanism enables the Initiator to first trigger I2 retransmissions based on the worst-case anticipated RTT. This affords a quick reaction to lost I2 messages based on the Initiator's observed network delay. Moreover, the Initiator resets and adapts the retransmission timeout to the indicated I2 processing time plus half the worst-case anticipated RTT as soon as it receives the Responder's intermediate NOTIFY message. This extended retransmission timeout then enables the Initiator to defer I2 retransmissions until the point in time when the Responder should have completed its I2 message processing and the network should have relayed the corresponding R2 message according to the employed worst-case estimates.

4.1.4.2 Further Considerations Regarding the Reception Acknowledgement

Concerning the extended retransmission timeout for the I2 message, we observe that the transmission of an R2 message implies the creation of session state at the Responder. Hence, the Responder no longer has to perform computationally expensive public-key operations when receiving an I2 retransmission from the Initiator for a lost R2 message (see Figure 4.16). Instead, the Responder can immediately reply to this I2 message based on its established session state. To account for this computationally inexpensive circumstance in our adaptive retransmission mechanism, we require the Responder *not* to send another NOTIFY message in case of an R2 retransmission. In addition, the Initiator sets the retransmission timeout for I2 retransmissions, which follow a NOTIFY message, to the worst-case anticipated RTT.

Moreover, it is worth noting that the additional NOTIFY message constitutes an unnecessary transmission overhead in case of a protocol handshake with an uncon-

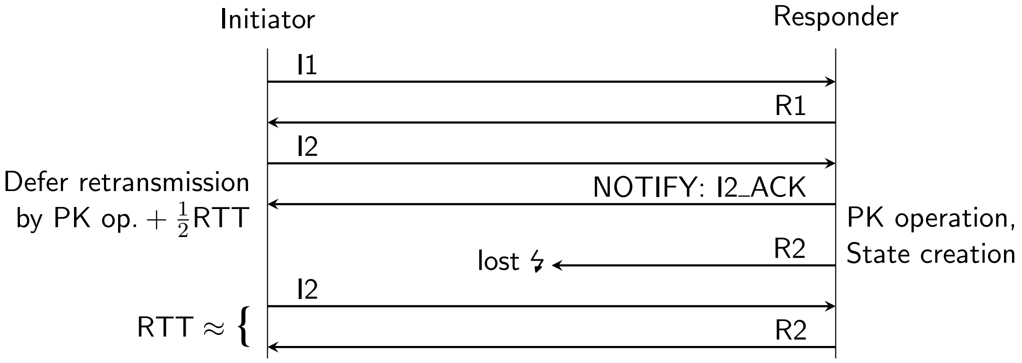

Initiator
Responder
I1
R1
I2
Defer retransmission by PK op. + ½RTT
NOTIFY: I2_ACK
PK operation, State creation
lost ϟ
R2
I2
RTT ≈ {
R2

Figure 4.16 Repeated R2 retransmissions no longer involve computationally expensive public-key operations. Instead, the Responder can immediately reply to the retransmission-triggering I2 message based on its previously established session state. Consequently, the Responder does not send another NOTIFY message in case of an R2 retransmission. Moreover, following a NOTIFY, the Initiator sets the I2 retransmission timeout to the worst-case anticipated RTT.

strained Responder. This is because the unconstrained Responder is able to quickly reply to all handshake messages, include those that trigger computationally expensive protocol operations. Hence, to prevent unnecessary transmissions, we allow Responders that are able to transmit an R2 message in a timely manner to omit the additional reception acknowledgement of our adaptive retransmission mechanism.

4.1.4.3 Pre-fetching Fragmented Handshake Messages

As we show in our evaluation in Section 4.1.6, the R1 and R2 messages typically have to be fragmented for transmission over size-constrained link layer technologies such as IEEE 802.15.4. Such fragmentation necessitates an additional adaptation of the introduced retransmission mechanism to further avoid premature retransmissions. Specifically, the Initiator may already have received *parts* of a Responder's handshake message when the retransmission timeout expires. At this point, the remaining fragments of the Responder's message may still be in transit. To also account for such remaining message fragments, the Initiator *pre-fetches* incomplete handshake messages from the lower layers of its network stack before triggering a retransmission with our adaptive retransmission mechanism. If the HIP DEX header is intact, this allows the Initiator to match a partially received message to the retransmission-triggering protocol handshake. As a result, the Initiator then further delays its retransmission based on the reassembly timeout at the lower protocol layers.

4.1.4.4 Integration with DTLS and Minimal IKEv2

As discussed in Section 4.1.1.3, the DTLS retransmission mechanism is prone to cause premature retransmissions in the context of the IoT, especially when considering current recommendations for public-key cryptography. Furthermore, it quickly delays the protocol handshake if consecutive message retransmissions are lost. We, therefore, propose to replace the existing retransmission approach of the DTLS protocol with our adaptive retransmission mechanism. To this end, the handshake peers simply adopt our pre-fetching approach and the RTT-based retransmission timeout as the components of our adaptive retransmission mechanism that do not affect interoperability. Moreover, the handshake peers can implement the intermediate reception acknowledgement via a DTLS Alert message that the peers send before performing computationally expensive protocol operations. Importantly, this Alert message should also contain the sequence numbers of the previously received handshake messages that belong to the same message flight. This accounts for the flight-based retransmission semantics of DTLS (see Section 3.3.3). A communication end-point that receives such an Alert message then deduces that its peer currently performs an expensive protocol operation and defers its retransmission of the missing flight messages based on the peer's indicated message processing time.

The pre-fetching approach and the RTT-based retransmission timeout can also be integrated with Minimal IKEv2. The separate transmission of a reception acknowledgement for computationally expensive handshake messages, however, currently cannot directly be adopted for Minimal IKEv2. This is because unsolicited notifications from the Responder during the initial IKE_SA_INIT message exchange (see Figure 3.8) are specified to cause the handshake to fail [KHN+14]. The Responder, however, is required to perform a computationally expensive ECDH operation as part of its IKE_SA_INIT message processing, thus requiring a reception acknowledgement in the context of our adaptive retransmission mechanism. To still afford a complete adoption of our adaptive retransmission mechanism for Minimal IKEv2, we propose that an exception for our reception acknowledgement notification should be defined as part of the on-going Minimal IKEv2 standardization process.

4.1.5 Security Considerations

We now briefly discuss potential attacks that an adversary can mount against the introduced protocol extensions based on the Internet Threat Model (see Section 3.1.1).

4.1.5.1 DoS Attack Against the Abbreviated Session Resumption Handshake

In contrast to the standard HIP DEX handshake, the I1 message of our abbreviated session resumption handshake additionally includes a MAC parameter (see Figure 4.3). This allows the Responder to re-authenticate the Initiator based on the previously established session state. Still, an adversary may also be able to exploit this inclusion of the MAC parameter in an I1 flooding attack against the Responder. To this end, the adversary would continuously send I1 messages with a forged MAC parameter to the Responder. The exploitable amplification effects of this attack, however, are comparable to the amplification effects that can be achieved in an I2 flooding attack against the original public-key-based protocol handshake. This is because the verification of the MAC parameter is based on efficient symmetric-key cryptography. As a result, the computation overhead of verifying the MAC parameter resembles the overhead of validating a puzzle solution in the I2 message of the *standard* HIP DEX protocol. Hence, our abbreviated session resumption handshake provides comparable security properties to the unaltered HIP DEX protocol design.

4.1.5.2 Replay Attack Against Session Resumption with State Compression

With session resumption, peer authentication is based on the stored session state and the fact that the MAC values in the exchanged messages of session resumption handshake can be verified correctly. The session resumption handshake with state compression, however, does not include variable protocol information. This prevents the Responder from determining the *freshness* (see Section 3.3.1) of a currently performed session resumption handshake. An eavesdropping adversary could exploit this fact by overhearing and later on replaying a legitimate session resumption handshake with state compression. As a result, the Responder would re-establish the corresponding legitimate session state. This enables the adversary to consume scarce memory resources at the Responder in a memory exhaustion attack.

To mitigate this attack, we require that the session state also contains a *session resumption counter*, which the communication end-points increment after each successful session resumption handshake. The end-points then use this counter as a modifier during the key derivation process of the subsequent session resumption handshake. As a result, the MAC values differ for each session resumption handshake although the conveyed protocol information is the same. This, in turn, allows the Responder to identify and to silently drop replayed handshake messages.

4.1.5.3 DoS Attacks Exploiting the Session Ticket Parameter

The session resumption handshake with state offloading introduces a new session ticket parameter as part of the I1 message. Notably, *any* network adversary could exploit the computation overhead at the Responder that is associated with the decryption and the verification of this parameter. To this end, the adversary would only have to flood the Responder with I1 messages that contain a forged session ticket. To efficiently identify such forged session tickets, we require that offloaded session state includes a (random) plaintext key identifier as part of the session ticket. The Responder then drops any session ticket with a key identifier that does not match its current session-state protection key (see Section 4.1.2.6). Thus, an adversary *without* overhearing capabilities is no longer able to mount the aforementioned attack.

An *eavesdropping* adversary, however, could still reproduce the correct key identifier in its forged session tickets after overhearing a legitimate session ticket. We, therefore, recommend that the Responder additionally rate-limits its processing of session tickets to further protect itself against this attack. Finally, an eavesdropping adversary could replay a *legitimate* session ticket in a session resumption handshake with state offloading. As a result, the Responder would re-establish the received session state, thus burdening its scarce memory resources. To mitigate this attack, we integrate a challenge-response mechanism in the design of our session resumption extension that allows the Responder to verify the freshness of a session resumption handshake with state offloading (see Section 4.1.2.4). Moreover, we note that the Initiator similarly can verify the freshness of a session resumption handshake with state offloading based on the session resumption counter introduced above.

4.1.5.4 On-path Attacks Against the Puzzle Signaling Extension

Our signaling extension for the HIP DEX puzzle mechanism enables an on-path gateway to inform the Responder about an unknown and therefore potentially untrustworthy Initiator by adding the VIA_UNKNOWN_NETWORK parameter to the I1 message of the HIP DEX handshake. The Responder then reacts to this notification parameter by issuing a puzzle difficulty for an unconstrained Initiator. The VIA_UNKNOWN_NETWORK parameter, however, is *not* cryptographically bound to a specific on-path gateway. This enables an on-path adversary who is located in the same constrained node network as the Responder to add this parameter to a traversing I1 message although both communication end-points are resource constrained. As a result, the adversary would be able to maliciously delay the targeted protocol handshake. The on-path adversary, however, could achieve the same results more effectively by postponing its message forwarding or even by simply dropping the messages of the targeted protocol handshake instead of forwarding them. Hence, our puzzle signaling extension does not degrade the security properties of HIP DEX.

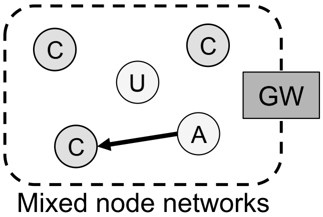


Figure 4.17 Example of a mixed node network that consists of constrained devices (C) and unconstrained communication end-points (U). The arrow indicates a protocol handshake that is triggered by an unconstrained adversary (A) but that does not involve an on-path gateway.

4.1.5.5 Collaborative DoS Protection and Mixed Node Networks

At the expense of diminished energy efficiency and increased production costs, recent advances in technology allow to equip networked embedded devices with computationally more powerful MCUs (see Section 2.2.1). These more powerful devices, however, may still employ constrained link layer technologies such as IEEE 802.15.4. Similarly, computationally constrained devices may be equipped with conventional wireless technologies such as IEEE 802.11 [IEEE12]. Both of these configurations allow network administrators to build mixed node networks that consist of constrained devices and unconstrained communication end-points. As depicted in Figure 4.17, this enables an unconstrained, network-internal adversary to exploit the low puzzle difficulty of a resource-constrained Responder in order to mount a DoS attack against this constrained device. This low puzzle difficulty results from the lack of an on-path gateway for our collaborative puzzle-based DoS protection mechanism.

To mitigate this attack, we propose that network administrators *logically* separate constrained and unconstrained communication end-points into individual IPv6 (sub-)networks. The resulting network structure then resembles Figure 4.9. Hence, a resource-constrained Responder again is able to select its puzzle difficulty based on the signaling information from the on-path gateway as described in Section 4.1.3.3.

4.1.5.6 Forged Message Reception Acknowledgements

Our adaptive retransmission mechanism introduces a new NOTIFY message that the Responder can use to inform the Initiator about the reception of an I2 message.

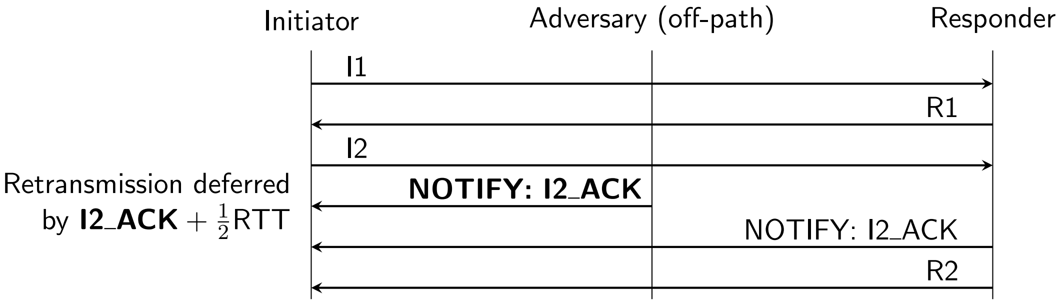


Figure 4.18 Spoofed I2 reception acknowledgments would enable an eavesdropping adversary to arbitrarily delay the protocol handshake. Importantly, the adversary cannot force the Initiator into setting the I2 retransmission timeout below half the worst-case anticipated RTT.

The Responder, however, cannot protect the authenticity of this message as the Responder did not yet perform the necessary public-key operation for the derivation of the corresponding keying material. As depicted in Figure 4.18, this enables an eavesdropping adversary to send a spoofed reception acknowledgement for an overheard I2 message and to signal an arbitrary I2 processing time to the Initiator. As a result, the adversary can, e.g., indicate a lower I2 processing time than actually required by the Responder in order to cause premature retransmissions.

To protect against this attack, the Initiator sets its computation-based retransmission timeout to the maximum indicated message processing time in case of conflicting NOTIFY messages. This allows the legitimate Responder to extend the retransmission timeout to the intended length. The adversary, however, can still arbitrarily delay the protocol handshake beyond the Responder's actual I2 processing time. To limit the extend of such a maliciously induced handshake delay, we additionally require the Initiator *not* to set the computation-based retransmission timeout beyond a configurable threshold. As this threshold is only required if either the I2 message is lost *after* it was observed by the adversary or in case of an R2 message loss, we propose to use a simple fixed threshold of a few seconds to frustrate this attack.

4.1.6 Evaluation

For our evaluation, we implemented the HIP DEX protocol for Contiki OS [DGV04] in version 2.5 according to the protocol specifications in [Mos12] and extended this implementation with our presented protocol extensions. As the underlying hardware platform for constrained devices, we used Zolertia Z1 motes[5] with a 16 MHz MSP430 MCU, 8 kB of RAM, 92 kB of ROM, and an IEEE 802.15.4 radio interface. To be able to evaluate the collaborative puzzle-based DoS protection mechanism in the context of an unconstrained adversary, we additionally ported this extended HIP DEX implementation to Linux. Moreover, we developed a simple Linux user-space firewall based on the *netfilter* framework [netfilter] for the gateway functionality of our presented DoS protection mechanism. The Linux-based protocol components then were executed on desktop-class computers with Intel i7 870 CPUs. All public-key operations relied on the *relic toolkit* [relic] and were based on elliptic curve SECP160R1.

In our evaluation, we did not consider the computation overhead of active link layer security. We, however, included the transmission overhead resulting from the *maximum length* of the link layer security header by reducing the available 6LoWPAN payload size by 21 byte. Moreover, we compressed the IPv6 and UDP header information via the LOWPAN_IPHC and LOWPAN_NHC compression mechanisms to the extend discussed in Section 2.4.1. As a result, HIP DEX handshake messages with a size above 42 byte, i.e., excluding lower layer headers, were fragmented by the 6LoWPAN layer prior to their transmission and reassembled at the fragment recipient (see Section 2.4.2). FRAG1s then contained up to 32 byte of HIP DEX protocol information, whereas FRAGNs carried between 1 and 72 byte of HIP DEX message content. Notably, by accounting for the link layer security header overhead and by considering modest header compression ratios, our evaluation results indicate a worst-case overhead regarding the number of message fragment transmissions.

---

[5] We refer to Section 2.2.1 for a brief overview and a comparison of the different experimentation platforms for constrained devices that we use throughout the course of this thesis.

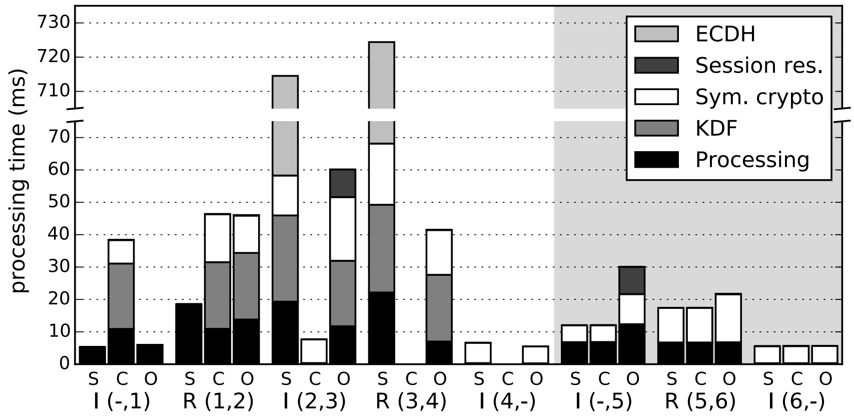


Figure 4.19 Processing time of the handshake and the connection teardown for the standard HIP DEX protocol (S), session resumption with state compression (C) and session resumption with Initiator-side state-offloading (O) for the Initiator (I) and the Responder (R). The numbers in brackets denote processing of the i-th and generation of the $(i+1)$-th handshake message.

4.1.6.1 Session Resumption Run-time Performance

To quantify the run-time performance of the session resumption extension, we measured the processing time and the transmission overhead of two consecutive *standard* HIP DEX handshakes (S) with an intermediate connection teardown exchange. We then compared this baseline against measurements for our session resumption with *state compression* (C) as the best-case and *Initiator-side state-offloading* (O) as the worst-case session resumption types with respect to the involved run-time overheads. We also evaluated the memory requirements for an inactive connection with session resumption. The processing overhead results (see Figure 4.19) denote the average over 100 measurements with two wirelessly connected Z1 motes. The standard deviation for the corresponding measurements was below 16.63 ms (i.e., 2.08 %) for the public-key-based operations and below 0.15 ms for all remaining protocol operations.

Processing overhead

As depicted in Figure 4.19, the significant reduction of the processing overhead for repeated connection establishments is the main strength of our session resumption extension (note the interrupted y-axis). The achieved performance gain primarily stems from the fact that our extension refrains from the use of public-key cryptography during the session resumption handshake. More precisely, the ECDH-based public-key operation in the *standard* HIP DEX handshake incurs a computation overhead of about 656.25 ms per communication end-point (see “I(2,3)” and “R(3,4)” in Figure 4.19). This high cryptographic overhead causes the standard handshake to amount to a total overhead of about 1469.41 ms for both end-points combined. This total overhead compares to 92.47 ms for session resumption with state compression and to 159.17 ms for Initiator-side state-offloading. We note that the performance difference of 66.7 ms between the two session resumption types is caused primarily by the protection of the Initiator’s offloaded session state (see “I(2,3)” in Figure 4.19) and by the extended message processing for the longer session resumption handshake in case of Initiator-side state-offloading (see “R(3,4)” and “I(4,-)” in Figure 4.19).

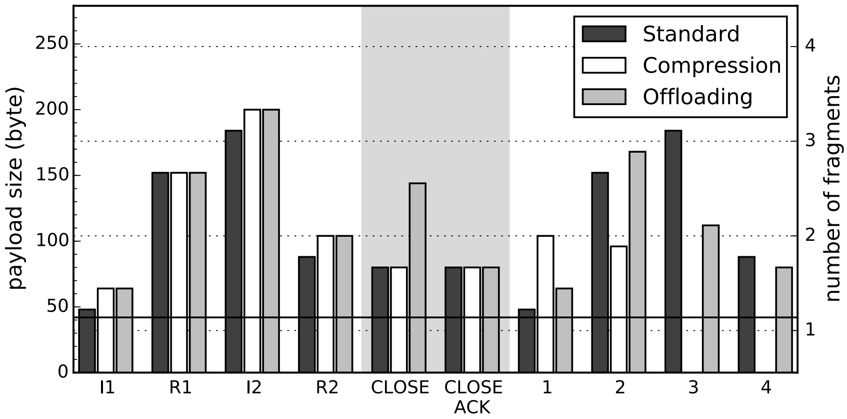


Figure 4.20 Message size for two subsequent handshakes with an intermediate connection teardown (marked with a gray background) for the *standard* HIP DEX protocol and for session resumption with state *compression* as well as with Initiator-side state-*offloading*. The continuous line indicates the 6LoWPAN fragmentation threshold in our evaluation setup. The dashed lines then indicate the maximum payload size of the individual 6LoWPAN fragments.

Regarding the connection teardown, we observe that session resumption with state compression exhibits the same processing overhead as the standard HIP DEX protocol. This is due to the unmodified connection teardown exchange for this type of session resumption. In contrast, Initiator-side state-offloading increases the processing overhead of the connection teardown exchange by 8.42 ms (i.e., 64.1 %). This is the result of the additional session state encryption at the Initiator (see “I(-,5)” in Figure 4.19) and a slight increase of the MAC overhead due to an increased size of the CLOSE message (see “I(-,5)” and “R(5,6)” in Figure 4.19). To put these numbers into perspective, session resumption *reduces* the overall processing overhead of a complete connection establishment and connection teardown cycle by at least 85.6 % (i.e., Initiator-side state-offloading) and by 91.5 % at best (i.e., state compression).

Transmission overhead

As shown in Figure 4.20, our session resumption extension marginally increases the size of the I1, I2, and R2 messages during the initial connection establishment handshake compared to the standard HIP DEX protocol. This increased transmission overhead exclusively stems from the additional signaling information for the negotiation of the mutually preferred session resumption type. In case of state compression, the connection teardown exchange and the subsequent session resumption handshake allow to compensate and, in fact, to significantly reduce the transmission overhead for repeated protocol handshakes. Specifically, each cycle of connection teardown and session resumption requires the transmission of 360 byte (i.e., 8 fragments). This compares to 632 byte (i.e., 15 fragments) for the standard HIP DEX handshake. In other words, state compression decreases the transmission overhead by 43.0 % and reduces the number of 6LoWPAN fragments by 46.7 %. Our session resumption extensions predominantly achieves this overhead reduction by abbreviating the session resumption handshake with state compression from 4 to 2 handshake messages.

Contrary to state compression, Initiator-side state-offloading requires the transmission of an additional 64 byte for the Initiator's offloaded session state in the CLOSE message of the connection teardown (see Figure 4.20). Correspondingly, the number of 6LoWPAN fragments for this message increases from 2 to 3. Moreover, the subsequent session resumption handshake consists of 4 handshake messages that also convey the Initiator's offloaded session state. As a result, each connection teardown and session resumption cycle requires the transmission of 648 bytes (i.e., 15 fragments). Hence, Initiator-side state-offloading slightly increases the transmission overhead by about 2.5 %, while showing an equal number of transmitted 6LoWPAN fragments.

Importantly, the transmission overhead of the standard HIP DEX protocol further increases when elliptic curves with larger field sizes are employed during the protocol handshake. In case of NIST P-256, e.g., we find that the transmission overhead of the standard HIP DEX protocol amounts to 696 byte for a connection establishment and connection teardown cycle (see Section 4.2.3.1). Our session resumption extension, contrarily, has a constant overhead. Thus, the relative transmission overhead reductions that we achieve with the session resumption extension further improve for an increasing elliptic curve size with the standard HIP DEX handshake. This leads to an effective 6.9 % overhead *reduction* for session resumption with Initiator-side state-offloading compared to a standard HIP DEX handshake with NIST P-256.

Memory trade-offs

With session resumption, the handshake peers have to maintain session state across inactive connections. In case of session resumption with *state compression*, this session state amounts to 38 byte per peer. This compares to 151 byte for an active HIP DEX connection. We achieve this 74.8 % compression of the session state by omitting protocol information that is re-negotiated during the subsequent session resumption handshake and by discarding session information that is only relevant for the initial session establishment. *State offloading* allows to further lift this storage requirement for one handshake peer. In addition to its own compressed session state, the other peer then also has to store 57 bytes of encrypted session state. Both session states combined, however, still result in an overhead reduction by about 37.1 % compared to an active HIP DEX connection. Consequently, from a memory perspective, session resumption always is advantageous to maintaining an active connection for an extended period of time. Based on the overall evaluation results, we, therefore, conclude that the session resumption extension provides an effective mechanism to reduce the run-time overhead of repeated protocol handshakes.

4.1.6.2 DoS Protection Properties

To evaluate the collaborative puzzle-based DoS protection mechanism, we considered a network setup consisting of three constrained devices C_1 to C_3 and two Linux machines L_1 and L_2 as depicted in Figure 4.21. L_1 was connected to L_2 via an Ethernet connection and took the role of an unconstrained adversary. We additionally attached L_2 to C_1 via a USB cable in order to equip $L2$ with support for the IEEE 802.15.4 radio links. Consequently, C_1 and L_2 combined represented the on-path gateway in our evaluation setup. The remaining two constrained devices acted as a benign Initiator (C_2) and a benign Responder (C_3). These devices communicated with each other via IEEE 802.15.4 radio links. Moreover, the adversary L_1 was able to interact with C_2 and C_3 in an end-to-end manner via the on-path gateway.

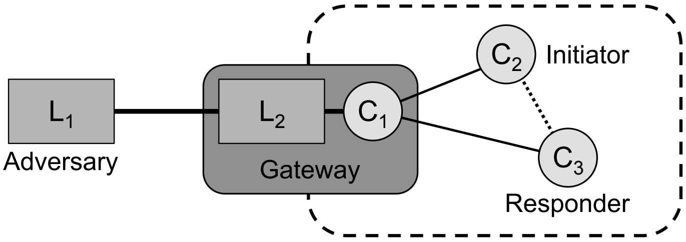


Figure 4.21 Evaluation setup for our collaborative puzzle-based DoS protection mechanism. Supported by the on-path gateway (C_1 and L_2), the Responder (C_3) was able to protect itself against an unconstrained adversary (L_1). At the same time, the benign Initiator (C_2) could still perform a limited amount of legitimate handshakes with the Responder (C_3).

Regarding the configuration parameters of the collaborative puzzle-based DoS protection mechanism, we set the window period to 64 s. This allowed us to implement the sliding window with a simple bitfield based on a single *uint64_t*. Furthermore, we considered five protocol handshakes per window period to be acceptable for a resource-constrained Responder and set the puzzle issuing threshold accordingly.

To derive the Responder's issued puzzle difficulty (see Section 4.1.3.3), we measured the average computation time of puzzles with varying difficulties on a constrained device and a Linux machine. In doing so, we assumed that the Linux machines in our evaluation setup were equipped with state-of-the-art high-end CPUs and, therefore, represented the most powerful adversary that the puzzle-based DoS protection mechanism should defend against. Based on the gathered measurement results, we derived a puzzle difficulty of 14 for resource-constrained Initiators with an average computation time of 64.56 s and of 22 for unconstrained Initiators with about 60.44 s.

With this evaluation setup, we then evaluated the impact of a DoS attack from L_1 against C_3. To this end, L_1 successively flooded C_3 with standard protocol handshakes. During this attack, we measured the maximum number of public-key operations at C_3 during a single window period over a timespan of 15 min. In addition, we analyzed the impact of this attack on legitimate protocol handshakes. To this end, we also measured the number of successful handshakes between C_2 and C_3 during the above attack. We considered three different configurations for comparison purposes: i) an end-point-*independent* puzzle difficulty of 22, ii) an end-point-*independent* puzzle difficulty of 14, and iii) end-point-*specific* puzzle difficulties of 14 or 22 assisted by the on-path gateway. As a baseline, we also counted the number of successful handshakes for a 15 min period *without* an attack.

For the end-point-independent puzzle difficulty of 22, we found that the number of legitimate handshakes decreased from 332 (i.e., no attack) to 7 within the evaluated time span. This is because the unconstrained adversary L_1 and the resource-constrained Initiator C_2 competed for the 5 handshakes per window period with a puzzle difficulty of 0. In addition, C_2 stopped processing puzzles with a difficulty of 22, for which the computation time of finding a solution exceeded the Responder's indicated puzzle validity period of 64 s. C_2 then requested a new puzzle from C_3 via a repeated I1 message as described in Section 4.1.3.4. Moreover, we also observed a puzzle-related decrease of *malicious* handshakes. Specifically, the Responder C_3 performed as few as 7 public-key operations per window period for both end-points L_1 and C_2 combined. Of these public-key operations, 5 belonged to handshakes with a puzzle difficulty of 0. Hence, the puzzle difficulty of 22 successfully protected the

Responder's scarce computation resources during L_1's attack against the protocol handshake. This high puzzle difficulty, however, also prevented a large number of the legitimate handshakes between C_2 and C_3 from completing successfully.

The end-point-independent puzzle difficulty of 14 similarly decreased the number of legitimate handshakes to 8. Contrary to the puzzle difficulty of 22, this, however, was the result of the undemanding puzzle difficulty for L_1 (i.e., 0.21 s) and the corresponding high load at C_3. Specifically, C_3 had to perform as many as 29 public-key operations per window period. With the *collaborative puzzle-based DoS protection mechanism*, the legitimate end-points performed a total of 14 handshakes. This increased number of legitimate handshakes was achieved by selectively issuing a puzzle difficulty of 14 for C_2 and a puzzle difficulty of 22 for L_1 supported by the on-path gateway. We note that this number is close to the goal of allowing at least one handshake per communication end-point and window period during an on-going attack and comprises of 2 handshakes with a puzzle difficulty of 0 as well as 12 handshakes with a puzzle difficulty of 14. Of these 12 handshakes, 6 involved puzzles that C_2 solved within a single puzzle validity period, whereas the remaining 6 handshakes required more than one puzzle solving iteration. Importantly, we also observed an equally low number of public-key operations per window period at C_3 (i.e., 7) as for the end-point-independent puzzle difficulty of 22. Thus, we conclude that our collaborative puzzle-based DoS protection mechanism can successfully defend a resource-constrained Responder against attacks from an unconstrained adversary, while still allowing for a controlled number of legitimate handshakes to succeed.

4.1.6.3 Retransmission Improvements

To determine the performance of the adaptive retransmission mechanism, we implemented the standard HIP DEX and a DTLS-based retransmission strategy as a baseline. We then compared the observed results to the performance of the adaptive retransmission mechanism. To this end, we measured the overall transmitted bytes and the time until handshake completion for two directly interconnected nodes in the Cooja network simulator [ODE+06]. We thereby varied the *end-to-end* loss probability from 0 % to 80 % for each transmitted 6LoWPAN fragment. We decided for simulation over a real testbed to be able to compare the different retransmission strategies for well-defined loss probabilities without side-effects on the wireless medium. The presented results denote the average over 5000 handshakes for each combination of retransmission mechanism and loss probability.

In a preceding simulation run without network load, we observed an RTT of about 30 ms. We still set the *aggressive* timeout of the standard HIP DEX retransmission mechanism to 100 ms instead of a few milliseconds as lower timeout values caused this strategy to trigger an excessive amount of premature I2 retransmissions and a high message processing delay at the simulated nodes, even in case of computationally inexpensive handshake messages. Similarly, we configured the worst-case anticipated RTT for our adaptive retransmission mechanism to 100 ms in order to account for moderate network and processing delays. Finally, we set the worst-case anticipated computation time to 750 ms. In doing so, we took the I2 processing overhead in Section 4.1.6.1 as a basis (also see "S" for "R(3,4)" in Figure 4.19). With these settings, the standard HIP DEX retransmission mechanism and our own approach primarily differ with respect to the newly introduced I2 reception acknowledgement.

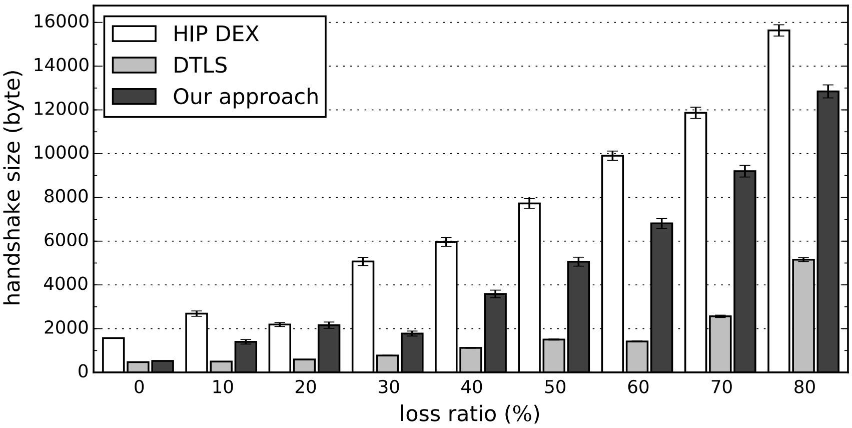


Figure 4.22 Transmission overhead of the standard *HIP DEX*, the *DTLS*-based, and our adaptive retransmission strategy for different loss probabilities. The error bars depict the 99% confidence intervals. The DTLS-based strategy causes the lowest transmission overhead, while *our approach* outperforms standard HIP DEX.

As shown in Figure 4.22, the standard HIP DEX retransmission strategy exhibits the worst transmission overhead of the considered retransmission strategies. In fact, it requires the transmission of about 1570 byte, i.e., 333 % of a standard HIP DEX handshake, even if no handshake message is lost. The reason for this high transmission overhead are premature I2 retransmissions that are triggered by the Initiator despite the fact that the initial I2 message was correctly received by the Responder. With our adaptive retransmission mechanism, the Responder avoids these premature retransmissions by sending an I2 reception acknowledgement to the Initiator. As a result, the transmission overhead of our approach is 526 bytes (i.e., a full handshake and the NOTIFY-based reception acknowledgement) in case of a 0 % loss probability. Also with increasing loss probabilities, our adaptive retransmission mechanism commonly achieves a lower transmission overhead than the standard HIP DEX retransmission mechanism (see Figure 4.22). This is because the loss of the newly introduced (small) NOTIFY message only causes a marginally elevated transmission overhead compared to the standard HIP DEX retransmission mechanism. In contrast, the correct reception of the NOTIFY message enables the Initiator to refrain from retransmitting while the Responder is busy processing the I2 message.

Besides the above transmission overhead reductions, we also observe that our adaptive retransmission mechanism outperforms the standard retransmission mechanism of HIP DEX with respect to the handshake run-time. For a loss probability of 0 %, our approach, e.g., requires about 1588 ms, whereas the standard mechanism needs about 1607 ms. We observe that this observation similarly holds for increasing loss probabilities (see Figure 4.23). This is because the decreased number message retransmissions reduces the fragment processing overhead at the 6LoWPAN layer as well as the message processing burden at the HIP DEX protocol implementation.

Concerning the DTLS-based retransmission strategy, premature retransmissions can largely be avoided in our evaluation setup (see 0 % in Figure 4.22). However, this is primarily because the Initiator starts triggering I2 retransmissions after a fixed timeout of 1 s, whereas the Responder requires about 724.4 ms to process a received I2 message and to generate the corresponding R2 response (see “R(3,4)” in Figure 4.19). Thus, when considering elliptic curves with larger field sizes such as NIST P-256,

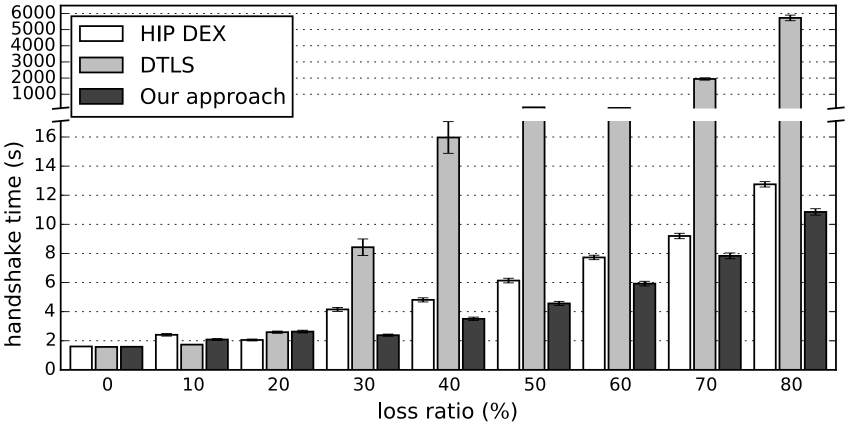


Figure 4.23 Handshake run-time of the standard *HIP DEX*, the *DTLS*-based, and our adaptive retransmission strategy depending on the loss probability. The error bars depict the 99% confidence intervals. *Our approach* outperforms the other considered retransmission strategies for nearly all loss probabilities.

the DTLS-based strategy is also prone to cause premature retransmissions due to an increased I2 processing time at the Responder (e.g., 1.87 s for NIST P-256). We, therefore, conclude that the integration of an additional reception acknowledgement also is desirable in the context of the DTLS-based retransmission mechanism.

Notably, the DTLS-based retransmission strategy shows a lower overall transmission overhead than our adaptive and the standard HIP DEX retransmission mechanisms (see Figure 4.22). For example, in case of loss probability of 50 %, the DTLS-based strategy requires the transmission of about 1502 byte, while our approach and the standard HIP DEX retransmission mechanism incur overheads of about 5062 byte and 7723 byte, respectively. As the main reasons for this difference in transmission overhead, we identified the single 6LoWPAN reassembly buffer of Contiki OS and its default reassembly timeout of 8 s. These cause the remaining message fragments of an incompletely transmitted handshake message to occupy the Responder's reassembly buffer until the reassembly timeout expired. During this timespan, the Responder drops all other handshake messages due to an already reserved reassembly buffer. The DTLS-based retransmission strategy handles this condition with only a few retransmissions as its exponential back-off quickly prolongs the retransmission timeout to a point where retransmissions are no longer dropped by the Responder. As a result, the DTLS-based retransmission strategy causes a lower transmission overhead in our evaluation setup than our adaptive or the standard HIP DEX retransmission mechanisms, which do not employ an exponential back-off.

It is important to note that the trade-off for this lower transmission overhead is a significant increase in handshake run-time for the DTLS-based retransmission strategy. For instance, considering the same 50 % loss probability, the DTLS-based retransmission strategy requires about 189.79 s for the successful conclusion of the HIP DEX protocol handshake. As shown in Figure 4.23, our adaptive retransmission mechanism contrarily still affords a timely handshake conclusion even for increasing loss probabilities by triggering retransmissions based on the worst-case anticipated RTT and computation-related feedback from the handshake peer. This affords our approach to significantly outpace the DTLS-based retransmission strategy.

Extension	ROM (byte)	RAM (byte)
Contiki with HIP DEX	58733	7042
+ Session resumption	63263 [+4530]	7198 [+156]
+ DoS protection	59369 [+ 636]	7066 [+ 24]
+ Retransmissions	59159 [+ 426]	7066 [+ 24]
All combined	64325 [+5592]	7246 [+204]

Table 4.2 Memory requirements of the presented protocol extensions. Numbers in brackets denote added overhead compared to Contiki OS with an unmodified HIP DEX implementation. Combined, our extensions require less than 5.6 kB of ROM and about 0.2 kB of RAM.

Overall, we conclude that the adaptive retransmission mechanism consistently improves on the characteristics of the standard HIP DEX retransmission mechanism. Moreover, our approach allows for a more timely handshake conclusion for lossy network scenarios than the DTLS-based retransmission strategy. This timeliness, however, involves a trade-off with respect to an increased transmission overhead that primarily stems from the inefficient handling of fragment loss at the 6LoWPAN layer. We note that the split buffer approach for the 6LoWPAN layer, which we will present in Chapter 6, enables the handshake peers to manage their reassembly resources more efficiently and, thus, presumably would allow to reduce the transmission overhead of the adaptive retransmission mechanism. Moreover, the run-time/transmission trade-off of our approach can further be adjusted by adding an exponential back-off to the network delay-based retransmission timeout. This back-off, however, must be limited to a few seconds in order to prevent considerable handshake delay in case of packet loss as demonstrated by the DTLS-based retransmission strategy.

4.1.6.4 RAM and ROM Overhead

As discussed in Section 2.2.1, constrained devices commonly have very limited RAM and ROM resources. Our evaluation platform, e.g., is equipped with 8 kB of RAM and 92 kB of ROM. Hence, to evaluate the memory impact of the presented protocol extensions on a constrained device, we derived RAM and ROM estimates for their respective implementation. To this end, we analyzed the Contiki OS binaries used during our evaluation with the *msp430-size* tool, which is part of the GCC toolchain for the MSP430 MCU [mspgcc]. The first binary contained an unmodified HIP DEX implementation, whereas the remaining four binaries also included the required protocol functionality for (i) the flexible session resumption mechanism, (ii) the collaborative puzzle-based DoS protection mechanism, (iii) the adaptive retransmission mechanism, and (iv) a combination of these protocol extensions.

As shown in Table 4.2, our session resumption extension incurs a moderately increased memory overhead compared to a standard HIP DEX protocol implementation, i.e., 4530 byte of ROM and 156 byte of RAM[6]. The ROM overhead primarily stems from the additional message processing functionality and the CCM mode of operation for AES that are required for session resumption with state-offloading.

---

[6] The presented RAM overheads denote statically allocated RAM requirements and do not include run-time overheads for the stack or the heap. Still, it is worth noting that our implementation refrains from dynamic memory allocation, e.g., via *malloc*, to maintain persistent run-time state.

The increased RAM requirements are caused by the fact that the handshake peers need to maintain additional session state during an active connection, e.g., the negotiated session resumption type. Moreover, we configured the constrained devices to accept two offloaded session states of 57 byte each from their handshake peers.

The collaborative puzzle-based DoS protection mechanism and the adaptive retransmission mechanism can be realized at a marginal memory overhead. More precisely, the overhead for both of these mechanisms combined amounts to less than 1100 byte of ROM and less than 50 byte of RAM. We achieve this notably low memory overhead by focusing on existing protocol mechanisms and by adapting them according to the special device and network characteristics in the embedded domain.

Combined, our protocol extensions require less than 5.6 kB of ROM and about 0.2 kB of RAM. These overheads denote the main trade-off for addressing the high computation requirements of the protocol handshake in the context of constrained devices.

4.1.7 Related Work

For our discussion of related work, we distinguish between the following two areas of research: i) progress in the context of public-key-related cryptographic primitives and ii) protocol mechanisms that are related to our presented protocol extensions.

4.1.7.1 Progress of Public-Key-Related Cryptographic Primitives

Concerning the overhead of public-key cryptography on constrained devices, Hu et al. [HCSO09] show that the integration of commodity Trusted Platform Modules (TPMs) in the hardware design of constrained devices allows to reduce the computation overhead of RSA-based cryptographic operations. Kothmayr et al. [KSH$^+$12, KSH$^+$13] subsequently confirm this result with a public-key-enabled DTLS implementation for TPM-equipped constrained devices. Such hardware-based solutions, however, further increase the production cost of a constrained device and do not address the root cause of the high message processing overhead in the protocol design. In contrast, session resumption allows to forfeit the use of public-key cryptography for repeated protocol handshakes and additionally affords to reduce the transmission overhead. Moreover, session resumption does not mandate the use of special-purpose hardware for constrained devices. Still, it is worth noting that session resumption and TPMs can complement each other effectively. Specifically, TPMs allow to reduce the computation overhead during the initial session establishment, while session resumption affords to save the scarce resources of an energy-constrained device by allowing the TPM to be turned off for the remaining session lifetime.

Various optimization techniques including the Barrett reduction algorithm [Bar87], projective coordinates [HMV04], and scalar multiplication based on a sliding window [HMV04] allow to reduce the computation overhead of software-based ECC implementations. Lui et al. [LN08], however, show that such optimizations also imply a memory trade-off for constrained devices. Implementors, therefore, may choose to forgo these optimizations in favor of our session resumption mechanism. This is especially true for memory-constrained devices. Session resumption then limits the resulting increase in computation time to the initial connection establishment and affords an efficient symmetric-key-based connection re-establishment. In addition, our adaptive retransmission mechanism adjusts the retransmission timeout for the initial connection establishment according to the increased computation time.

Implicit authentication schemes allow to reduce the computation and transmission overhead of public-key cryptography by integrating operations that traditionally are separated. Implicit certificates [Zav11, PKG$^+$13], e.g., super-impose the attested public key of a certificate issuer and an attesting signature of a Certificate Authority (CA). This allows to decrease the size of implicit certificates compared to traditional X.509 certificates [CSF$^+$08]. Moreover, implicit certificates do not require an explicit verification of the CA's signature. Instead, the authenticity of the attested public key can be verified via the correct use of the corresponding private key during the protocol handshake, e.g., via a signature. As a result, implicit certificates also exhibit a lower computation overhead than traditional certificates. Similarly, authenticated DH protocols such as HMQV [Kra05] allow to implicitly authenticate an ephemeral DH key agreement via long-term public-key identities without the need to separately verify these identities. Still, while significantly reducing the overhead of public-key cryptography by integrating previously separated public-key operations, such implicit authentication schemes still incur the overhead of at least one public-key operation. Hence, our presented protocol extensions also apply when employing these novel cryptographic mechanisms for peer authentication purposes.

In [KKG10, VMZS$^+$13, GMKK$^+$13b], the authors propose to replace the public-key operations in the DTLS, HIPv2, and HIP DEX protocols with polynomial schemes. Similar to a purely symmetric-key-based approach, these approaches, however, depend on the secure provisioning of polynomial shares to the communication endpoints *prior* to the end-to-end handshake (see Section 4.1). Thus, while providing an adequate solution for isolated networks, these approaches require non-trivial coordination between administrative domains to afford secure communication across network boundaries. In contrast, our work focuses on standard cryptographic primitives that allow to exchange public key information and to use established Public-Key Infrastructures (PKIs) for secure communication across network domains.

4.1.7.2 Related Protocol Mechanisms

Several TLS extensions [SB01, SB02, SBR04, Lan10, ST15] have been proposed that enable the Initiator to cache static handshake information such as the Responder's public key during an initial protocol handshake. The handshake peers then omit the transmission of this cached information during a subsequent handshake. However, while allowing to reduce the transmission overhead, these approaches do not address the computation overhead stemming from public-key operations during a subsequent protocol handshake. Moreover, the memory burden caused by these approaches during device run-time may be significant for constrained devices depending on the cached handshake information. This is especially true when caching involves certificates or even certificate chains. Hence, session resumption typically exhibits a lower computation and memory overhead than these caching-based approaches.

Cryptographic puzzles were initially presented by Dwork et al. [DN93] as a defense mechanism against spam email. Juels et al. [JB99] subsequently showed that client puzzles also allow to frustrate DoS attacks that aim at exhausting a server's session-state resources. Aura et al. [ANL01] similarly apply client puzzles to protect a server's public-key-based peer authentication process. In contrast to our collaborative puzzle-based DoS protection mechanism, these approaches focus on the underlying puzzle construction and do not consider the selection of an appropriate puzzle difficulty. Dean et al. [DS01] propose to adopt client puzzles for TLS. Their approach, however, only employs a single fixed puzzle difficulty for all handshake peers and introduces separate thresholds to turn puzzle issuing on and off in order to prevent oscillation around a single puzzle issuing threshold. In contrast, our collaborative puzzle-based DoS protection mechanism differentiates between constrained and unconstrained handshake peers and leverages such oscillation effects to afford legitimate handshakes to complete despite an otherwise excessive puzzle difficulty. Similar to our work, Nie et al. [NVHAG11] identify high puzzle difficulties to have an adverse effect on a HIP DEX handshake involving a constrained device. The authors therefore propose to consider the Received Signal Strength Indication (RSSI) for received handshake messages to detect an unconstrained adversary that is located *inside* the same constrained node network as the target device. Contrary to our attack detection mechanism, this approach is based on the assumptions that the adversary uses a higher transmission power than legitimate constrained devices and that the adversary is located within the one-hop neighborhood of the target device.

With our adaptive retransmission mechanism, we propose to augment an RTT-based retransmission timeout with feedback information from the handshake peer in order to prevent premature retransmissions for computationally expensive handshake messages. Our feedback approach is inspired by similar feedback mechanisms such as the Explicit Congestion Notification (ECN) mechanism for IP [RFB01] and the RTP Control Protocol (RTCP) [SCFJ03]. These mechanisms, however, primarily consider *previously observed network characteristics*, whereas our approach focuses on the *anticipated message processing time* at a handshake peer. As discussed in Section 4.1.6.3, the network delay-based retransmission timeout of our adaptive retransmission mechanism may be extended with an exponential back-off to account for 6LoWPAN fragment loss. Such an exponential back-off can, e.g., be based on the latest recommendations for the TCP retransmission timeout [PACS11]. However, in contrast to this specification, we propose to limit the exponential back-off to a few seconds in order to prevent a considerable handshake delay in case of packet loss.

Finally, several delegation-based architectures were recently proposed that allow to offload expensive protocol operations during the handshake phase to a more powerful network entity. For a detailed discussion of these approaches, we refer to the related work section of the handshake delegation architecture in Chapter 5.

4.1.8 Summary

In this section, we analyzed the impact of public-key cryptography on the DTLS, HIP DEX, and Minimal IKEv2 handshakes in the context of the IP-based IoT. During our protocol analyses, we identified three main challenges that directly stem from the use of public-key cryptography during a protocol handshake with a constrained device. First, public-key operations require a substantial computation time. Hence, a public-key-based protocol handshake should only be performed infrequently and if not avoidable. Second, expensive public-key operations aggravate the risk of a DoS attack against constrained devices as even a single unconstrained adversary can successfully mount such attacks against the protocol handshake. Consequently, DoS protection mechanisms must account for the high resource asymmetry in the IoT in order to mitigate such attacks. Third, the different messages of the protocol handshake incur a highly varying computation time on a constrained device. Thus, retransmission mechanisms must no longer only rely on fixed or purely network delay-based retransmission timeouts in order to prevent premature retransmissions.

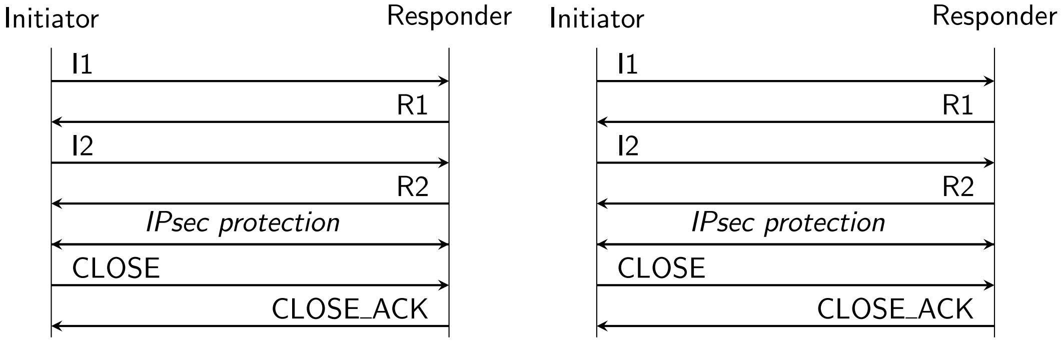
To address these protocol design issues, we introduced three lightweight, complementary protocol extensions for HIP DEX as an exemplary end-to-end security protocol for the IoT. These three protocol extensions most notably are: i) a flexible session resumption mechanism, ii) a collaborative puzzle-based DoS protection mechanism, and iii) an adaptive retransmission mechanism. As the evaluation results show, the session resumption mechanism allows to reduce the computation overhead by up to 91.5 % and transmissions by up to 43.0 % for repeated protocol handshakes compared to the standard HIP DEX protocol. Moreover, the collaborative puzzle-based DoS protection mechanism effectively accounts for the resource asymmetry in the IoT and successfully protects constrained devices against a significantly more powerful adversary. Lastly, the adaptive retransmission mechanism allows for a timely handshake conclusion despite packet loss while preventing premature retransmissions. In combination, the presented protocol extensions, thus, considerably enhance the efficiency and security of the HIP DEX protocol and similarly afford an improved applicability of DTLS and Minimal IKEv2 in the context of the IP-based IoT.

4.2 Tailoring the Transmission Overhead

As discussed in Sections 3.3.3 to 3.3.5, a core design principle for the specification of the DTLS, HIP DEX, and Minimal IKEv2 protocol adaptations is to preserve the protocol semantics and the security guarantees of TLS, HIPv2, and IKEv2, respectively. The TLS, HIPv2, and IKEv2 protocols, however, were primarily designed with extensibility and flexibility in mind. Consequently, message conciseness only was a secondary goal during the standardization process. The DTLS, HIP DEX, and Minimal IKEv2 protocol adaptations, therefore, contain a considerable amount of dispensable information in their protocol messages that constitutes undesirable transmission overhead in the context of constrained node networks. As we show in our evaluation, this dispensable information effectively leads to an *increased message fragmentation* during the protocol handshake. The transmission of these additional fragments, however, commonly wastes the scarce resources of energy-constrained devices and unnecessarily burdens forwarding nodes on the communication path.

To adapt the message wire-format to IoT requirements, related work proposes compression schemes that focus on DTLS [RSH^{+}13, RSD14] and the IPsec protocol suite [GMSS10, RDC^{+}11, RVJ12, MG14]. Similar to these approaches, we present a compression layer called *Slimfit* that addresses message conciseness for HIP DEX.

We now introduce the HIP DEX message wire-format as the main protocol component under investigation in this section and identify dispensable protocol information



(a) HIPv2 **(b)** HIP DEX

Figure 4.24 Comparison of the connection establishment handshake and the connection teardown exchange for HIPv2 and HIP DEX. Notably, both protocols employ the same exchange structure. This is because HIP DEX inherits the primary message exchanges from HIPv2.

in Section 4.2.1. Based on these results, we present the design of the Slimfit compression layer for HIP DEX in Section 4.2.2. Here, we also highlight the applicability of Slimfit in the context of HIPv2. Section 4.2.3 then presents our evaluation results and Section 4.2.4 discusses security considerations. Finally, we address related work in Section 4.2.5 and conclude this section with a brief summary in Section 4.2.6.

4.2.1 Analysis of the HIP DEX Message Wire-Format

As a result of preserving the HIPv2 protocol semantics (see Figure 4.24), HIP DEX most notably inherits the general protocol structure for the connection establishment handshake and the connection teardown exchange. In addition, HIP DEX also adopts the general HIP message wire-format for the individual protocol messages. In particular, this affords the (optional) integration of existing protocol extensions such as the support for end-point mobility [HVA15] or the integration of IPsec for payload protection purposes [JMM15] without the need for further modifications.

As shown in Figure 4.25, the HIP message wire-format includes a *fixed HIP protocol header* that is included in all protocol messages. Moreover, the wire-format contains

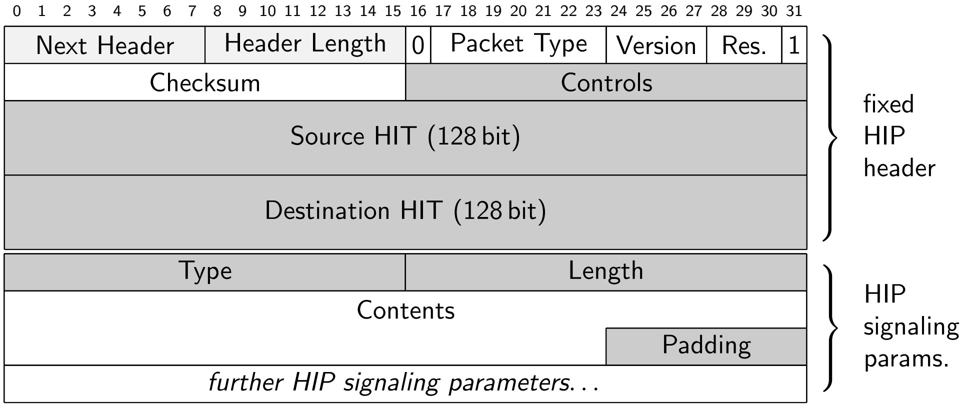


Figure 4.25 The HIP message structure consists of a fixed protocol header and a number of HIP signaling parameters. Each signaling parameter has a type-length-value encoding and includes individual parameter padding. Compressible information is marked gray. The fields marked light gray can be handled by existing compression mechanisms at the 6LoWPAN layer.

a variable number of *message-type-dependent HIP signaling parameters* that carry the actual signaling information. In the following sections, we describe these two HIP message components in more detail and identify dispensable protocol information that should be removed or compressed before transmission inside a constrained node network. We thereby distinguish between the following three types of protocol information: (i) semantically irrelevant or statically defined information that should be *elided*, (ii) variable, redundant information that should be represented in a *compressed* form, and (iii) information that cannot or should *not be compressed.*

4.2.1.1 The Fixed HIP Protocol Header

The HIP header logically[7] is designed as an IPv6 extension header [KWK$^+$12]. Thus, it starts with the mandatory *Next Header* and *Header Length* fields and requires an 8 byte alignment of the entire HIP message content (see Figure 4.25). The HIP-specific part of the protocol header begins with information that is required to identify the specific message parsing routines at a receiving communication end-point, i.e., the *Packet Type* and the protocol *Version* field. The next three bits are currently unused. Moreover, the two fixed bits in the HIP header that enclose the *Packet Type* and the three reserved bits are set statically to guarantee wire-format interoperability with the Shim6 protocol[8] [NB09]. The HIP header continues with a *Checksum* for early message verification as well as a *Controls* field. The main purpose of this latter header field is to enable a communication end-point to convey further information about the message structure. The HIP header ends with a HIT address pair that affords identification of the handshake peers at the HIP layer.

Conceptually, HITs are a type of an Overlay Routable Cryptographic Hash IDentifier (ORCHID) [LD14]. An ORCHID, in turn, maps a cryptographic identifier, i.e., the public key of a communication end-point in case of HIP, into a dedicated subnet of the IPv6 address space. This affords HITs to be used in combination with IPv6-capable applications that are unaware of the underlying HIP protocol layer [HNK08]. Following the specification of an ORCHID, a HIT consists of the following three components: (i) a fixed 28 bit IPv6 prefix that allows to distinguish HITs from standard IPv6 addresses, (ii) a 4 bit ORCHID generation algorithm identifier, and (iii) a 96 bit public-key representation that is derived via the indicated ORCHID generation algorithm. For HIP DEX, the ORCHID generation algorithm is specified as the left-truncation of a communication end-point's public ECDH key[9].

Dispensable protocol information:

While allowing to differentiate HITs and IPv6 addresses at the HIP layer and above, the HIT prefix constitutes static information regarding the HIP wire-format as the

---

[7] It is worth noting that, although adhering to the requirements of an IPv6 extension header, HIP protocol messages can also be conveyed over IPv4 links.

[8] This specification appears to be for historic reasons. For IPv6 implementations adhering to [KWK$^+$12], HIP and Shim6 protocol messages can uniquely be identified based on the protocol number in the *Next Header* field of the preceding IPv6 (extension) header.

[9] In the latest protocol revision [MH14], the public ECDH key is folded into 96 bit via an iterated application of an XOR function. This modification, however, does not affect our later assertion that HIP DEX currently defines only a *single* ORCHID generation algorithm.

HIP header exclusively carries HITs in its source and destination fields. Thus, the HIT prefix can be *elided* from the source and the destination HIT in the HIP header of all protocol messages without causing ambiguities. Similarly, HIP DEX currently only supports a single ORCHID generation algorithm. This also renders the corresponding identifier *elidable* for all protocol messages. Furthermore, the R1 and I2 handshake messages contain the sender's public ECDH key (see Figure 3.7). Hence, the public-key representation of the source HIT constitutes redundant information in case of these two handshake messages as the ECDH key allows to compute the public-key representation of the source HIT. Consequently, the entire source HIT can be *elided* for the transmission of the R1 and I2 handshake messages.

Concerning the remaining HIP header, the mandatory IPv6 extension fields, i.e., the *Next Header* and the *Header Length*, can be handled with existing 6LoWPAN compression mechanisms [HT11]. Moreover, the HIP DEX specification currently only defines the highest order bit of the *Controls* field. This bit indicates that the exchanged public key is anonymous and, thus, should not be stored by the handshake peer. As the memory overhead of maintaining multiple anonymous identities likely is excessive for constrained devices, the *Controls* field may often be empty. An empty *Controls* field, however, can be *elided* for transmission purposes. Importantly, the message parsing information, i.e., the *Packet Type* and the protocol *Version*, should remain *uncompressed*. This is to allow for immediate identification of retransmitted protocol messages and to afford an early checksum-based message verification without the need for prior message decompression. Still, we note that the *Checksum* has to be computed over the compressed message content instead of the original HIP DEX message in order to afford such early message verification.

4.2.1.2 The Message-Type-Dependent HIP Signaling Parameters

As illustrated in Figure 4.25, the general HIP parameter layout employs a Type-Length-Value (TLV) encoding. This encoding enables a protocol implementation to identify its supported signaling parameters based on the parameter type value. Unsupported or unneeded parameters then can simply be skipped by advancing the message parsing offset according to the length indication of the skipped parameter. Moreover, each signaling parameter includes its own padding information. This is to guarantee the 8 byte alignment that is required for an IPv6 extension header.

For HIP messages with multiple parameters, the HIP message wire-format additionally specifies the order of the included signaling parameters. Specifically, parameters must be arranged in an ascending order based on their parameter type number. This well-defined parameter order, e.g., allows to separate a HIP message into an integrity protected and an unprotected part. The latter is especially useful when aiming to add further signaling information on the communication path (e.g., see our newly introduced VIA_UNTRUSTED_NETWORK parameter in Section 4.1.3.3).

The HIPv2 and HIP DEX specifications employ the above message wire-format to define a limited set of *mandatory* parameters that all handshake peers must understand to establish or to tear down a connection context. Protocol extensions then specify further, *optional* parameters that allow for additional protocol functionality.

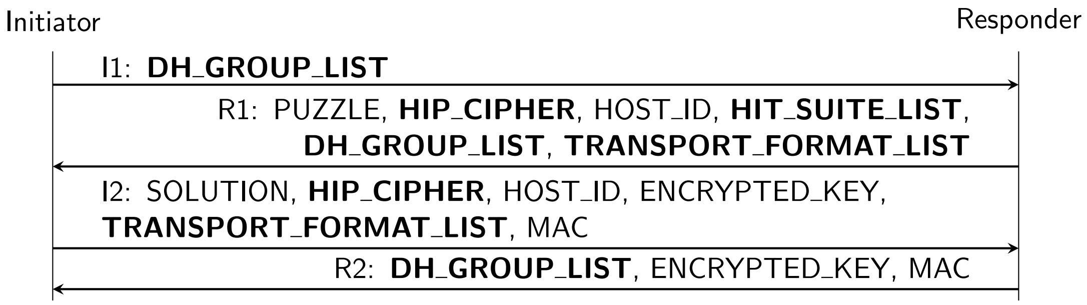

Initiator
Responder
I1: **DH_GROUP_LIST**
R1: PUZZLE, **HIP_CIPHER**, HOST_ID, **HIT_SUITE_LIST**, **DH_GROUP_LIST**, **TRANSPORT_FORMAT_LIST**
I2: SOLUTION, **HIP_CIPHER**, HOST_ID, ENCRYPTED_KEY, **TRANSPORT_FORMAT_LIST**, MAC
R2: **DH_GROUP_LIST**, ENCRYPTED_KEY, MAC

Figure 4.26 Detailed sequence diagram of the HIP DEX handshake. HIP signaling parameters that are marked bold contain negotiation information that can be compressed. The remaining signaling parameters mainly contain random data with a low compression potential.

Dispensable protocol information:

Although provided to ensure the 8 byte alignment of the HIP message content, the semantically irrelevant parameter padding unnecessarily adds up to 7 byte per parameter to the overall message size. This padding, therefore, should be *elided* during message transmission. Moreover, the length information of mandatory parameters with a static content size denotes redundant information. This is because all protocol implementations can derive the length of these parameters via the protocol specification. Similarly, the type field of the mandatory parameters constitutes redundant information for protocol messages that do not contain any optional parameters. The reason for this is that the order of the mandatory parameters then is well-defined. Hence, the type and the length information of the mandatory parameters should typically be *elided* to achieve an efficient message transmission (see Figure 4.25).

In contrast to mandatory parameters, the occurrence of optional parameters varies for the same protocol message depending on the employed protocol extensions. Thus, the type indication of optional parameters constitutes important protocol information for the correct parameter parsing and must *never be elided*. Similarly, optional parameters are not necessarily supported by all handshake peers. Hence, the length indication of optional parameters likewise is *unavoidable* in order to enable handshake peers to skip over unsupported parameters in a received message.

Concerning the value of the TLV-encoded HIP signaling parameters, we observe that the actual parameter content primarily consists of structured information for protocol parameter negotiation or random data for the employed cryptographic mechanisms. Random data, however, only has a negligible compression potential. We, therefore, focus on the compressibility of the protocol parameter negotiation.

As shown in Figure 4.26, the protocol parameter negotiation is primarily related to the agreement on mutually supported cipher suites during the protocol handshake. In the context of the IoT, we expect these cipher suites to be restricted to a small set of cryptographic primitives and cipher modes due the limited amount of RAM and ROM on constrained devices and the memory overhead that each additional cipher suite incurs. For this reason, the HIP DEX specification, e.g., only defines a single value for the HIP_CIPHER parameter, i.e., the cipher ID AES-128-CTR. Consequently, well-known default values in the corresponding negotiation parameters can typically be *compressed* before transmission. Still, such content-specific compression must not just be limited to the status quo regarding the supported cipher suites. In-

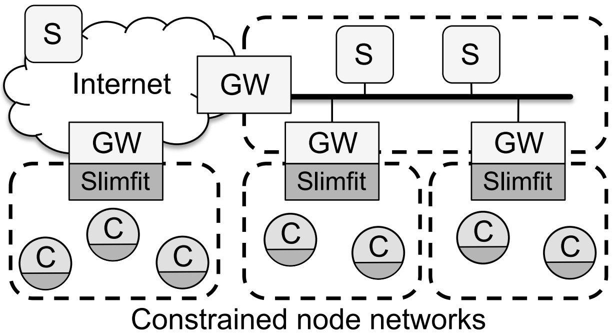


Figure 4.27 Constrained devices (C) communicate with each other and with local or Internet-based services (S) via a gateway (GW). Only network entities that belong to a constrained node network are equipped with our Slimfit compression layer (marked dark in gray). All other network entities can remain oblivious to the Slimfit-compressed HIP message wire-format.

stead the compression of the cipher suite negotiation parameters has to evolve with future security recommendations to ensure continuous HIP message compressibility.

4.2.2 The Slimfit Compression Layer

As we showed in Section 4.2.1, the HIP message wire-format still contains several elements that constitute dispensable protocol information in the context of the IoT. Hence, to tailor the HIP message wire-format to IoT requirements, we now present the design of our Slimfit compression layer. We thereby primarily focus on the HIP DEX protocol. Still, we also tackle the applicability of Slimfit for HIPv2.

The design of Slimfit aims at affording a limited deployment at constrained devices and interconnecting gateways that belong to a specific constrained node network as depicted in Figure 4.27. This allows communication end-points and on-path network elements that are located outside a constrained node network to remain oblivious to the Slimfit-compressed HIP wire-format due to a message decompression at an interconnecting gateway. Moreover, this design trait also affords an incremental deployment of Slimfit on a per-network basis. Still, it is important to note that the devised compression mechanisms also allow for an end-to-end deployment of Slimfit.

We now continue with a description of how Slimfit integrates with the HIP DEX message processing at the handshake peers as well as at the interconnecting gateways. We then present the specific compression mechanisms, which Slimfit employs to reduce the size of the HIP protocol header and the HIP signaling parameters.

4.2.2.1 Integration of Slimfit in the Network Stack

As illustrated in Figure 4.28, the devised HIP message wire-format compression mechanisms are bundled in a new layer of the network stack that is located between the network and the HIP DEX protocol layer. This allows to transparently integrate Slimfit with an unmodified HIP DEX implementation depending on the underlying link layer technology. This separation, however, also implies that outbound HIP messages must be redirected through the Slimfit compression layer before they are processed at the network layer. For a constrained device, this redirection

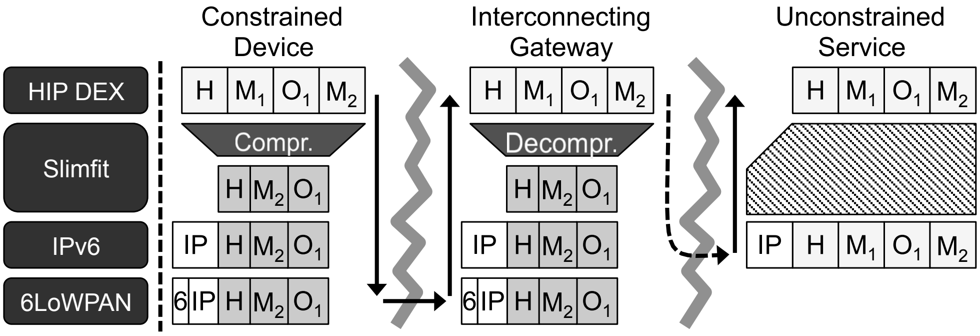


Figure 4.28 Integration of the Slimfit compression layer with an existing network stack. Arrows indicate the processing flow for a HIP DEX message with an interconnecting gateway. Slimfit compresses the HIP header (H), reorders mandatory (M_i) and optional (O_j) parameters, and compresses or elides these parameters. The dashed arrow indicates that the HIP DEX message skips the Slimfit and the 6LoWPAN adaptation layers for message transmission.

can typically be realized by inserting a function call to our compression layer at the corresponding code point in the embedded network stack. A similar integration can also be achieved for most commodity operating systems by hooking into their network stacks via dedicated networking facilities such as *netfilter* [netfilter]. Slimfit then compresses outbound HIP messages and informs the communication partner and on-path network entities that the wire-format has changed by indicating the employed compression mechanisms in the unoccupied bits of the *Controls* field of the HIP header as shown in Figure 4.29. After a successful message compression, Slimfit passes the adapted HIP message down the network stack for further processing.

When the compressed HIP message reaches the 6LoWPAN adaptation layer in the network stack of the message sender, the LOWPAN_NHC mechanism (see Section 2.4.1) performs an additional compression step on the traversing message. More precisely, it removes the need for an 8 byte message alignment from the HIP message content by indicating that the *Header Length* field of the HIP header was computed as a multiple of 1 byte. Hence, this step enables the removal of the per-parameter padding information from the original HIP message wire-format at our Slimfit layer.

To indicate the modified semantics of the *Header Length* field, the LOWPAN_NHC mechanism prepends a 1 byte compression header to the adapted HIP message content [HT11]. This compression header includes an Extension Header ID (EID) that signals the type of the subsequent IPv6 extension header. We propose to use one of the free EIDs, e.g., 6, to indicate the Slimtfit-compressed message wire-format. The sender finally transmits the compressed HIP message without further modification.

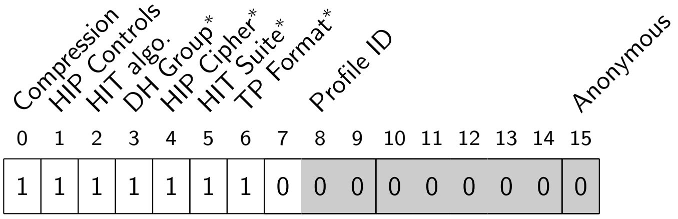


Figure 4.29 The compression bits set in the Controls field of the HIP header indicate a HIP DEX message with maximum compression. Compression flags for negotiation parameters are marked with an asterisk. Elidable bits are marked gray. Bits 10 to 14 remain unused.

Upon reception of a compressed HIP message, a device that *forwards* this message inside a constrained node network does not need to perform any Slimfit-related operations. This is because a mere message forwarder commonly does not process the message content above the network layer. However, once a compressed HIP message reaches an interconnecting gateway that bridges the constrained node network with a conventional IP network, Slimfit decompresses the received HIP message (see Figure 4.28). To this end, our Slimfit layer has to hook into the network layer at the gateway, e.g., via *netfilter* facilities. Slimfit then determines the compression mechanisms that were applied to the received HIP message via the *Controls* field and runs their decompression counterparts. Similarly, when the handshake peer receives a compressed HIP message, Slimfit decompresses this message according to the indicated compression mechanisms before passing it up to the HIP DEX layer.

4.2.2.2 Compression of the HIP Protocol Header

As highlighted above, our Slimfit compression layer leverages the 2 byte *Controls* field in the HIP header to indicate the compression mechanisms that were applied to the original HIP message. More precisely, Slimfit uses the first bit of this header field to signal an active HIP message compression. It then always elides the static prefix of the source and the destination HIT as well as the public-key representation of the source HIT for protocol messages that also carry the sender's public key. Moreover, Slimfit elides the second byte of the *Controls* field if it is unused in the processed protocol message (see Figure 4.29). The Slimfit layer indicates this compression of the original *Controls* field in the second bit of the compressed *Controls* field. We note that, contrary to our analysis in Section 4.2.1.1, the first byte of the *Controls* field cannot be elided. This is because Slimfit uses this byte for signaling purposes.

Finally, Slimfit also elides the ORCHID generation algorithm identifier of the source and the destination HIT if they contain the HIP DEX-specific HIT suite ID, i.e., 8. Our compression layer indicates this elision via the third bit of the *Controls* field. Importantly, the ORCHID generation algorithm identifier would no longer be elidable if future HIP DEX protocol specifications defined an additional HIT suite ID and if this HIT suite ID was used in the protocol message. To address this issue, we extend Slimfit with the possibility to define compression profiles in Section 4.2.2.4.

4.2.2.3 HIP Signaling Parameter Compression

Regarding the HIP signaling parameters, we further distinguish between the compression of the *general* HIP parameter fields (i.e., the type, length, and padding information) and the *content-specific* compression of the HIP negotiation parameters. We now first discuss how Slimfit compresses the general HIP parameter fields and then continue with the content-specific HIP negotiation parameter compression.

General Parameter Compression

As we discussed in Section 4.2.1.2, the type and length information of the HIP signaling parameters can commonly be elided if the entire HIP message only consists

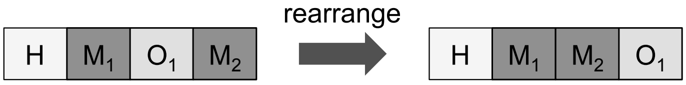


Figure 4.30 Slimfit rearranges the mandatory (M_i) and the optional (O_j) parameters in a HIP message to enable the elision of the type and length information of mandatory parameters.

of mandatory parameters. To also achieve this elision for messages that additionally contain optional parameters, our Slimfit compression layer first reorders the included HIP signaling parameters of an uncompressed HIP message. As depicted in Figure 4.30, Slimfit thereby breaks the parameter-type-dependent order of the standard HIP wire-format and moves the optional parameters behind the mandatory ones. Importantly, this reordering step must not change the relative position of the mandatory parameters towards each other to allow a receiving end-point to unambiguous identify these parameters for a specific protocol message[10] based on the order in their protocol specification, i.e., even when their parameter type is elided. In contrast, the order of the trailing optional parameters is irrelevant for the following compression steps as the type field of optional parameters remains present.

After the parameters have been rearranged successfully, the Slimfit layer continues by stripping the padding information from *all* parameters of the rearranged HIP message. Moreover, it elides the type field from all mandatory parameters and removes the length information from the fixed-length mandatory parameters. We note that this general parameter compression is unambiguously reversible by performing the outlined steps in reverse order. Finally, there is no need for Slimfit to specifically signal this general parameter compression via the *Controls* field as this type of parameter compression always is performed when compressing HIP messages.

Content-Specific Compression of the Negotiation Parameters

Concerning the content-specific HIP parameter compression, our Slimfit layer focuses on the four mandatory DH_GROUP_LIST, HIP_CIPHER, HIT_SUITE_LIST, and TRANSPORT_FORMAT_LIST parameters of the connection establishment handshake (see Figure 4.26). Specifically, Slimfit compresses these parameters if they contain default information. Slimfit then represents the entire parameter as a single bit in the *Controls* field of the HIP header (see the elided parameter M_1 in Figure 4.28 and the *Controls* bits marked with an asterisks in Figure 4.29). We now briefly discuss our choice of the specific compressible negotiation parameter content and refer to Section 4.2.2.4 for a description of how additional compression profiles afford compression evolvability with the goal to address deviating parameter content.

The DH_GROUP_LIST parameter contains a list of well-defined DH group IDs that are supported by the message sender. In case of HIP DEX, we observe that these DH groups are limited to elliptic curve cryptography. Moreover, the most probable ECDH groups conveyed in this parameter can further be narrowed down by considering recent security recommendations [NIST12a]. Specifically, the elliptic curve NIST P-256 is the elliptic curve with the smallest field size that is currently recommended for key management purposes and that also is supported by HIP DEX. Moreover, this curve also is assumed to be secure for use until 2030 [NIST12a].

---

[10] We recollect that the uncompressed packet type is conveyed in the HIP header (see Figure 4.25).

Hence, we expect constrained devices to predominantly employ this elliptic curve during the HIP DEX handshake and compress the DH_GROUP_LIST parameter if it indicates the corresponding DH group ID. For the remaining three protocol negotiation parameters, the HIP DEX protocol specification currently only defines a single valid suite ID. Thus, we declare these well-known suite IDs as the default parameter content for our Slimfit layer and compress matching parameters accordingly.

4.2.2.4 Compression Evolvability

The Slimfit compression layer provides both general and content-specific HIP message compression mechanisms. The general compression mechanisms always are applicable for a HIP message unless future protocol iterations fundamentally change the HIP message wire-format. The content-specific mechanisms, however, only apply as long as the ORCHID generation algorithm identifier in the HIT pair and the content of the protocol negotiation parameters equal our selected defaults. Our choice of these defaults, however, will not hold indefinitely and may also be scenario-specific.

To enable evolvability of our selected defaults, we extend the Slimfit compression layer with the ability to specify compression profiles. Each profile defines a compressible ORCHID generation algorithm identifier along with compressible negotiation parameter content similar to our definitions above. As depicted in Figure 4.29, Slimfit then indicates the employed compression profile ID in the *Controls* field.

We note that this signaling of a compression profile ID prevents the elision of the second byte of the *Controls* field in the HIP header. As a result, the use of compression profiles causes a slight increased in transmission overhead. This additional transmission overhead, however, is compensated by the compressibility of protocol information that otherwise could not be considered for compression purposes.

Importantly, compression profiles and their IDs do not need to be globally unique when Slimfit is deployed on the intended per-network and not on an end-to-end basis. Each constrained node network then can define its own compression profile(s) in order to achieve the optimal compression ratio based on the specific exchanged protocol parameter negotiation information. The constrained devices and the interconnecting gateways of each constrained node network, however, still must have a common understanding of the employed compression profiles and their IDs. To this end, we propose to disseminate Slimfit compression profiles via the adapted IPv6 Neighbor Discovery mechanism for constrained devices similar to the Context Option of the LOWPAN_IPHC header compression mechanism (see Section 2.4.1).

4.2.2.5 Applicability of Slimfit for the HIPv2 Protocol

Besides addressing message conciseness for HIP DEX, Slimfit also allows to decrease the transmission overhead of the HIPv2 protocol. This is especially beneficial in the context of computationally unconstrained devices that employ constrained link layer technologies and that employ HIPv2 for secure end-to-end communication.

Regarding the applicability of Slimfit for the HIPv2 protocol, we recollect that HIPv2 and HIP DEX share the same HIP message wire-format (see Section 4.2.1). Moreover, both protocols define mandatory as well as optional parameters for their specified message exchanges. Thus, the presented *general* Slimfit compression mechanisms for the HIP header and for the HIP signaling parameters also apply to HIPv2. Both protocols, however, considerably differ with respect to their intended deployment scenarios. While HIP DEX focuses on constrained devices, HIPv2 was originally designed to secure communication between comparably powerful Internet hosts. As a result, the cryptographic primitives employed in HIPv2, and thus the content of the corresponding negotiation parameters, often may differ from our above defaults.

To achieve a similar compressibility of the HIP message wire-format in case of HIPv2 as for HIP DEX, we propose to leverage the previously introduced Slimfit compression profiles to adapt the content-specific compression mechanisms of our Slimfit layer to HIPv2 requirements. These profiles then most notably must cover the potential use of traditional public-key cryptography such as RSA as the main difference between HIPv2 and HIP DEX regarding the provided cryptographic primitives.

4.2.3 Evaluation

For our evaluation, we extended the embedded network stack of Contiki OS [DGV04] with the required functionality for the presented Slimfit compression layer and leveraged our HIP DEX protocol implementation (see Section 4.1.6) for all protocol interactions at the HIP DEX layer. As in the previous evaluation, Zolertia Z1 motes[11] then served as the underlying hardware platform for constrained devices. Hence, constrained devices in our evaluation setup were equipped with a 16 MHz MSP430 MCU, 8 kB of RAM, 92 kB of ROM, and an IEEE 802.15.4 radio interface.

In addition to these implementations, we also realized a simple Python application that compresses captured HIP DEX message traces with the generic *LZ77* [ZL77], *LZSS* [SS82], and *LZMA* [Pav13] compression algorithms as well as with the *zlib* compression library [zlib], which implements the *DEFLATE* algorithm [Deu96]. This Python application provides a baseline for our evaluation by allowing to compare the achieved compression ratios of the protocol-specific Slimfit compression layer to the compression ratios of alternative, protocol-independent compression algorithms.

Lastly, we considered the same security-related link layer overheads as described in Section 4.1.6. Hence, the discussed 6LoWPAN fragmentation thresholds also apply to our evaluation setup in this section. Despite these similarities, this evaluation setup, however, differs from our previous setup concerning the following two aspects. First, Contiki OS was updated to version 2.6. Second, the public-key operations followed recent security recommendations by employing elliptic curve NIST P-256.

4.2.3.1 Transmission Overhead

To quantify the transmission overhead of our Slimfit compression layer, we first measured the individual message sizes of a *standard* connection establishment handshake and of a subsequent connection teardown exchange at the HIP DEX protocol layer for two wirelessly connected Zolertia Z1 motes. A comparison with the corresponding

---

[11] We refer to Section 2.2.1 for a brief overview and a comparison of the different experimentation platforms for constrained devices that we use throughout the course of this thesis.

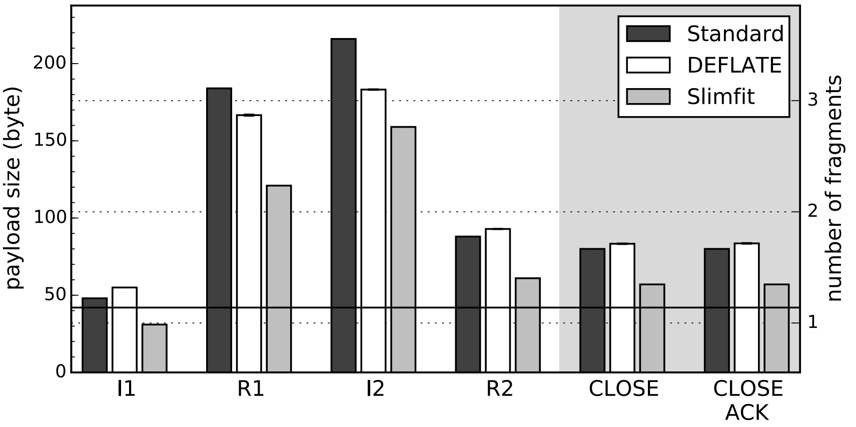


Figure 4.31 HIP message size for the HIP DEX connection establishment handshake and a subsequent connection teardown exchange (marked with a gray background) for the *standard* HIP DEX protocol, with *DEFLATE* compression, and with our *Slimfit* compression layer. The continuous line indicates the 6LoWPAN fragmentation threshold in our evaluation setup. The dashed lines then indicate the maximum payload size of the individual 6LoWPAN fragments.

Slimfit-compressed HIP messages then allowed us to determine the achieved compression ratio. Moreover, we compared these results to the compression ratio of generic compression algorithms when applied to the same HIP message content. To this end, we ran our Python application with the *LZ77*, *LZSS*, *LZMA*, and *zlib* compression mechanisms on a captured HIP message trace that consisted of 100 HIP DEX connection establishment handshakes and connection teardown exchanges. Each algorithm was set to the highest available compression level. Regarding the considered generic compression mechanisms, *zlib* achieved the best overall compression ratio and, therefore, serves as the focus of the following comparison.

As shown in Figure 4.31, the Slimfit compression layer is able to notably reduce the size of *all* HIP messages that are transmitted during the HIP DEX connection establishment handshake and the connection teardown exchange. Furthermore, Slimfit *consistently* outperforms the *zlib* compression mechanism. More precisely, Slimfit achieves a compression ratio that ranges from 1.36 for the I2 message as the worst case to 1.55 for the I1 message as the best case. In contrast, *zlib* exhibits a considerably lower compression ratio of 1.18 in its best case and even adds a small overhead of up to 7 bytes to short HIP DEX handshake messages (see the I1 and R2 messages as well as the connection teardown exchange in Figure 4.31). This is primarily due to the constant 11 byte header overhead of the zlib compressed data format [DG96].

Moreover, the *zlib* approach showed a maximum standard deviation of 0.69 byte when applied to the captured HIP message trace. This non-zero standard deviation is evidence for the dependency of the underlying *DEFLATE* compression algorithm on the actual HIP message content. The evaluation results for Slimfit, contrarily, exhibited a standard deviation of zero. This circumstance demonstrates that our Slimfit compression layer always achieves the above compression ratios as long as the protocol negotiation parameters contain the previously specified defaults.

Concerning the effective transmission overhead of the HIP DEX protocol, Slimfit reduces the overall size of the connection establishment handshake from 536 byte to 372 byte. This is an overhead reduction by 30.60 %. In contrast, the generic *zlib* compression mechanism only achieves an overhead reduction to 497.74 byte (i.e., by

7.14 %) on average. In case of the connection teardown exchange, Slimfit similarly decreases the transmission overhead from 160 byte to 114 bytes, i.e., an overhead reduction by 28.75 %. The *zlib* mechanism, instead, increases the transmission overhead by about 6.9 byte (i.e., 4.31 %). Hence, we conclude that our Slimfit compression layer substantially outperforms the considered generic compression algorithms.

6LoWPAN message fragmentation and impact on HIP DEX retransmissions

As the message fragmentation results in Figure 4.31 show, Slimfit-compressed HIP messages lead to fewer packet transmissions at the lower layers in the network stack compared to a standard HIP DEX connection establishment handshake. Specifically, Slimfit decreases the 6LoWPAN message fragmentation by a total of 3 fragments during the HIP DEX protocol handshake (see "I1", "R1", and "I2" packets in Figure 4.31). This constitutes a reduction by 25 % with respect to the overall number of transmitted link layer frames transmission. Consequently, our Slimfit compression layer also considerably improves the frame-wise utilization of the wireless medium.

We observe that this reduction in message fragmentation at the 6LoWPAN layer also has a positive impact on the transmission overhead that is caused by the HIP DEX retransmission mechanism. This is because already the loss of a single 6LoWPAN fragment results in the loss of an entire HIP DEX message due to the best effort 6LoWPAN fragment transmission semantics and the fate-sharing between the 6LoWPAN fragments of a HIP DEX message. Thus, the loss probability at the HIP DEX layer decreases with a reduced number of 6LoWPAN message fragments.

To further quantify this aspect, we measured the overall transmitted bytes until the successful conclusion of a HIP DEX connection establishment handshake, i.e., including retransmitted handshake messages, between two directly connected nodes in the Cooja network simulator [ODE$^+$06]. We varied the *end-to-end* loss probability from 0 % to 80 % on a per 6LoWPAN fragment basis and performed 5000 HIP DEX protocol handshakes for each of these loss probability. Similar to our evaluation in Section 4.1.6.3, we decided for simulation over a real testbed to analyze the impact

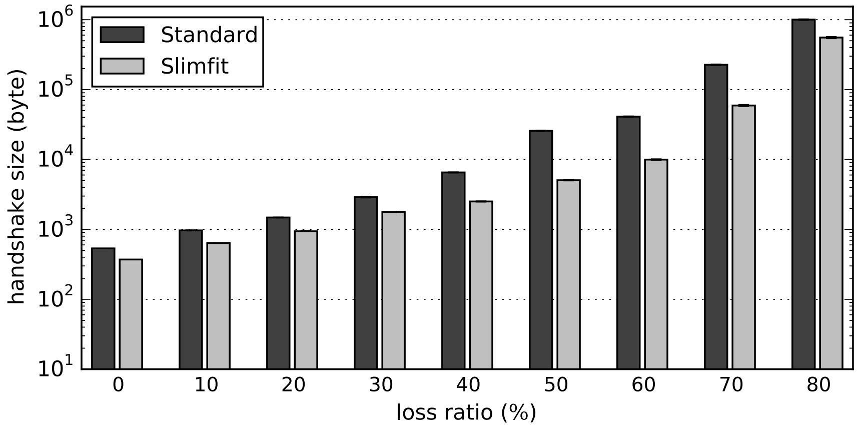


Figure 4.32 Transmission overhead of the standard *HIP DEX* and the *Slimfit*-compressed connection establishment handshake for different loss probabilities. The error bars depict the 99% confidence intervals. Note the logarithmic scale of the y-axis. The Slimfit-compressed handshake consistently has a lower transmission overhead than standard HIP DEX.

of our Slimfit compression layer on the HIP DEX retransmission mechanism for well-defined loss probabilities without side-effects on the wireless medium.

As shown in Figure 4.32 (note the logarithmic scale), the decreased 6LoWPAN message fragmentation with our Slimfit compression layer considerably reduces the amount of HIP DEX retransmissions during the connection establishment handshake. The average transmission overhead of a Slimfit-compressed handshake, e.g., constitutes about 63.5 % of a standard HIP DEX handshake in case of a loss probability of 20 %. Importantly, this percentage further improves for rising fragment loss probabilities. Hence, our Slimfit compression layer not only substantially reduces the size of the individual HIP DEX messages, but also considerably improves the HIP DEX protocol performance in the context of lossy network environments.

4.2.3.2 Processing Overhead

To evaluate the computational overhead of the Slimfit message compression, we analyzed the processing time of the HIP DEX connection establishment handshake and of the connection teardown exchange with two wirelessly connected Zolertia Z1 motes. In this, we specifically focused on the computation overhead (i) at the HIP DEX protocol layer, (ii) at our Slimfit compression layer, (iii) and at the lower layers in the Contiki OS network stack. For the lower layers of the network stack, we restricted our evaluation to the processing time of *outbound* HIP DEX messages in order to prevent overhead misattribution of inbound 6LoWPAN fragments that, e.g., belong to the RPL protocol. *Inbound* HIP DEX messages, thus, cause additional computation overhead at the lower layers of the network stack that is not depicted in Figure 4.33. The results presented below denote the average over 100 measurement runs. The standard deviation of these runs was below 23.09 ms (i.e., 1.25 %) for the public-key-based operations and below 0.09 ms for all remaining operations.

Curiously, we found that our Slimfit layer does *not* result in a computation penalty. Instead, Slimfit even leads to a modest *performance gain* that amounts to about 6.58 ms for the connection establishment handshake and to about 0.83 ms for the

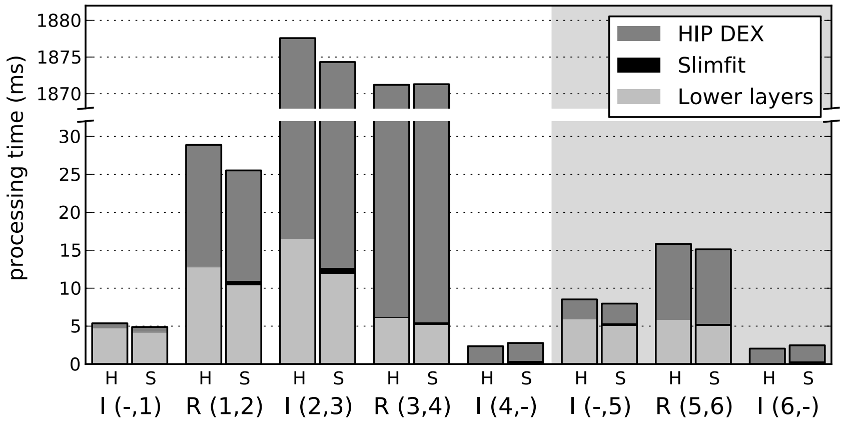


Figure 4.33 Message processing time for the connection establishment and the connection teardown (marked with a gray background) for the standard HIP DEX protocol (H) and with our Slimfit compression layer (S) for the Initiator (I) and the Responder (R). The numbers in brackets denote processing of the i-th and generation of the (i+1)-th handshake message.

connection teardown exchange. As depicted in Figure 4.33 (see “Lower layers”), this performance gain mainly stems from a reduced computation overhead at the lower layers of the network stack for Slimfit-compressed HIP DEX messages.

During our detailed analysis of this phenomenon, we observed that the decreased number of 6LoWPAN fragments for Slimfit-compressed HIP DEX messages caused a reduced computation overhead at the 6LoWPAN adaptation layer. Similarly, the link layer had to handle fewer IEEE 802.15.4 frames for tasks such as transmission scheduling and carrier sensing. The corresponding performance gains amounted to 8.61 ms for the complete HIP DEX connection establishment handshake and to 1.67 ms for the connection teardown exchange. We also observed these performance gains when manually inspecting a small number of *inbound* HIP DEX messages.

The message compression and message decompression operations at the Slimfit layer, in contrast, only incurred a small computation overhead. This overhead amounted to a minimum of 0.12 ms for the compression of an I1 message and a maximum of 0.73 ms for the decompression of an R1 message and the compression of the subsequent I2 response (see “I (-,1)” and “I (2,3)” in Figure 4.33). The performance gain at the lower layers of the network stack, thus, outweighed the low computation overhead of our Slimfit layer and resulted in an overall performance gain.

To summarize, our Slimfit compression layer not only decreases the transmission overhead and the 6LoWPAN fragmentation of HIP DEX messages, but also decreases the message processing time when considering the entire network stack.

4.2.3.3 RAM and ROM Overhead

To derive RAM and ROM estimates for our Slimfit compression layer in the context of constrained devices, we analyzed the two Contiki OS binaries that we employed during our evaluation with the *msp430-size* tool, which is part of the GCC toolchain for the MSP430 MCU [mspgcc]. The first binary contained an unmodified Contiki OS with our HIP DEX protocol implementation. The second binary additionally included the implementation of our Slimfit compression layer. We note that the presented evaluation results do not include support for compression profiles as we focused our evaluation on handshakes with well-defined parameter defaults.

Most importantly, the unchanged (static) RAM requirements of our Slimfit compression layer shown in Table 4.3 indicate that our developed HIP message compression mechanisms indeed are stateless. This claim is further substantiated by the fact that our implementation of the Slimfit compression layer does not use dynamic memory allocation, e.g., via *malloc*, in order to maintain persistent run-time compression

Functionality	ROM (byte)	RAM (byte)
Contiki OS incl. HIP DEX	58659	7624
+ Slimfit compression layer	61157 [+2498]	7624 [+0]

Table 4.3 RAM and ROM requirements for our presented Slimfit compression layer in byte. Numbers in brackets denote added overhead compared to an unmodified Contiki OS with our HIP DEX protocol implementation. Notably, our Slimfit compression layer can be realized with about 2.5 kB of ROM and with no additional (static) RAM requirements.

information. Still, it is worth noting that Slimfit requires additional RAM resources when employing dynamically configured compression profiles. These RAM resources then, however, denote a tradeoff for an improved HIP message compression ratio.

Moreover, our Slimfit compression layer causes a ROM overhead of about 2.5 kB as shown in Table 4.3. This overhead most notably includes the presented Slimfit compression mechanisms along with a static definition of the discussed defaults for the HIP DEX negotiation parameters. In fact, this modest memory overhead is the only tradeoff of the Slimfit compression layer that we could identify in our evaluation.

4.2.4 Security Considerations

We now briefly discuss potential attacks that an adversary can mount against the Slimfit compression layer based on the Internet Threat Model (see Section 3.1.1).

DoS protection properties of HIP DEX with the Slimfit compression layer

The HIP DEX protocol design incorporates two complementary DoS protection mechanisms to defend the Responder against attacks that target the computation and the memory overhead of the protocol handshake (see also Section 4.1.3.1). First, the Responder can delay the creation of session state until the I2 handshake message has been verified successfully. As a result, the Responder only needs to store session state for correctly authenticated Initiators. Second, HIP DEX employs a puzzle-based DoS protection mechanism that enables the Responder to demand an adjustable resource commitment from the Initiator. This allows the Responder to only invest own resources for the processing of a received I2 message *after* the Initiator has committed to the handshake by solving a Responder-defined puzzle.

The presented Slimfit compression layer does *not* alter these DoS protection properties of the HIP DEX protocol as a Slimfit-enabled Responder can still remain stateless until the Initiator has been authenticated successfully. This is because Slimfit only applies compression mechanisms that do not require the Responder to maintain connection-specific compression state across HIP messages (see Section 4.2.3.3). Moreover, Slimfit even modestly *reduces* the message processing overhead at the lower layers in the network stack (see Section 4.2.3.2). Hence, Slimfit in fact further improves the DoS protection properties of HIP DEX as the unprotected computations until the verification of the puzzle at the HIP DEX layer are further reduced.

Slimfit and attacks against on-path network entities

As discussed in Section 4.2.2.1, Slimfit does not cause additional message processing overheads at packet forwarders inside a constrained node network. Hence, it does not render these devices vulnerable to new attacks. Slimfit, however, requires gateways that bridge a constrained node network with another network domain to apply the presented compression and decompression mechanisms to traversing HIP messages.

An adversary could exploit this circumstance by flooding a gateway with standard HIP DEX protocol messages that are destined for a device inside the constrained node network. The gateway then would perform the applicable Slimfit compression operations on the traversing HIP messages. We observe that this flooding-based DoS attack would only be able to target the processing resources at the gateway. This is because Slimfit only performs *stateless* message compression operations. The computation overhead of Slimfit, however, is modest as shown in Section 4.2.3.2. Hence, the amplification effects that the adversary could exploit would be marginal. As we will show in Chapter 6, this stands in stark contrast to the achievable amplification effects in the context of the 6LoWPAN fragmentation mechanism, thus rendering general 6LoWPAN message fragmentation a considerably more attractive target.

Furthermore, network elements such as firewalls may inspect traversing HIP DEX protocol messages in order to provide additional on-path security services. However, as Slimfit introduces a new compressed HIP message wire-format, network elements that support the standard HIP message wire-format do not necessarily support its Slimfit-compressed counterpart. In this context, it is important to note that Slimfit is primarily intended for use inside a constrained node network and, thus, typically does not impact network entities that are located outside these networks. Moreover, network elements that are located inside or at the border of a constrained node network still can identify the message type, the message length, and the source as well as the destination of a Slimfit-compressed HIP DEX message. Hence, these network entities can, e.g., drop exceedingly large HIP DEX protocol messages without the need for decompression. If an on-path services still requires deeper inspection of the message content, the HIP DEX protocol message must first be decompressed. The overhead of this operation, however, is modest as shown in our evaluation.

Attacks against the Slimfit message compression operations

Data compression before encryption has been shown to leak information about the uncompressed data by allowing an adversary to infer information about the amount of redundancy in the uncompressed plaintext based on the achieved compression rate [Kel02]. Importantly, our Slimfit compression layer is not vulnerable to such side-channel attacks as it compresses the HIP message content *after* the cryptographic operations have been applied at the HIP DEX protocol layer. Thus, Slimfit does not reveal sensitive information about the uncompressed HIP message content.

Packet loss and out-of-order packets typically invalidate the compression context of stateful compression mechanisms [DHJS00]. An on-path adversary could exploit this circumstance by dropping compressed HIP DEX handshake messages and, thus, prevent the performed handshake from completing successfully. Slimfit, however, only employs stateless compression mechanisms. Consequently, Slimfit-compressed HIP DEX protocol messages are resistant to such types of attacks.

4.2.5 Related Work

For our discussion of related work, we distinguish between the following two areas of research: (i) protocol-specific compression mechanisms for the IP-based IoT and (ii) generic, standardized protocol compression schemes.

4.2.5.1 Protocol-Specific Compression Mechanisms for the IP-Based IoT

Among the first protocol standards for the IP-based IoT, the specification of the 6LoWPAN adaptation layer [MKHC07] defines stateless header compression mechanisms for the IPv6 and UDP protocols. These initial header compression techniques later were replaced with the LOWPAN_IPHC and LOWPAN_NHC mechanisms [HT11]. As a key differentiator, the LOWPAN_IPHC and LOWPAN_NHC mechanisms introduce configurable context information that also allows to compress global IPv6 addresses (see Section 2.4.1 for further details). Our Slimfit compression layer is inspired by these existing compression mechanisms with respect to the following two aspects. First, like the 6LoWPAN adaptation layer, we bundle the presented HIP DEX compression mechanisms in an intermediate layer of the network stack. This allows to separate the implementation of HIP DEX from the implementation of our Slimfit compression layer and to only include our compression functionality in network scenarios that benefit from the performed compression operations. Second, Slimfit optionally employs network-specific compression contexts similar to LOWPAN_IPHC. In contrast to LOWPAN_IPHC contexts, which only convey network- or end-point-specific IPv6 address information, our introduced contexts, however, define entire compression profiles that afford evolvability and flexibility for Slimfit.

Concerning end-to-end IP security protocols for the IoT, Raza et al. recently presented Lithe, a header compression mechanism for DTLS [RTV12, RSH$^+$13]. The authors also proposed this mechanism for standardization at the IETF [RSD14]. Similarly, Raza et al. [RVJ12] published initial ideas on compressing the IKEv2 protocol handshake. However, while these compression approaches tie into the LOWPAN_NHC mechanism in the same way as our Slimfit compression layer, their work most importantly differs from ours by *not* considering compression evolvability and flexibility in their respective approach. As a result, the achieved compression efficiency strongly depends, e.g., on the employed cryptographic primitives and, thus, will degrade over time. To address this short-coming, we propose to integrate our presented compression profiles with these existing DTLS and IKEv2 compression mechanisms. Moreover, we propose to integrate the introduced general HIP parameter compression techniques with Raza et al.'s initial compression ideas for IKEv2. This would allow to further improve the compression ratio for IKEv2 as the authors so far do not consider the compression of the HIP-like IKEv2 parameter format.

Finally, there exists a rich body of research [GMSS10, RVR11, RDC$^+$11] and active standardization efforts [RDS14, MG14] that consider the compression of the Authentication Header (AH) and the Encapsulating Security Payload (ESP) for IPsec-protected payload channels. These approaches most notably improve the applicability of IPsec as the default payload protection mechanism for HIP DEX. In contrast, our work focuses on the efficient establishment of such IPsec-protected payload channels and, thus, effectively complements these related compression efforts.

4.2.5.2 Generic Protocol Compression Schemes

Several generic protocol compression schemes have been standardized for IPv4 and IPv6 as well as for several upper layer protocols. As one of these compression schemes, the IP Payload Compression Protocol (IPComp) [SMPT01] affords the application of generic compression algorithms such as *DEFLATE* to IPv4 and IPv6 packets. However, as we showed in our evaluation, such generic compression algorithms do not leverage domain knowledge about the message wire-format and, thus, typically achieve a lower compression ratio than our Slimfit compression layer.

Furthermore, the RObust Header Compression (ROHC) [SPJ10] specification defines a general and extendable framework for header compression. ROHC profiles [PS08] then specify protocol-specific compression mechanisms for protocols such as IP, UDP, and ESP. In contrast to the compression mechanisms employed in our Slimfit compression layer, ROHC profiles commonly employ stateful compression approaches with *per-flow compression contexts*. As a result, these approaches require additional context repair mechanisms that further increase the transmission overhead in lossy network environments. Lastly, the Generic Header Compression (GHC) mechanism was recently standardized as an extension of the 6LoWPAN header compression facilities [Bor14]. GHC enables the compression of generic protocol headers and header-like payloads via an LZ77-based compression algorithm. However, while specifically designed for the IP-based IoT, its generic nature prevents GHC, e.g., from rearranging the HIP message wire-format in order to realize additional compression potentials. As a result, GHC is prone to offer a significantly lower compression ratio for HIP DEX protocol messages than the presented Slimfit compression layer.

Besides these protocol compression schemes, several *protocol extensions* aim at reducing the transmission overhead of the protocol handshake. As discussed in Section 4.1.7.2, these extensions most notably include caching of static handshake information [SB01, SB02, SBR04, Lan10, ST15] and session resumption [SBR04, SZET08, DR08, ST10, HWZ^{+}13, HGS13]. Complementary to our work in this section, these mechanisms require a complete initial handshake to establish the necessary state information at the communication end-points. Moreover, delegation architectures allow to offload part of the protocol handshake to a more powerful network entity. This also reduces the handshake-related transmission overhead inside a constrained node network. For a detailed discussion of these delegation approaches, we refer to the related work section of the handshake delegation architecture in Chapter 5.

4.2.6 Summary

In this section, we analyzed the HIP DEX message wire-format for dispensable protocol information that constitutes undesirable transmission overhead in the context of constrained node networks. During our protocol analysis, we identified such dispensable information in both the fixed HIP DEX header as well as in the message-specific HIP DEX signaling parameters. Notably, the identified dispensable information not only needlessly wastes the limited resources of energy-constrained communication end-points. Instead, it also increases the message fragmentation during the protocol handshake and, thus, unnecessarily burdens the scarce transmission, processing, and energy resources of constrained devices that are located on the forwarding path.

To adapt the HIP message wire-format to IoT requirements, we devised the Slimfit compression layer that provides both general as well as content-specific HIP message compression mechanisms. Slimfit reduces the transmission overhead of this wire-format (i) by eliding message content that is statically defined in the HIP DEX protocol specification, (ii) by rearranging the HIP signaling parameters to achieve a higher compression efficiency, and (iii) by employing compression profiles to afford evolvability and flexibility of our presented compression mechanisms. Importantly, due to this latter aspect, the design of our Slimfit compression layer is not limited to the message compression of the status quo, but additionally facilitates the adaption to future security recommendations as well as to the HIPv2 protocol.

As the evaluation results show, our Slimfit layer effectively reduces the HIP DEX transmission overhead. More precisely, it achieves compression ratios between 1.36 and 1.55 depending on the protocol message. This high compression efficiency affords Slimtfit to consistently outperform generic compression algorithms. Moreover, Slimfit reduces the 6LoWPAN message fragmentation during the HIP DEX protocol handshake by 25 %. As a result, Slimfit considerably decreases the HIP DEX retransmission overhead. Interestingly, the achieved message size reduction with Slimfit also leads to a 7.41 ms computation overhead *reduction* when considering the message processing time of the entire network stack. The 2.5 kB ROM overhead is the only trade-off we identified for the observed transmission and computation gains.

To summarize, Slimfit considerably reduces the run-time-related overheads of the HIP DEX protocol and, thus, is highly beneficial for constrained devices.

4.3 Conclusion

In this chapter, we analyzed the run-time properties of the DTLS, HIP DEX, and Minimal IKEv2 protocols for their adequacy in the IP-based IoT and identified several design-level efficiency and security issues that significantly hamper the applicability of these protocols in the context of constrained devices. To tailor DTLS, HIP DEX, and Minimal IKEv2 to IoT requirements, we presented four complementary protocol extensions. Combined, these extensions tailor the computation and transmission properties of the HIP DEX protocol in particular and of end-to-end IP security in general to the special device and network characteristics in the embedded domain as well as to the high resource asymmetry in the IoT. Still, the evaluation results also indicate non-negligible memory requirements when employing standard end-to-end IP security in the embedded domain. In the following chapter, we will show how to address this issue with respect to highly memory-constrained devices.

5

Delegating the Protocol Handshake

In the previous chapter, we presented four protocol extensions that address the high run-time overheads, i.e., the computation and transmission requirements, when employing DTLS, HIP DEX, or Minimal IKEv2 for secure end-to-end communication in the IP-based IoT. Still, our evaluation of HIP DEX also revealed non-negligible memory overheads in the context of constrained devices. These overheads amount to about 7.7 kB of RAM and 58.7 kB of ROM for the underlying operating system and a HIP DEX protocol implementation (see Sections 4.1.6.4 and 4.2.3.3). Such high memory requirements render a *complete* protocol implementation infeasible for a wide range of memory-constrained devices that, e.g., have similar memory limitations as the TelosB platform [PSC05] with 10 kB of RAM and 48 kB of ROM.

Our contributions in this chapter are threefold. First, we present the results of our detailed RAM and ROM analysis of a DTLS protocol implementation for constrained devices. We chose DTLS as the main protocol under investigation in this chapter to either confirm or refute the previously observed, often prohibitive memory requirements for a second end-to-end IP security protocol adaptation for the IoT. Importantly, the results of this analysis confirm our previous observations and identify public-key cryptography as the main driver for the high memory requirements.

As a second contribution, we present the handshake delegation architecture that we specifically designed for constrained devices with insufficient memory resources for a public-key-enabled protocol implementation. The key idea behind this architecture is to separate the initial connection establishment from the subsequent protection of application data and to delegate the initial connection establishment handshake to an off-path, trusted delegation server. The unconstrained nature of the delegation server then affords the use of public-key cryptography for peer authentication and key agreement purposes when performing the connection establishment on behalf of a constrained device. Moreover, by subsequently handing over the established connection context to the constrained device, this device no longer needs to implement expensive public-key cryptography for the connection establishment and can leverage efficient symmetric-key cryptography for the protection of application data.

As a third and final contribution, we describe how our architecture also provides an authorization framework for constrained node networks. This framework builds on the central role of the delegation server during the initial connection establishment, which allows to exercise control over new connections for local constrained devices.

The evaluation results show that, compared to a public-key-based DTLS protocol implementation, the handshake delegation architecture reduces the memory overhead by 64 %, computations by 97 %, and transmissions by 68 %. As we show that the core concepts of this architecture also apply to HIP DEX and Minimal IKEv2, similar results can likewise be achieved in the context of these protocols. Overall, handshake delegation, therefore, provides a comprehensive, yet compact solution for authentication, authorization, and secure data transmission in the IP-based IoT.

The remainder of this chapter is structured as follows. In Section 5.1, we present the results of our detailed memory analysis for DTLS. Section 5.2 then introduces the design of the handshake delegation architecture as a lightweight key provisioning mechanism for memory-constrained devices. Here, we also describe how handshake delegation affords to authorize new connections. Moreover, we show how the core ideas of our presented architecture similarly apply to HIP DEX and Minimal IKEv2. Next, we discuss the security considerations of the introduced architecture in Section 5.3 and present the evaluation results in Section 5.4. Finally, Section 5.5 discusses related work and Section 5.6 concludes this chapter with a summary. We note that the contents of this chapter is based on our published work in [HRW12, HZS$^+$13, HSR$^+$14] and our on-going standardization efforts in [HGS13].

5.1 Memory Analysis for DTLS

We now quantify and discuss the memory requirements of a DTLS protocol implementation for constrained devices. We thereby focus on inter-domain communication scenarios as exemplified by the interconnection of constrained devices with Internet-based services. Such communication scenarios typically involve the use of public-key cryptography for peer authentication and key agreement purposes during the connection establishment handshake. This is primarily for the following three reasons. First, public-key cryptography provides high scalability as already a single public/private key-pair suffices to authenticate an end-point to multiple communication partners. Second, the public key of a public/private key-pair does *not* denote secret information and, thus, can be transferred via untrusted network infrastructure, e.g., the Internet, without the need for cryptographic protection. Third, the public key can also be augmented with additional information such as a domain name via *certificates* in order to bind this key to a specific end-point or network domain [CSF$^+$08]. Such a binding then even allows *previously unassociated* communication end-points to authenticate each other and to establish secret keying material as long as these end-points trust a common third party, i.e., a Certificate Authority (CA).

For our memory analysis, we estimated the memory requirements of a DTLS protocol implementation with support for the certificate-based handshake via the *msp430-size* and *msp430-objdump* tools[1]. In this, we used the open source tinyDTLS implementation for constrained devices [tinydtls] as a basis. We refer to Section 5.4 for detailed

---

[1] We note that these tools are part of the GCC toolchain for the MSP430 MCU [mspgcc].

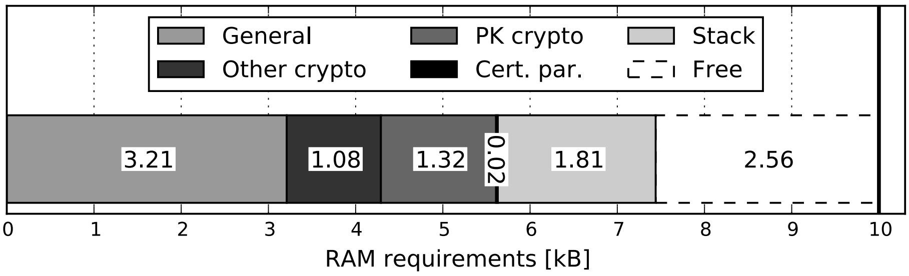


Figure 5.1 RAM requirements of a protocol implementation with support for the certificate-based DTLS handshake. Constrained devices with similar memory limitations as the TelosB platform (i.e., 10 kB of RAM as indicated by the bold vertical line) only have 2.56 kB of RAM available for the underlying operating system and the actual application logic.

information about our evaluation methodology as well as our evaluation setup and now continue with the discussion of the analysis results. To structure this discussion, we differentiate between (i) the RAM requirements, (ii) the ROM requirements, and (iii) considerations regarding the combined memory impact on a constrained device.

5.1.1 RAM Requirements

As illustrated in Figure 5.1, our memory analysis of the certificate-based DTLS protocol implementation reveals RAM requirements of about 7.44 kB (i.e., the sum of all depicted overheads). These RAM requirements consist of statically allocated RAM resources of about 5.63 kB and a maximum stack size of about 1.81 kB. To put these numbers into perspective, the TelosB platform, e.g., is equipped with a total of 10 kB of RAM. This only leaves about 2.56 kB of RAM for the underlying operating system and the actual application logic. Contiki OS, as one possible operating system for constrained devices, however, already requires about 4.02 kB of RAM for a simple hello-world example application[2] and the operating system itself. Similar to the HIP DEX implementation in Chapter 4, the certificate-based DTLS protocol implementation, therefore, also incurs prohibitively high RAM requirements for constrained devices with similar memory limitations as the TelosB platform.

When further investigating the root cause of these high RAM requirements, we observed that the certificate parsing functionality and public-key cryptography cause about 24 % (i.e., 1.34 kB) of the overall static RAM requirements. As depicted in Figure 5.1, about 1.32 kB of these RAM requirements are directly attributable to the *relic toolkit* [relic] as the underlying public-key library of the certificate-based DTLS protocol implementation. Importantly, the choice of a different cryptographic library would not have changed this observation significantly as similar RAM requirements also were reported for other open-source cryptographic libraries [LN08, SAKR12].

In addition, we found that the RAM requirements caused by certificates and public-key cryptography are not limited to the above cryptographic implementation components. Instead, they also influence the RAM requirements of non-cryptographic components such as the maintained message buffers and the persistent protocol state. These non-cryptographic implementation components require a total of about

---

[2] This observation refers to the hello-world example that is included in Contiki OS version 2.7.

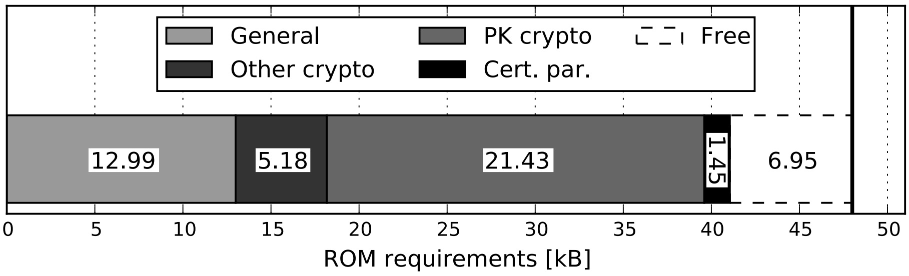


Figure 5.2 ROM requirements of a DTLS protocol implementation with support for the certificate-based handshake. Only about 6.95 kB of ROM remain for the underlying operating system and the actual application logic in case of a constrained device with similar memory limitations as the TelosB platform (i.e., 48 kB of ROM as indicated by the bold vertical line).

3.21 kB of static RAM in case of the certificate-based DTLS handshake (see “General” in Figure 5.1). Regarding the message buffers, we observe that the per-flight retransmission semantics of the DTLS protocol require a message-*sending* communication end-point to buffer sufficient information for the recreation of an *entire* message flight. Likewise, the handshake verification hash[3] in the Finished message requires a message-*receiving* communication end-point to buffer *all* out-of-order messages of a message flight to reconstruct the original sending order for an accurate computation of the hash digest. The long message flights of the certificate-based DTLS handshake (e.g., see flights 4 and 5 in Figure 5.15), however, call for capacious message buffers. Similarly, the protocol state of the certificate-based DTLS implementation involves at least 0.40 kB of RAM for the identifying certificate(s) and the private key of a constrained device. Based on these observations, we conclude that public-key cryptography and certificate-related functionality are prone to cause high RAM requirements concerning the *overall* protocol design.

5.1.2 ROM Requirements

Similar to our observations regarding the RAM requirements, our memory analysis of the certificate-based DTLS protocol implementation also reveals a substantial implementation size. Specifically, we found that the overall ROM requirements amount to as much as 41.05 kB (see all overheads in Figure 5.2 combined). Again referring to the TelosB platform as an example for a memory-constrained device, we note that this platform is equipped with 48 kB of ROM. Hence, for constrained devices with similar memory limitations as this platform, about 6.95 kB of ROM remain for the underlying operating system and the application logic. Contiki OS and the previously considered example application, however, already require about 16.83 kB of ROM. The certificate-based DTLS implementation, thus, is prone to also incur excessive ROM requirements in the context of memory-constrained devices.

Our detailed ROM analysis of the certificate-based protocol implementation shows that over 50 % of the above memory requirements (i.e., 21.43 kB) directly stem from the *relic toolkit*. Notably, this high ROM overhead of the underlying cryptographic library can be reduced by disabling the performance optimization techniques that

---

[3] We refer to Section 3.3.3 for further information about the handshake verification hash.

are typically employed to speed up the computation time of the performed public-key operations (see Section 4.1.7.1). The multiple precision integer and prime field operations as well as the public-key primitives themselves then demand about 10 kB of ROM [LN08] in the context of the certificate-based handshake. This, however, still leaves little memory space for the implementation of the actual application logic.

Moreover, even when disregarding the above overheads of public-key cryptography, our memory analysis of the certificate-based DTLS protocol implementation reveals significant ROM requirements. More precisely, already the base functionality of the certificate-based DTLS handshake, i.e., excluding all public-key primitives, causes ROM overheads of about 18.17 kB. As illustrated in Figure 5.2, these overheads include about 5.18 kB for the implementation of other cryptographic primitives, i.e., the AES-CCM mode of operation and the SHA-256 hash function. Furthermore, this base functionality also includes the extensive message processing routines that are required to handle the numerous handshake messages and the corresponding state machine of the certificate-based DTLS handshake. As a result, these processing routines denote a major contributor of the “General” ROM overhead in Figure 5.2.

Importantly, additional certificate-related functionality is likely to increase these handshake-related implementation overheads even further. Specifically, certificate verification requires loose but global *time synchronization* to validate the expiration time of the exchanged certificates and *certificate status verification* to ensure that these certificates neither have been revoked when the handshake takes place. In contrast, we note that the exclusion of public-key cryptography and certificate-related functionality in the DTLS protocol implementation would allow to significantly reduce the identified ROM overheads, i.e., as similarly observed in our RAM analysis.

5.1.3 Remarks About the Combined RAM and ROM Impact

In the previous sections, we specifically focused on the *individual* RAM and ROM requirements that are associated with the certificate-based DTLS handshake. The *accumulated* memory requirements, however, often also play an important role for constrained devices. All purely 16-bit-based MSP430 MCUs, e.g., can only address a maximum of 64 kB of RAM and ROM combined. This is because the underlying von-Neumann architecture maps program instructions and data into a *single* address space [SGG08]. The inherent memory bounds of these MCUs then only leave about 15.51 kB of RAM *and* ROM for the operating system and the application logic when considering the above memory requirements of the certificate-based DTLS implementation. Consequently, even when fully utilizing the addressing capabilities of this hardware architecture, there still is not enough memory space for Contiki OS and the previously considered example application due to a combined memory overhead of about 20.85 kB. Thus, we conclude that, similar to the HIP DEX implementation with overall memory requirements of about 66.28 kB (see Section 4.2.3.3), the certificate-based DTLS implementation often is infeasible for memory-constrained devices that, e.g., employ a 16-bit-based von-Neumann hardware architecture.

5.1.4 The Need for Lightweight Key Provisioning

We observe that the support for a purely symmetric-key-based handshake [ET05] in the DTLS protocol design already provides for a memory-efficient alternative to

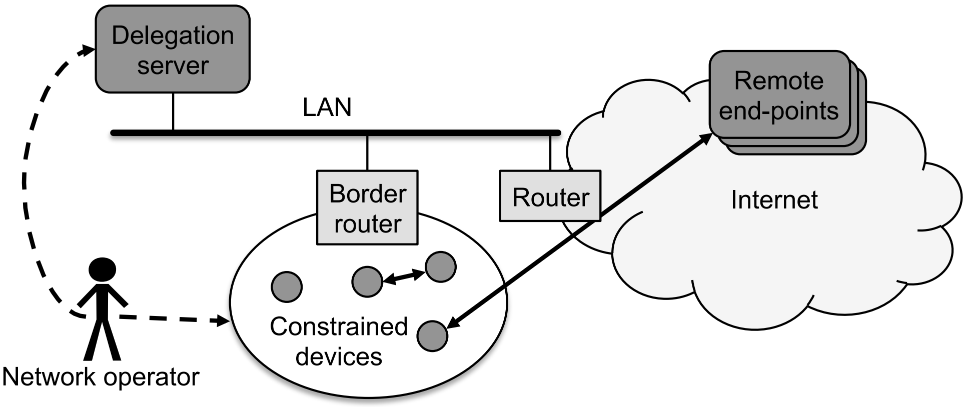


Figure 5.3 The handshake delegation architecture consists of the following entities (marked in dark gray): a network operator, a delegation server, constrained devices, and remote communication end-points. Continuous arrows highlight the different connections that are supported by the presented architecture. Dashed arrows indicate responsibilities of the network operator.

public-key cryptography during the DTLS handshake. More precisely, by replacing the public-key-based peer authentication and key agreement operations with symmetric-key-based alternatives, these operations involve a near zero memory overhead. This is because the employed cryptographic primitives then are also used for other tasks in the protocol design, e.g., the derivation of the session keys and the protection of application data. The symmetric-key-based DTLS handshake, however, requires end-point-specific cryptographic information to be *pre-shared* and readily available at both communication end-points. In other words, it requires a key provisioning mechanism that securely deploys secret keying material at both handshake peers *before* these peers can establish a secure connection via the DTLS protocol.

5.2 The Handshake Delegation Architecture

To solve the above dilemma of the symmetric-key-based handshake, we now present the design of our handshake delegation architecture as a *lightweight key provisioning mechanism* for inter-domain communication scenarios. Moreover, we show how the presented architecture also provides an *authorization framework* for inter- and intra-domain communication scenarios. As a general overview, we now first introduce the involved entities and our main assumptions. We then provide a detailed description of the key provisioning and authorization functionalities of the devised architecture.

5.2.1 Entities and Assumptions

As illustrated in Figure 5.3, our handshake delegation architecture consists of the following entities: (i) a network operator, (ii) a delegation server, (iii) constrained devices, and (iv) remote communication end-points. Regarding the *network operator*, we assume that this logical entity represents the owner of constrained devices and a delegation server in a private network scenario or (a group of) professional network administrators in an enterprise setting. In either case, we assume that the network operator is responsible for its constrained devices and for the decision, which other communication end-points these devices should be allowed to interact with.

The *delegation server* is an unconstrained, trusted network entity in our architecture that is located outside a constrained node network (see Figure 5.3). The main task of this server is to establish new connections on behalf of the network operator's constrained devices. *Constrained devices*, in turn, exhibit similar memory limitations as discussed in Section 5.1. These devices, however, should still be able to interact securely with other communication end-points. Besides the secure transmission of application data, this includes that constrained devices should be able to validate if the network operator authorized[4] the communication with a specific end-point. Such validation then, e.g., allows to prevent unauthorized, malicious end-points from accessing restricted information at a constrained device (see Section 3.1.4).

Concerning the *remote communication end-points*, we assume that these network entities have sufficient resources for a comprehensive, public-key-enabled protocol implementation as well as certificate validation functionality. Hence, the key provisioning mechanism facilitated by our handshake delegation architecture primarily focuses on less or unconstrained *remote* end-points such as conventional Internet-based services. With the authorization framework provided by our architecture, we, however, also target communication scenarios that only involve constrained devices.

5.2.2 Lightweight Key Provisioning

We now present how the handshake delegation architecture offers a lightweight key provisioning mechanism for memory-constrained devices. Our main goal thereby is to address the following two opposing objectives. On the one hand, we intend to facilitate current best practices for peer authentication and key agreement in *inter-domain* communication scenarios. We, thus, aim at re-using existing security protocols and public-key infrastructures as, e.g., employed for conventional Internet-based communication. On the other hand, we strive to enable constrained devices to securely interact with communication end-points from remote network domains.

The key idea behind the handshake delegation architecture is to *delegate* the initial connection establishment handshake, which would otherwise be performed between a constrained device and a remote communication end-point, to the delegation server. This allows to employ certificates and public-key cryptography when performing the peer authentication and key agreement operations at the delegation server *on behalf of* a constrained device. Moreover, by subsequently *handing over* the established connection context to the constrained device, the handshake delegation procedure effectively provides this device with the necessary keying material to communicate securely with the remote end-point. As a result, the constrained device then only needs to implement a partial protocol specification and efficient symmetric-key cryptography for the protection of application data. We now continue with a detailed description of the handshake delegation procedure, which is illustrated in Figure 5.4.

---

[4] We refer to Section 3.3.1 for a brief discussion of the main differences between authentication and authorization.

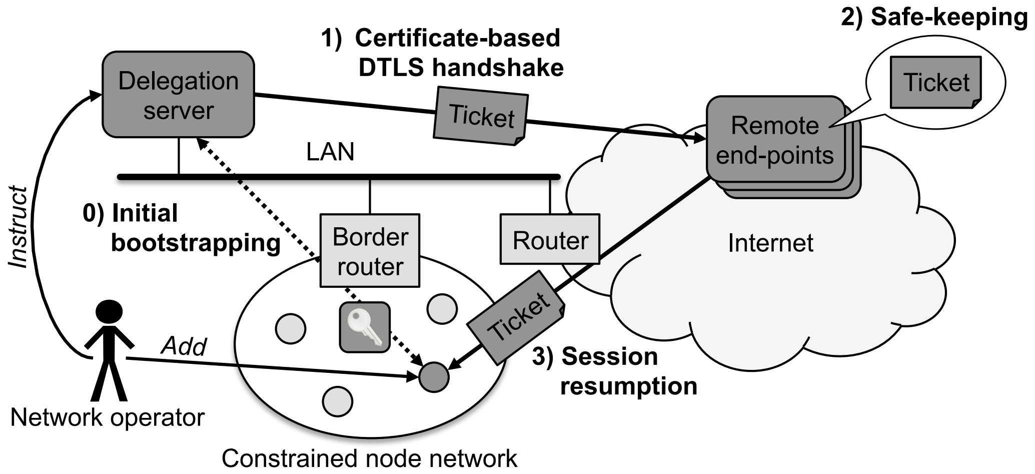


Figure 5.4 For the handshake delegation procedure, a newly added constrained device first has to perform an initial bootstrapping operation with the delegation server (Step 0). Instructed by the network operator, the delegation server then is able to perform a certificate-based DTLS handshake on behalf of the constrained device (Step 1). By encrypting the established connection context for the constrained device and by offloading it to the remote end-point in a session ticket via our previously introduced session resumption extension for DTLS (Steps 1 and 2), the delegation server can securely hand over the established connection context to the constrained device during a subsequent session resumption handshake (Step 3).

5.2.2.1 Initial Bootstrapping Phase

To establish new connections on behalf of a constrained device, the delegation server has to know the specific protocol parameters, e.g., the cipher suites, that it should negotiate for the constrained device during the connection establishment handshake. Moreover, the constrained device and the delegation server must share a common secret, i.e., the *device-specific delegation key*, to afford a secure handover of the established connection context. As shown in Step 0 of Figure 5.4, a constrained device and the delegation server, therefore, need to perform an initial bootstrapping operation prior to the actual handshake delegation procedure. Such bootstrapping can, e.g., be realized via physical contact [SA00] or physically secured wireless communication [KLNP07]. As part of this bootstrapping operation, the delegation server imprints the device-specific delegation key on the constrained device. Furthermore, the constrained device notifies the delegation server about its supported protocol parameters. The constrained device and the delegation server then are in the possession of all relevant information to perform the handshake delegation procedure.

5.2.2.2 Establishing a New Connection on Behalf of a Constrained Device

After the above bootstrapping phase, the network operator can trigger the establishment of a new connection between the constrained device and a remote communication end-point at the delegation server. To this end, the network operator provides the delegation server with the necessary addressing information of the intended communication partners. This addressing information can, e.g., be the IP address of an end-point or an identifier that resolves to the end-point's IP address. The delegation server then performs a certificate-based DTLS handshake with the requested remote end-point on behalf of the constrained device (see Step 1 in Figure 5.4).

During this handshake, the delegation server and the remote end-point most notably authenticate each other. Specifically, the delegation server authenticates the remote end-point via standard public-key certificates. The delegation server, therefore, maintains a pool of trusted root certificates similar to today's web browsers and operating systems. The remote end-point, in turn, either similarly authenticates the delegation server based on public-key certificates during the DTLS handshake or via an application-level password after the DTLS handshake has concluded. Importantly, while not allowing to explicitly authenticate the constrained device, this authentication operation still enables the remote end-point to verify the identity of the delegation server or of the network operator as a representative of this device.

For delegation purposes, the delegation server and the remote end-point additionally employ our previously introduced session resumption extension (see Section 4.1.2) during the certificate-based DTLS handshake. As a brief summary of this extension, we note that the basic idea behind session resumption is for the communication end-points to only perform a complete protocol handshake once during an initial connection establishment phase. The end-points then keep the established session state even after the connection is torn down. This enables the end-points to efficiently re-establish the previous connection based on the stored session state. Moreover, session resumption also allows one end-point to offload its session state to the other end-point in a protected session ticket for safe-keeping purposes. As a result, the offloading end-point then can remain stateless while the connection is inactive and can re-claim its offloaded session state in a subsequent session resumption handshake.

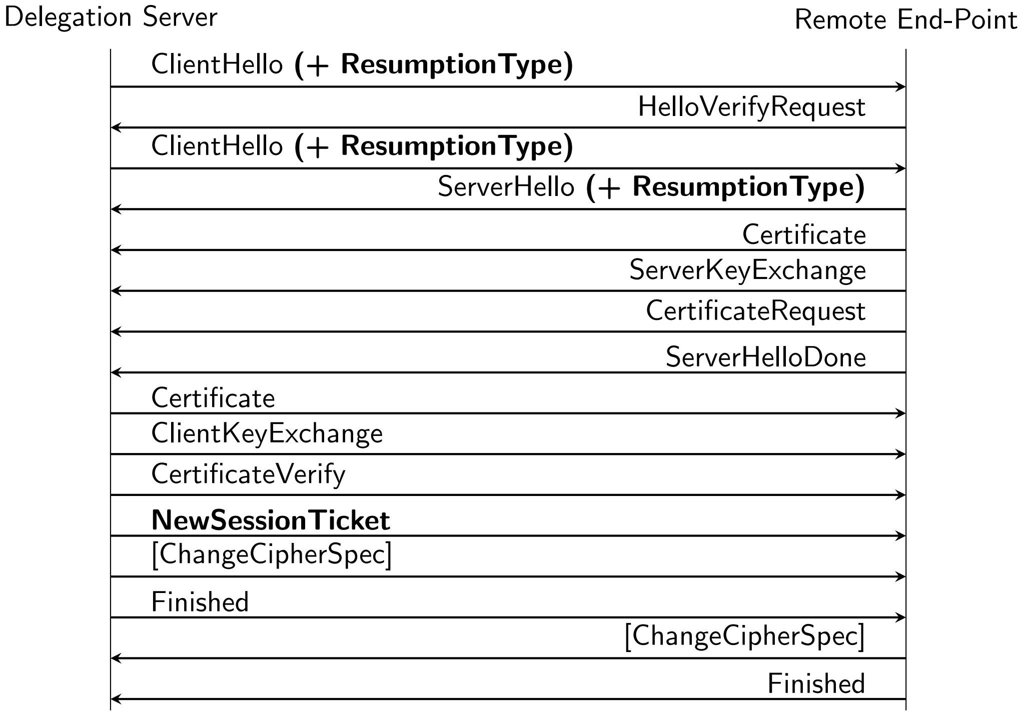


Figure 5.5 The delegation server and the remote communication end-point agree on session resumption with client-side state-offloading in the Hello messages of the certificate-based DTLS handshake. The delegation server then offloads its encrypted and integrity-protected session context to the remote end-point in the NewSessionTicket message. We note that session resumption-related information is marked bold. Brackets denote DTLS protocol extensions.

In the context of the handshake delegation procedure, the delegation server uses the latter session resumption type to transfer its established session context to the constrained device via the remote end-point. To this end, the delegation server indicates *session resumption with client-side state-offloading* as the only session resumption type in the ResumptionType parameter of the ClientHello messages. As depicted Figures 5.4 and 5.5, the corresponding handshake then allows the delegation server to push its session context to the remote end-point in a session ticket. The delegation server thereby encrypts and protects the integrity of the contained session context with the device-specific delegation key of the constrained device. This enables the constrained device to later decrypt the session context and to re-establish the connection with the remote end-point based on the session ticket from the delegation server. Moreover, the session ticket also constitutes an *implicit*, cryptographically verifiable notification for the constrained device stating that the remote end-point was correctly authenticated during the certificate-based DTLS handshake.

In addition, the delegation server also attaches the addressing information of the constrained device to the session ticket in an unencrypted form. The ticket then contains all information that is required for the remaining steps of the handshake delegation procedure. Finally, the delegation server closes the just established DTLS connection with the remote communication end-point and purges all associated connection state in order to limit the potential impact of a system compromise.

5.2.2.3 Leveraging the Session Context for the Protection of Application Data

At this point of the handshake delegation procedure, the remote communication end-point can establish the new connection with the constrained device as illustrated in Step 3 of Figure 5.4. To this end, the remote end-point initiates a session resumption handshake and indicates *session resumption with server-side state-offloading* in the ResumptionType parameter of the ClientHello message. Moreover, the remote end-point transfers the delegation server's session ticket to the constrained device (see ClientHello message in Figure 5.6). Upon reception of this ticket, the constrained device first decrypts the included session context and verifies its integrity via the device-specific delegation key. The handshake peers then derive new session keys

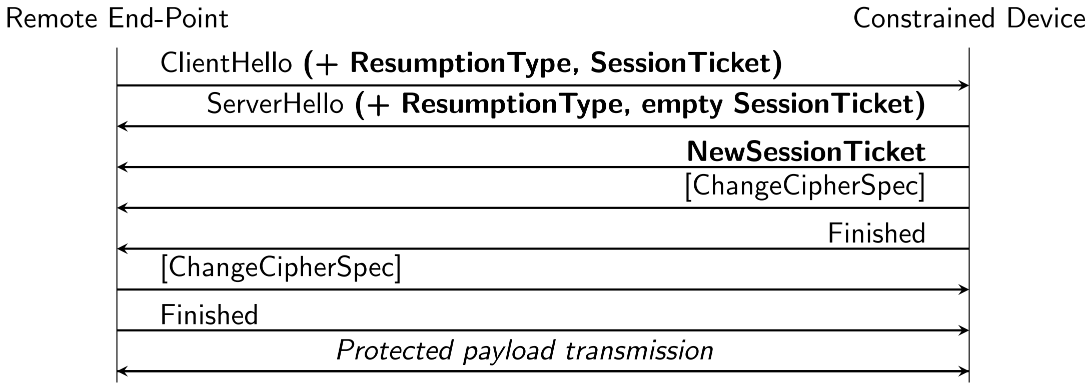

Remote End-Point
Constrained Device
ClientHello **(+ ResumptionType, SessionTicket)**
ServerHello **(+ ResumptionType, empty SessionTicket)**
NewSessionTicket
[ChangeCipherSpec]
Finished
[ChangeCipherSpec]
Finished
Protected payload transmission

Figure 5.6 Protocol handshake between the constrained device and the remote end-point in case the remote end-points triggers the session resumption handshake during our delegation procedure. Session resumption-related information is marked bold. Brackets denote protocol extensions. The NewSessionTicket message allows for repeat session resumption handshakes.

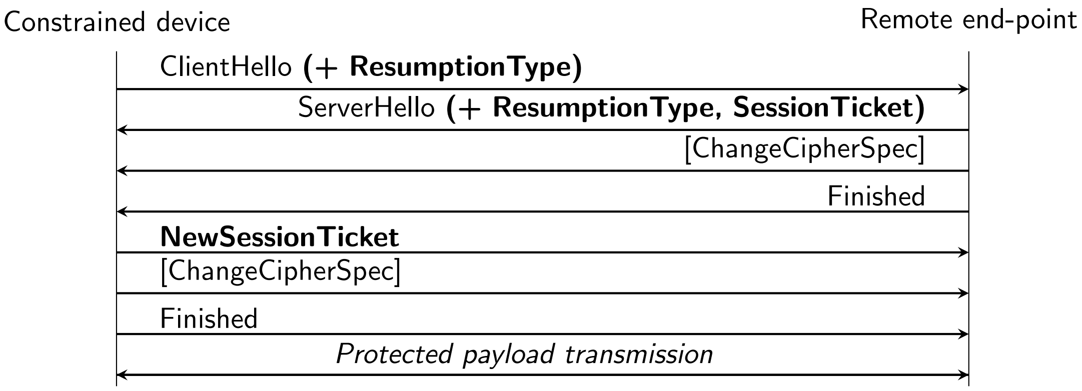

Constrained device
Remote end-point
ClientHello (+ ResumptionType)
ServerHello (+ ResumptionType, SessionTicket)
[ChangeCipherSpec]
Finished
NewSessionTicket
[ChangeCipherSpec]
Finished
Protected payload transmission

Figure 5.7 Protocol handshake between the constrained device and the remote end-point in case the constrained device triggers the session resumption handshake during our delegation procedure. Session resumption-related information is marked bold. Brackets denote protocol extensions. The NewSessionTicket message allows for repeat session resumption handshakes.

based on the previously established keying material in their session contexts [HGS13] and use these newly generated keys to authenticate each other in the Finished messages of the session resumption handshake [DR08]. Moreover, the peers employ these new session keys to protect their end-to-end transmission of application data.

The handshake delegation procedure additionally affords the constrained device to initiate the session resumption handshake. This is especially useful if, e.g., an on-path firewall blocks in-bound connection establishments from remote communication end-points. The device-initiated session resumption handshake, however, requires additional communication with the delegation server. Specifically, the delegation server has to send a connection initiation command with the addressing information of the remote end-point to the constrained device. As a result, the constrained device initiates the session resumption handshake with the indicated end-point and signals *session resumption with client-side state-offloading* in the ResumptionType parameter of the ClientHello message. As illustrated in Figure 5.7, the remote end-point then transfers the session ticket from the delegation server to the constrained device in the ServerHello message of the corresponding handshake. Upon reception of this session ticket, the constrained device decrypts the contained session context and uses this context to efficiently re-establish the previous connection with the remote end-point as before. Similarly, the handshake peers also employ the newly generated session keys to protect their end-to-end transmission of application data.

5.2.2.4 Benefits of Using Session Resumption for Delegation Purposes

By leveraging the session resumption extension for delegation purposes, the handshake delegation procedure exhibits a number of highly desirable properties in the context of memory-constrained devices. First, session resumption neither requires public-key cryptography nor certificate-related functionality at a constrained device. Thus, constrained devices no longer need to implement the corresponding expensive protocol components. Second, the session resumption handshake takes an abbreviated form compared to a certificate-based handshake. This allows us to unburden constrained devices from all handshake complexities that result from long message flights. Third, the session resumption handshake also enables the constrained device to offload its own session context to the remote end-point for safe-keeping purposes while a connection is *inactive* (i.e., see the NewSessionTicket messages in Figures 5.6 and 5.7). As a result, the handshake delegation procedure only requires constrained devices to maintain a *single* device-specific delegation key as well as session state for *actively* used connections. This allows our handshake delegation architecture to provide similar scalability properties as a purely public-key-based approach.

5.2.3 Authorizing Inter- and Intra-Domain Communication

In the previous sections, we specifically focused our discussion on how the handshake delegation architecture facilitates a lightweight key provisioning mechanism for memory-constrained devices. The presented architecture, however, also provides an authorization framework for these devices via the handshake delegation procedure. More precisely, the network operator already *implicitly* informs its constrained devices about their *authorized* remote end-points by only instructing the delegation server to facilitate the connection establishment for a limited number of remote end-points pertaining to a constrained device. Consequently, all other remote end-points constitute unauthorized communication end-points for this devices. Hence, to prevent such unauthorized end-points from communicating with a constrained device in the first place, we introduce a single new policy for the handshake delegation procedure. Specifically, we require a constrained device to only accept new connections that involve its device-specific delegation key during the performed DTLS protocol handshake. The constrained devices then only interact with those remote end-points that the network operator previously authorized via the delegation server.

We note that the handshake delegation architecture is not limited to such *end-point-specific* authorizations. Instead, the use of session tickets for delegation purposes also afford the secure transfer of more fine-granular *application-level* authorization information. Moreover, by leveraging the device-specific delegation key during the initial connection establishment, the handshake delegation procedure additionally enables the authorization of new connections in *intra-domain scenarios* that only involve constrained devices. Finally, a minor modification to the device-specific delegation key allows to provide an *efficient revocation procedure* for previously granted authorizations. In the following sections, we discuss these conceivable authorization-related extensions for the basic handshake delegation procedure in more detail.

5.2.3.1 Fine-Granular Application-Level Authorizations

The main purpose of application-level authorizations is to restrict access to sensitive information that is provided by a constrained device. This, e.g., includes access to configuration interfaces or to sources of private information. Exactly how an end-point accesses such information depends on the employed application layer protocol. For our discussion, we, therefore, focus on CoAP[5] as an application layer protocol that was specifically designed for IoT network scenarios. Still, we note that the following solution can similarly be integrated with other application layer protocols.

---

[5] We refer to Section 2.3.2 for more information about the CoAP protocol and its role in the adapted IP network stack for constrained devices.

```
{
  "URI":"path/to/restricted/resource",
  "REQ":["POST","PUT"]
}
```

Figure 5.8 Example application-level authorization. When the delegation server includes this authorization information in its session ticket, the remote end-point gains access to the indicated resource at the constrained device via the CoAP request methods POST and PUT.

Concerning the CoAP protocol, we recollect that its protocol design is based on a RESTful client/server architecture, similar to HTTP. As such, CoAP enables remote end-points to address specific functionality provided by a constrained device via a URI-based addressing scheme. Moreover, the CoAP protocol specification defines a number of request methods that allow the remote end-points to operate on these resources, i.e., GET, PUT, POST, and DELETE. The main goal of application-level authorizations in the context of CoAP, therefore, is to limit access to sensitive URIs and request methods to a select number of trusted communication end-points.

To facilitate such fine-granular application-level authorizations, we extend the session ticket with *explicit* authorization information as depicted in Figure 5.8. Regarding the URI, we note that this explicit authorization information may also contain wildcard characters to allow grouping of authorized URIs with equal request methods. Such *extended session tickets* then enable a constrained device to determine if a specific request method may be performed on a per-device *and* per-resource basis.

Fine-granular application-level authorizations, however, also exhibit an increased ticket size and, thus, an elevated transmission overhead during the session resumption handshake. Moreover, they cause increased run-time overheads at a constrained device due to the need to store and process the included fine-granular authorization information for incoming CoAP requests. We note that these overheads constitute trade-offs for the achieved gain in granularity regarding the authorization decision.

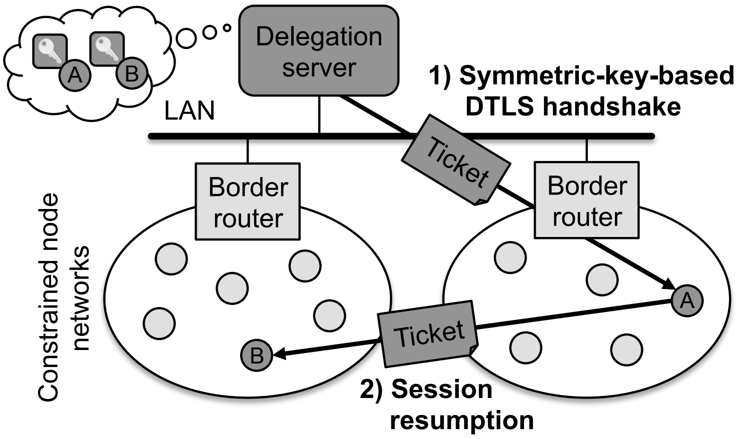


Figure 5.9 Leveraging our handshake delegation architecture for intra-domain authorizations. Instructed by the network operator, the delegation server performs a symmetric-key-based connection establishment handshake with a constrained device (A) on behalf of another constrained device (B). A and B subsequently perform a session resumption handshake. Notably, A and B can verify that the newly established connection was indeed authorized by the network operator as the handshakes involve the device-specific delegation key of A and B, respectively.

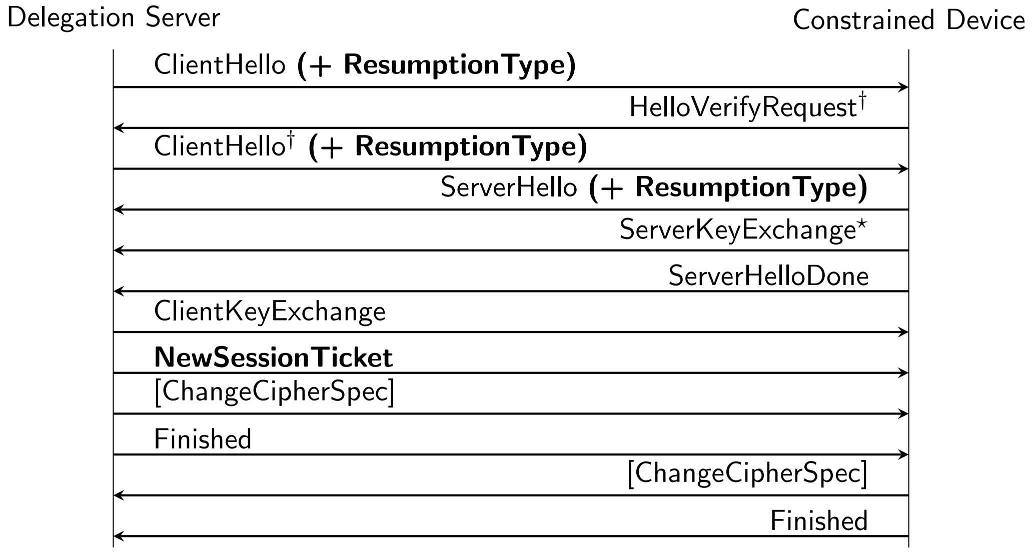


Figure 5.10 The delegation server and the constrained device agree on session resumption with client-side state-offloading in the Hello messages of the symmetric-based DTLS handshake. The delegation server then offloads its encrypted and integrity-protected session context to the constrained device in the NewSessionTicket message. Messages with a dagger (†) implement a return-routability test. Starred messages (*) are optional and situation-dependent. Session resumption-related information is marked in bold. Brackets denote DTLS protocol extensions.

5.2.3.2 Extension to Intra-Domain Authorizations

Until now, our discussion of the handshake delegation architecture – and, thus, of its authorization capabilities – focused on the interactions between constrained devices and remote communication end-points. As shown in Figure 5.9, the device-specific delegation keys, which the delegation server shares with its associated constrained devices, however, also allow to employ an adapted form of the handshake delegation procedure for intra-domain communication scenarios. Notably, this adapted handshake delegation procedure requires the involved devices to be associated with the same delegation server. For our following discussion, we additionally assume that these devices have already been bootstrapped as described in Section 5.2.2.1.

Similar to the original handshake delegation procedure, its adapted form is triggered by the network operator when authorizing a new connection between two communication end-points, denoted *Alice* and *Bob*, at the delegation server. With the adapted procedure, the delegation server then, however, performs a *symmetric-key-based* DTLS handshake with Alice on behalf of Bob (see Figure 5.10). During this handshake, Alice and the delegation server most notable employ Alice's device-specific delegation key to authenticate each other. The use of Alice's device-specific key assures her that the performed handshake involves the delegation server and, due to the trusted nature of the delegation server, that the new connection also was authorized by the network operator. As illustrated in Figure 5.10, Alice and the delegation server additionally employ our session resumption extension during the symmetric-key-based DTLS handshake to transfer the session ticket from the delegation server to Alice. The delegation server thereby protects its session context with Bob's device-specific delegation key. This allows Bob to later verify that the delegation server facilitated the connection establishment. Finally, Alice and Bob perform a session resumption handshake as discussed in Section 5.2.2.3 and employ the newly generated session keys for their secure transmission of application data.

Reducing Alice's Memory Burden

We note that the above procedure is prone to require a notable amount of Alice's limited RAM resources when authorizing her to communicate with several other constrained devices. This is especially true when employing extended session tickets for fine-granular application-level authorizations as described in the previous section. To reduce Alice's memory burden, we require the delegation server to only offload a *reference* to the actual session ticket during the symmetric-key-based handshake. To this end, the delegation server creates a locally unique identifier for each authorized connection and transfers this identifier as the sole information in the session ticket during the symmetric-key-based DTLS handshake. In addition, the delegation server associates this connection identifier with the original session ticket[6] and stores this information in a local database. As a result, offloaded session tickets in intra-domain communication scenarios then have a reduced, fixed size. This allows to relieve Alice from the memory burden of storing a multitude of full-size session tickets.

Due to the above modification, Alice, however, only transfers a session ticket with referencing information to Bob during the subsequent session resumption handshake. Hence, when receiving the session ticket from Alice during the session resumption handshake, Bob leverages the included connection identifier to request the original session ticket from the delegation server, e.g., via CoAP. Bob then uses the included session context to conclude the session resumption handshake with Alice. Overall, such *ticket referencing* offers an appealing memory/transmission trade-off for Alice.

5.2.3.3 Revocation of Inter- and Intra-Domain Authorizations

As discussed above, the handshake delegation procedure enables the network operator to establish and authorize new connections for constrained devices. The network operator, however, may eventually decide to revoke once granted authorizations. This can, e.g., be the case when a remote service is no longer used or when an authorized communication end-point has been compromised. To handle such revocations in an efficient manner, we require the delegation server to store the previously discussed, locally unique connection identifier as a reference for *all* authorized connections. Moreover, the delegation server includes this identifier in all session tickets, i.e., also in the session tickets of the certificate-based DTLS handshake. This enables the constrained device and the delegation server to leverage the connection identifier for backlisting purposes. More precisely, the delegation server then only needs to instruct the constrained device that is affected by a revocation decision to add the corresponding connection identifier to its connection backlist. As a result, the constrained device closes the corresponding connection and no longer accepts new protocol handshakes that involve the blacklisted connection identifier.

The size of this connection blacklist, however, increases linearly with each additional revocation. Revocations, thus, may result in non-negligible RAM requirements at

---

[6] This ticket is still encrypted and integrity-protected with Bob's device-specific delegation key.

the constrained device if blacklisting information needs to be maintained for an extensive period of time. Hence, to mitigate an unbounded growth of the blacklist, we additionally enable the delegation server to instruct the constrained device to change its device-specific delegation key. This invalidates all previously issued session tickets for this specific device and enables the device to flush its connection blacklist.

With the current design of our handshake delegation architecture, changing the device-specific delegation key implies the need for a new bootstrapping operation between the delegation server and the affected constrained device. Hence, to enable a constrained device to flush its connection blacklist without the need for re-bootstrapping, we now adjust the derivation of the device-specific delegation key.

Specifically, the delegation server and the constrained no longer directly employ the exchanged secret of the initial bootstrapping operation as the device-specific delegation key during the handshake delegation procedure. Instead, they leverage this secret for authentication and key agreement purposes during a symmetric-key-based DTLS handshake that is performed subsequent to the initial bootstrapping operation from Section 5.2.2.1. Both entities then use the derived *session key* of this handshake, i.e., the symmetric key that would otherwise be used for the encryption of application data, as the new device-specific delegation key. Resetting the connection blacklist of a constrained device then means performing another symmetric-key-based DTLS handshake between the constrained device and the delegation server.

Finally, the delegation server concludes the change of the device-specific delegation key by re-establishing the session contexts with these end-points that the constrained device is still authorized to interact with. Importantly, the overhead of this operation is limited to the delegation server and the individual affected end-points as the first step of the handshake delegation procedure does not involve the constrained device.

5.2.4 Integration with HIP DEX and Minimal IKEv2

As discussed in Section 5.1.3, a HIP DEX protocol implementation is prone to exhibit similarly prohibitive RAM and ROM requirements for a wide range of memory-constrained devices as a certificate-based DTLS protocol implementation. We expect this observation to also hold for an implementation of the Minimal IKEv2 protocol. This is primarily for the following two reasons. First, the protocol design of Minimal IKEv2 *mandates* the use of public-key cryptography during the protocol handshake. As a result, a Minimal IKEv2 implementation inevitably involves similar memory requirements concerning the employed cryptographic primitives as DTLS and HIP DEX. Second, the overall protocol layout of Minimal IKEv2 closely resembles the structure of HIP DEX (see Section 3.3.6). As such, also the non-cryptographic implementation components, e.g., the message parsing routines and the retransmission mechanism, are likely to incur comparable memory requirements. To still afford a memory-efficient operation of HIP DEX and Minimal IKEv2 in the context of constrained devices, we now briefly discuss how the key concepts of the presented handshake delegation architecture can also be applied to these protocols.

We observe that the bootstrapping phase of the handshake delegation procedure is protocol independent (see Section 5.2.2.1). As such, the performed bootstrapping operation equally allows to establish the device-specific delegation key for DTLS, HIP DEX, and Minimal IKEv2. Moreover, this operation also enables a constrained device to notify the delegation server about its supported protocol parameters in case of HIP DEX or Minimal IKEv2. Finally, session resumption can be employed with DTLS, HIP DEX, or Minimal IKEv2 as previously shown in Section 4.1.2. Overall, this allows to leverage the key provisioning and the inter-domain authorization functionalities of the handshake delegation architecture (see Sections 5.2.2 and 5.2.3, respectively) in the context of HIP DEX or Minimal IKEv2 by merely exchanging the underlying security protocol. This observation likewise applies to the fine-granular application-level authorizations introduced in Section 5.2.3.1.

Intra-domain authorizations and the presented efficient revocation procedure, however, cannot directly be translated to HIP DEX or Minimal IKEv2. This is because these authorization-related extensions require a symmetric-key-based protocol handshake. The protocol design of HIP DEX and Minimal IKEv2, however, only provides for public-key-based handshakes. Hence, these capabilities of the handshake delegation architecture constitute DTLS-specific functionality. Still, it is important to note that the revocation of authorizations in case of HIP DEX or Minimal IKEv2 can still employ the blacklisting approach described in Section 5.2.3.3. To flush the connection blacklist, the affected constrained device then, however, needs to be re-bootstrapped via the initial bootstrapping operation discussed in Section 5.2.2.1.

5.3 Security Considerations

The network interactions in the handshake delegation architecture are based on standard DTLS, HIP DEX, and Minimal IKEv2 protocol functionality as well as on our previously introduced session resumption extension. As such, the security considerations in Section 4.1.5 also apply to our work in this chapter. In addition, we now briefly discuss the impact of a potential system compromise concerning the different entities in the handshake delegation architecture ordered by their severity.

5.3.1 Impact of a Compromised Remote End-Point

The handshake delegation architecture requires a remote communication end-point, e.g., an Internet-based service, to store an offloaded as well as its own session context for each authorized, but inactive connection with a constrained device. Consequently, an adversary could aim at compromising a remote end-point in order to gain access to a *large number* of stored session contexts. As a result, the adversary would then be able to impersonate the remote end-point towards the affected constrained devices and gain access to potentially sensitive information or configuration interfaces. We note that the impact of this attack is comparable to the compromise of the remote end-point's private key in case of public-key cryptography.

Importantly, offloaded session contexts always are encrypted and integrity-protected in a session ticket. Potentially retrieved session tickets, therefore, cannot be exploited by the adversary. Still, the adversary could also try to gain access to the stored session contexts of the compromised remote end-point. Following current best practices for the protection of private keys in case of public-key cryptography, we, thus, require remote end-points to encrypt their session contexts for inactive connections[7]. Such encryption then prevents the adversary from accessing the sensitive information in potentially retrieved session contexts from the remote communication end-point.

5.3.2 Impact of a Compromised Delegation Server

An adversary may also aim at compromising a delegation server and at retrieving the device-specific delegation keys. While the impact of this attack is limited to the constrained devices that are associated with the delegation server, this attack would enable the adversary to issue new or to revoke existing authorizations. Hence, we require a delegation server to be hardened against a system compromise via firewalls and access control security policies [selinux], i.e., similar to a Key Distribution Center (KDC) of centralized authentication protocols such as Kerberos [NYHR05].

Notably, if an adversary still manages to compromise the delegation server, the network operator has to revoke the certificate of this delegation server and issue a new one after the server has been sanitized or replaced. Moreover, all constrained devices that were associated with the compromised server must be re-bootstrapped and provided with a new device-specific delegation key to prevent the adversary from impersonating the delegation server towards these constrained devices. To return to the state prior to the system compromise, the delegation server then has to re-establish the authorized connections for all of its constrained devices.

5.3.3 Impact of a Compromised Constrained Device

Finally, an adversary could also aim at compromising a constrained device and try to gain access to the device-specific delegation key. The adversary then would be able to exploit her knowledge of this key in two ways. First, the adversary could impersonate the delegation server towards the constrained device in order to issue new or to revoke existing authorizations. Second, the adversary could impersonate the constrained device towards its authorized communication end-points.

After the compromise of the constrained device has been detected, the constrained device, thus, must be sanitized and re-bootstrapped in order to imprint a new device-specific delegation key on this device. Moreover, the delegation server must revoke all existing session tickets for the device by re-establishing its authorized connections. We note that comparable procedures would also be required in case of public-key cryptography to handle the compromise of the private key of a constrained device.

5.4 Evaluation

For the evaluation of the handshake delegation architecture, we analyzed the overheads of the presented lightweight key provisioning mechanism and the end-point-specific authorization functionality in the context of constrained devices. To this

---

[7] This protection incurs a computation overhead that resembles the overhead of encrypting and decrypting a session ticket. The evaluation results in Section 5.4.2.1 show that these operations cause moderate computation overheads on constrained devices and, thus, are also feasible when protecting the session context on a constrained device in case of intra-domain authorizations.

end, our evaluation focussed on the final step of the handshake delegation procedure, i.e., the session resumption handshake, as the main delegation phase involving a constrained device. The symmetric-key-based and the certificate-based DTLS handshakes served as comparison baselines. The former handshake type thereby relied on the open source tinyDTLS implementation [tinydtls], whereas the latter required us to extend this implementation with certificate-related protocol functionality. Moreover, we added support for the session resumption extension to this protocol implementation and provisioned the communication partner of a constrained device with a pre-computed session ticket from the delegation server for handshake delegation purposes. We then evaluated and compared the three different connection establishment types on constrained devices with Contiki OS [DGV04] in version 2.7.

As the underlying hardware platform, the constrained devices in our evaluation setup employed the WiSMote platform[8] [wismote] with a 16 MHz MSP430 MCU, 16 kB of RAM, 128 kB of ROM, and an IEEE 802.15.4-2006 radio interface. This platform provided the constrained devices with extended 20 bit addressing support. As a result, we were able to run and compare all three considered connection establishment types based on the *same* underlying hardware platform. A purely 16 bit-based hardware platform such as the TelosB would not have allowed us to do so as an insufficient amount of ROM prevented the execution of the certificate-based DTLS handshake on these devices. Still, it is important to note that, while our evaluation focuses on the WiSMote platform for comparison purposes, we also successfully verified that the TelosB platform supports the evaluated handshake delegation procedure.

Our evaluation setup followed current security recommendations for constrained devices[9]. Specifically, we used elliptic curve NIST P-256 for all public-key-related operations, AES-128 for symmetric-key-based operations, and SHA256 for hashing purposes. The public-key and hashing operations relied on the *relic toolkit* [relic]. To evaluate the certificate-based DTLS handshake, we created a self-signed certificate for our own root Certificate Authority (CA) and pre-provisioned this certificate on all constrained devices in our evaluation setup. We also issued per-device certificates that were signed by our root CA. With this setup, the certificate-based DTLS handshake required the transmission of one certificate per communication end-point and one signature verification for the validation of the corresponding certificate chain. Notably, the observed overheads of the certificate-based DTLS handshake constitute a lower bound as the per-device certificates in our evaluation setup had a small size (i.e., 0.37 kB) and did not involve intermediate certificates in the certificate chain.

Similar to the evaluation setups in Chapter 4, our evaluation did not specifically consider the computation overhead of link layer security, but took the transmission overhead that stems from the *maximum length* of the link layer security header into account. This header overhead amounts to 30 byte for IEEE 802.15.4-2006. Hence, we reduced the available 6LoWPAN payload size accordingly. Furthermore, the IPv6 and UDP headers were compressed at the 6LoWPAN layer to the extend discussed in Section 2.4.1 via the LOWPAN_IPHC and LOWPAN_NHC compression

---

[8] We refer to Section 2.2.1 for a brief overview and a comparison of the different experimentation platforms for constrained devices that we use throughout the course of this thesis.

[9] The CoAP protocol specification defines cipher suite TLS_PSK_WITH_AES_128_CCM_8 [MB12] as mandatory-to-implement in the context of the symmetric-key-based DTLS handshake and TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8 [MBCD14] with elliptic curve NIST P-256 as mandatory-to-implement for the certificate-based DTLS handshake [SHB14].

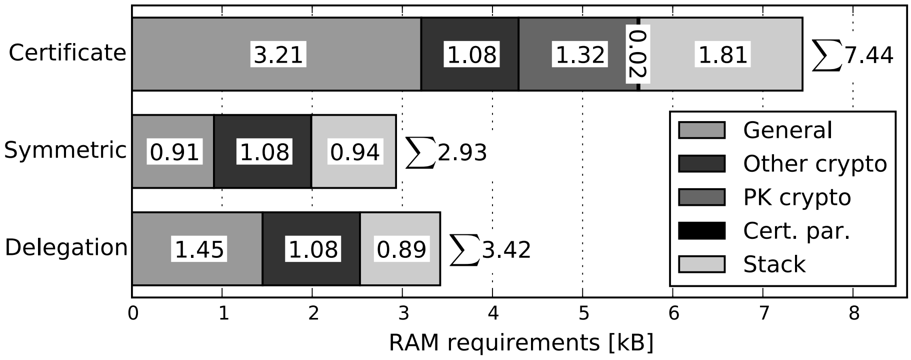


Figure 5.11 RAM requirements of the DTLS implementation variants supporting the certificate-based handshake, the symmetric-key-based handshake, and the handshake delegation procedure. Handshake delegation requires slightly more RAM resources than the symmetric-key-based handshake, but considerably less than the certificate-based handshake.

mechanisms. As a result, DTLS handshake messages with a size above 33 byte, i.e., excluding all lower layer headers, were fragmented by the 6LoWPAN layer prior to their transmission and reassembled at the fragment recipient (see Section 2.4.2 for more information about the 6LoWPAN fragmentation mechanism). Moreover, FRAG1s contained 24 byte of DTLS protocol information, whereas FRAGNs carried between 1 and 64 byte of DTLS message content. We note that, by accounting for the maximum size of the link layer security header and by only considering modest header compression ratios, the following evaluation results exhibit a worst-case message fragmentation with respect to the above evaluation setup.

5.4.1 Reduced RAM and ROM Requirements

To evaluate the memory overhead of the handshake delegation architecture in the context of constrained devices, we analyzed and compared the RAM and ROM requirements of Contiki OS binaries with support for (i) the symmetric-key-based handshake, (ii) the certificate-based handshake, and (iii) our handshake delegation procedure via the *msp430-size* and *msp430-objdump* tools. Moreover, we measured the maximum stack size of these DTLS implementation variants. To this end, the employed constrained device initialized its RAM resources with a well-defined pattern as the first task of its boot-up procedure and then executed a series of connection establishments, during which it acted as a server. After each connection establishment, we dumped the RAM content of this device in order to determine the memory region that still contained the well-defined memory pattern. The subtraction of the determined amount of unallocated RAM and the previously identified statically allocated RAM from the overall RAM resources of the WiSMote platform then yielded an estimation of the maximum stack size. The presented stack size results constitute the maximum over 10 measurement runs with a standard deviation of zero.

Based on this memory analysis, we found that the certificate-based DTLS protocol implementation exhibits an almost threefold RAM and ROM increase compared to the symmetric-key-based DTLS implementation (see overall overheads in Figures 5.11 and 5.12). The key differentiators are the underlying public-key library and the certificate parsing functionality with a combined overhead of 1.34 kB of

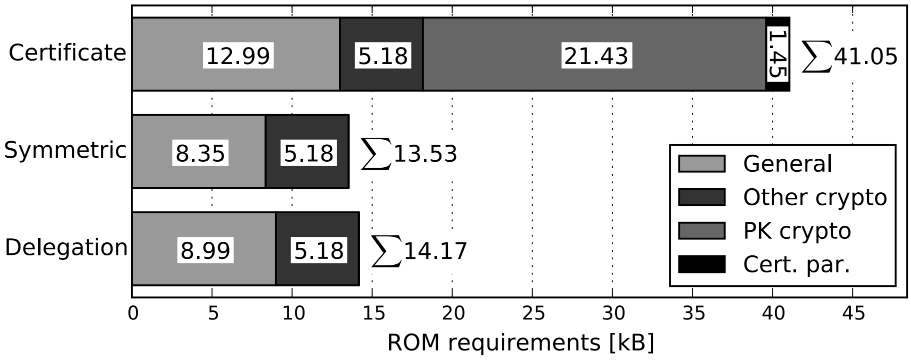


Figure 5.12 ROM requirements of the DTLS implementation variants supporting the certificate-based handshake, the symmetric-key-based handshake, and the handshake delegation architecture. The public-key-related protocol functionality constitutes the major contributor to the ROM requirements of the certificate-based DTLS handshake. Handshake delegation and the symmetric-key-based handshake exhibit similar ROM requirements.

statically allocated RAM and 22.88 kB of ROM. Still, also the DTLS base functionality exhibits a higher memory overhead in case of the certificate-based DTLS implementation (see "General" in Figures 5.11 and 5.12). This overhead increase predominantly stems from the additional and more complex DTLS message processing routines as well as from the larger message buffer for the longer message flights of the certificate-based DTLS handshake (compare the unstarred messages in Figure 3.6 with the complete DTLS protocol handshake). Notably, the message buffer-related RAM requirements of the certificate-based DTLS implementation are prone to be even higher in many real-world deployment scenarios. This is because larger certificates and longer certificate chains than the ones employed in our evaluation setup further increase the size of the exchanged Certificate messages and, thus, add to the overall length of message flights 4 and 5 of the certificate-based DTLS handshake.

In addition to these static memory requirements, we also observed that the maximum stack size of the certificate-based DTLS implementation variant is almost twice as high as the maximum stack size of the symmetric-key-based implementation variant (i.e., 1.81 kB and 0.94 kB, respectively). Also here, the most significant overhead driver is the public-key library that dynamically allocates RAM resources during its initialization procedure as well as during the performed cryptographic operations.

Concerning the *handshake delegation procedure*, we identified significantly reduced memory requirements compared to the certificate-based DTLS implementation (see Figures 5.11 and 5.12). Specifically, handshake delegation achieves an *overhead reduction* that amounts to 4.02 kB (i.e., 54.03 %) of RAM and 26.88 kB (i.e., 65.48 %) of ROM. Moreover, the handshake delegation procedure only shows a modest increase in memory requirements compared to the symmetric-key-based DTLS protocol implementation, i.e., 0.49 kB of RAM and 0.64 kB of ROM. This increase in memory overhead primarily originates from the additional protocol logic that is required for the session resumption extension and the storage capacity for one session ticket that the constrained devices in our evaluation setup were configured to store for intra-domain authorization purposes. We note that each additional session ticket that a constrained device can store further increases this RAM overhead by about 0.11 kB.

Based on the overall memory reductions of 30.90 kB (i.e., 63.72 %) compared to the certificate-based DTLS implementation, we conclude that our handshake del-

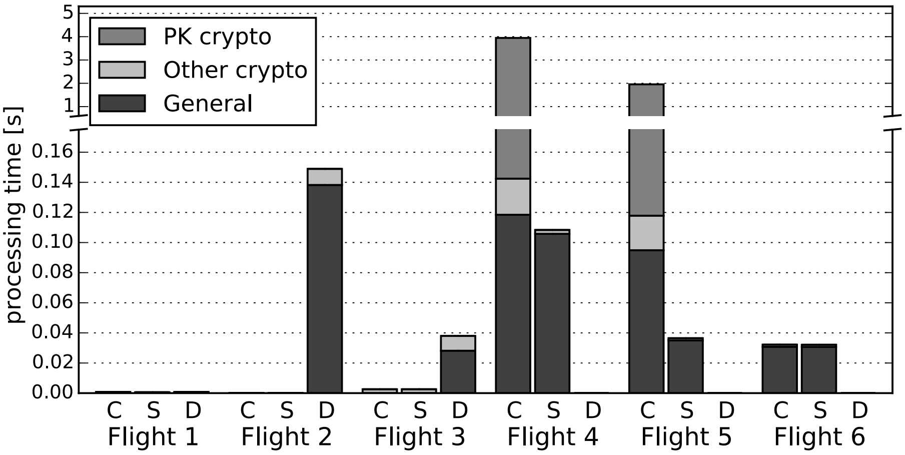


Figure 5.13 Client-side computation overhead of the certificate-based DTLS handshake (C), the symmetric-key-based DTLS handshake (S) and the presented handshake delegation procedure (D) aggregated on a per-flight basis. The handshake delegation procedure causes a low computation overhead that is comparable to a symmetric-key-based DTLS handshake.

egation procedure significantly improves the feasibility of the DTLS protocol for memory-constrained devices. Notably, these memory reductions also include the necessary protocol functionality for end-point-specific authorizations. Fine-granular application-level authorizations, however, would require additional memory resources as we did not consider the corresponding protocol functionality in the analyzed binaries. The increased memory requirements for such application-level authorizations then denote a trade-off for the optional gain in authorization granularity.

5.4.2 Additional Run-time Improvements

In addition to the above memory evaluation, we also analyzed the run-time performance of our handshake delegation procedure in the context of constrained devices based on the above Contiki OS binaries. We thereby first measured the computation and transmission overheads of the certificate-based and the symmetric-key-based DTLS handshake between two wirelessly connected constrained devices. We then compared these baselines against the evaluation results of the handshake delegation procedure. The discussed evaluation results represent the average over 100 measurement runs. The standard deviation was below 13.9 ms (i.e., 0.73 %) for the public-key-based operations and below 1.55 ms for all remaining protocol operations.

5.4.2.1 Computation Overhead

As shown in Figures 5.13 and 5.14, the certificate-based DTLS handshake causes an overall computation overhead of about 6 s *per communication end-point* independent from the role during the protocol handshake. This significant overhead primarily stems from the ECDSA-based peer authentication process with about 4.32 s and the ECDH-based key agreement procedure with about 1.32 s. To put these numbers into perspective, the above public-key operations combined cause about 95 % of the per-end-point computation overhead. Hence, the public-key operations of the certificate-based DTLS handshake dwarf the remaining computation overheads of 0.25 s for the general packet processing and of 0.05 s for the symmetric-key and hash

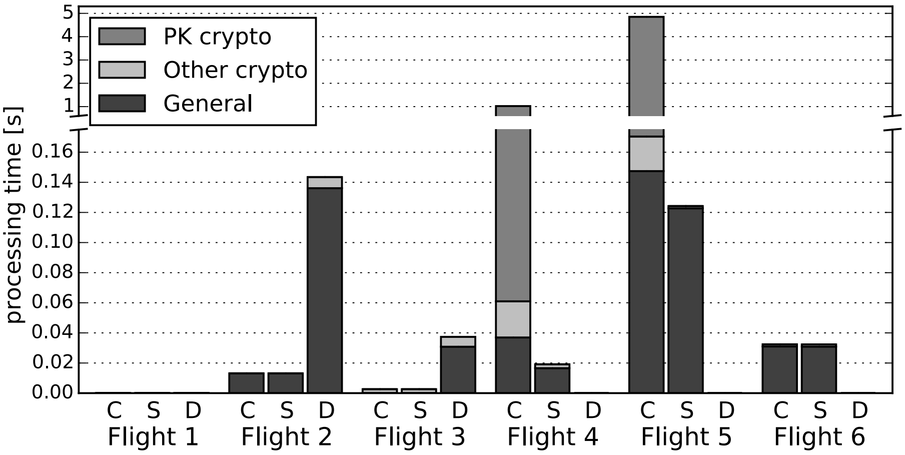


Figure 5.14 Server-side computation overhead of the certificate-based DTLS handshake (C), the symmetric-key-based DTLS handshake (S) and the handshake delegation procedure (D) aggregated on a per-flight basis. The overheads at the server are similar to those at the client.

operations (e.g., see flights 4 and 5 in Figure 5.13). Importantly, the validation of a certificate chain only requires a single ECDSA signature verification of about 1.90 s in our evaluation setup. We note that this overhead of the certificate-based DTLS handshake increases linearly with each additional certificate in the certificate chain.

Regarding the symmetric-key-based DTLS handshake, we observed considerably reduced computation overheads compared to the certificate-based handshake. Specifically, the symmetric-key-based DTLS handshake causes an overall computation overhead of about 0.18 s per communication end-point. This observation likewise holds in the context of the *handshake delegation procedure* with a similarly low computation overhead of about 0.19 s. The slightly increased overhead of the handshake delegation procedure thereby is primarily caused by the verification and the decryption of a session ticket with about 8.59 ms as well as by the generation of a new session ticket for subsequent handshakes with about 11.08 ms (e.g., see flight 2 in Figure 5.13). A part of these additional overheads, however, is compensated by the fewer DTLS messages of the session resumption handshake and, thus, the decreased packet processing costs compared to the symmetric-key-based DTLS handshake.

To summarize, our handshake delegation procedure reduces the overall computation overhead compared to a certificate-based DTLS handshake by 96.80 % and achieves a computation efficiency that resembles the standard symmetric-key-based handshake.

5.4.2.2 Transmission Overhead

As illustrated in Figure 5.15, our general observations regarding the computation overhead of the three different connection establishment types similarly apply to the measured transmission overheads. More precisely, we found that the certificate-based DTLS handshake causes an overall transmission overhead of 1609 bytes at the DTLS protocol layer for all handshake messages combined. In contrast, the symmetric-key-based DTLS handshake and the handshake delegation procedure only incur transmission overheads of 458 bytes and 515 bytes, respectively.

Concerning the certificate-based DTLS handshake, all 15 messages of the protocol handshake are required when performing the connection establishment (see all mes-

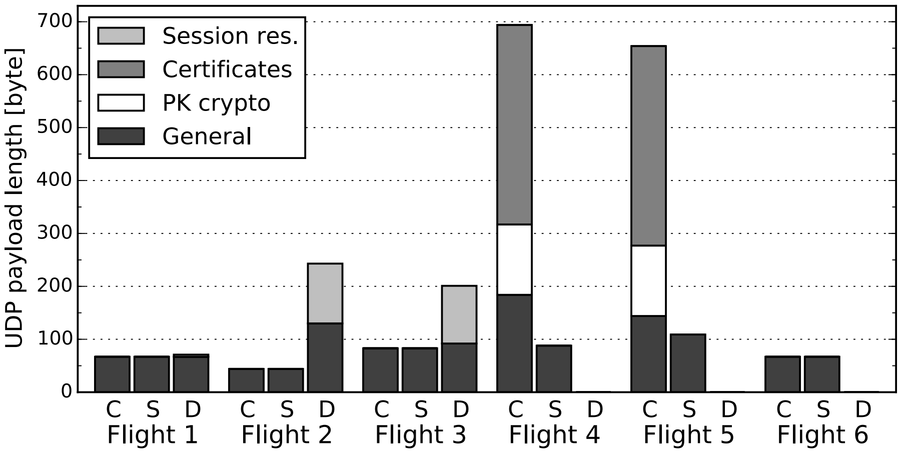


Figure 5.15 Transmission overhead of the certificate-based DTLS handshake (C), the symmetric-key-based DTLS handshake (S) and the presented handshake delegation procedure (D) aggregated on a per-flight basis. The handshake delegation procedure causes a reduced transmission overhead compared to the certificate-based DTLS handshake and a similar transmission overhead as the symmetric-key-based DTLS handshake.

sages in Figure 3.6). In addition, the extensive size of these messages causes the certificate-based handshake to be split into a total of 41 6LoWPAN fragments. This high level of fragmentation is especially critical when considering the flight-based retransmission semantics of the DTLS protocol that cause an entire message flight to be retransmitted in case a single 6LoWPAN fragment is lost. As each additional 6LoWPAN fragment per message flight further adds to the overall probability of fragment loss, DTLS message flights should preferably consist of only few 6LoWPAN fragments. The evaluation results, however, show that the longest message flights of the certificate-based DTLS handshake, i.e., flights 4 and 5, are split into a total of 16 6LoWPAN fragments each. As these message flights also contain certificate-related information, larger certificates or longer certificate chains than the ones employed in our evaluation setup add to the size of these message flights and, thus, even further increase the probability of retransmissions in case of lossy radio links.

In contrast, the symmetric-key-based DTLS handshake allows to reduce the number of handshake messages to a total of 10 (see unstarred messages in Figure 3.6). As the individual message sizes additionally are considerably smaller than those of the certificate-based DTLS handshake, only 17 6LoWPAN fragments need to be transmitted for the entire symmetric-key-based handshake. Interestingly, the *handshake delegation procedure* even further reduces the required message fragmentation to 15 6LoWPAN fragments. This is mainly because the employed session resumption handshake only consists of a total of 7 handshake messages. The largest message, i.e., the ServerHello conveying the session ticket of 109 byte, thereby consists of 4 6LoWPAN fragments. Hence, while exhibiting a slightly increased absolute transmission overhead, our handshake delegation procedure achieves a lower overall packet forwarding overhead than the symmetric-key-based DTLS handshake.

To conclude, the presented handshake delegation procedure significantly outperforms the certificate-based DTLS handshake with respect to the observed computation, memory, and transmission overheads. At the same time, handshake delegation offers a lightweight key provision mechanism for secure inter-domain communication scenarios and provides an authorization framework for constrained node networks.

While providing these additional capabilities, our handshake delegation architecture only incurs a modest memory overhead compared to a purely symmetric-key-based DTLS implementation and achieves similarly low computation and transmission overheads as the symmetric-key-based handshake. Hence, our presented architecture considerably improves the feasibility of the DTLS protocol in particular and of end-to-end IP security in general with respect to many memory-constrained devices.

5.5 Related Work

For our discussion of related work, we distinguish between the following two research directions: (i) delegation-based approaches for standard end-to-end IP security protocols and (ii) authorization frameworks for the IP-based IoT. Moreover, we refer to our discussion of related work in Section 4.1.7.1. Here, we additionally note that, while ECC hardware support allows to decrease the computation and memory requirements that are directly attributable to the corresponding public-key primitives, such hardware support does *not* reduce the additional memory overheads of the certificate-based DTLS handshake, e.g., the large message buffers and the functionality for global time synchronization as well as for certificate status verification. In contrast, by purely relying on symmetric-key-based primitives in the context of constrained devices, our handshake delegation procedure does not cause these additional overheads in the first place. Consequently, our architecture also caters to highly memory-constrained devices that are equipped with ECC hardware support.

5.5.1 Delegation-based Approaches for End-to-End IP Security

Several delegation-based approaches for standard end-to-end IP security have recently been proposed that aim at offloading expensive protocol operations from a constrained device to less constrained network entities (see Table 5.1 for a comparison overview). In [SO12a, SO12b, SO12c], Saied et al. present three delegation-based approaches to offload the public-key-based handshake operations of the HIP protocol from a constrained device to a group of less constrained *neighboring devices*, called proxies. Common to all three approaches is the idea to split the secret key of the constrained device into multiple blocks and to distribute the individual key shares to a number of proxies as part of the performed protocol handshake. The proxies then perform the peer authentication and the key agreement operations on behalf of the constrained device. For key agreement purposes, the proxies thereby employ the received key shares from the constrained device. In a last step, the constrained device securely reconstructs the end-to-end keying material based on the received handshake messages from the involved proxies. By employing *multiple* proxies in the HIP protocol handshake, the proposed approaches, however, add a considerable number of handshake messages to the connection establishment inside a constrained node network. In contrast, our handshake delegation architecture achieves a reduction of the computation, the memory, *and* the transmission overheads.

Similarly, Sahraoui et al. [SB14] recently proposed CD-HIP, an approach to delegate the public-key signature verification and the DH key agreement of the HIP protocol

Approaches	Security	Computation	Memory	Transmission	Trusted entity
Said et al.	$+$	$-$	$\approx$	$-$	Constr. proxies
CD-HIP	$-$	$-$	$-$	$\approx$	Constr. device
Tiny 3-TLS	$-$	$-/\approx$	$-/\approx$	$\approx/+$	On-path GW
Bonetto et al.	$-$	$+$	$\approx$	$+$	On-path GW
Granjal et al.	$-$	$\approx$	$\approx$	$\approx$	On-path GW
Park et al.	$-$	$+$	$+$	$-$	On-path server
SCVP	$\approx$	$-$	$-$	$-$	Off-path server
Kerberos	$\approx$	$\approx$	$-$	$\approx$	Off-path KDC

Table 5.1 High-level comparison of the discussed related delegation-based approaches. These approaches exhibit inferior ($-$), similar ($\approx$), or improved ($+$) properties when compared to our handshake delegation architecture. In case the authors did not quantify overheads, the presented comparison results denote estimates based on the provided architectural descriptions.

to a *single* less constrained neighboring device. In contrast to our work, this delegation approach, however, still requires constrained devices to implement public-key cryptography in order to sign their handshake messages. Moreover, the authors require a subset of the devices inside a constrained node network to be trusted.

With Tiny 3-TLS [FMMA06], Fouladgar et al. propose to delegate the public-key-based operations of the certificate-based TLS handshake to an *on-path gateway*. Tiny 3-TLS thereby distinguishes between two trust models, i.e., a partially and a fully trusted on-path gateway. In case of partial trust, Tiny 3-TLS offloads the certificate-based peer authentication from the constrained device to the on-path gateway. Subsequently, the *constrained device* and the remote end-point perform an ECDH key agreement in order to establish the end-to-end keying material. Thus, contrary to our handshake delegation procedure, Tiny 3-TLS still requires constrained devices to implement expensive public-key cryptography in case of a partially trusted gateway.

With a fully trusted gateway, Tiny 3-TLS delegates the *entire* handshake to the on-path gateway. The gateway then securely transmits the established end-to-end keying material to the constrained device. Similar to this second delegation variant, Bonetto et al. [BBL$^+$12] propose to offload the IKEv2 handshake to an on-path gateway. Moreover, Granjal et al. [GMSS13] propose to integrate the certificate-based DTLS handshake between a remote end-point and the gateway with a symmetric-key-based DTLS handshake between the gateway and a constrained device. Importantly, these approaches entail the exposure of the established end-to-end keying material to the on-path gateway. This enables the gateway to eavesdrop on protected end-to-end communication between the end-points. Our handshake delegation procedure, contrarily, offloads the initial connection establishment to a *dedicated, off-path network entity*, i.e., the delegation server. Hence, the delegation server would first need to subvert the routing topology of a given network in order to mount this type of attack. Moreover, in contrast to these approaches, our handshake delegation architecture does *not* require on-path gateways to perform (expensive) security-related tasks and, thus, supports the deployment of commodity IP-level gateways.

Subsequent to our work in this chapter, Park et al. [PK14] proposed to delegate the DTLS connection establishment handshake to a *dedicated network entity*, called the DTLS delegator. After the connection establishment handshake, the DTLS delegator securely transmits the established session context to the constrained device. In contrast to our handshake delegation procedure, this approach, however, incurs additional transmission overheads for the subsequent protection of application data. Moreover, the authors require the DTLS delegator to be located *on* the communication path between a constrained device and a remote communication end-point, i.e., even after the session context has been handed over successfully. As mentioned above, this potentially enables the DTLS delegator to eavesdrop on protected end-to-end communication between the legitimate communication end-points. Conversely, our delegation server is located *off-path*, thus hampering this type of attack.

Regarding standardized delegation-based approaches, the Server-based Certificate Validation Protocol (SCVP) [FHM$^+$07] enables a constrained device to delegate its certificate verification procedure to a trusted *validation server*. The constrained device then no longer needs to implement the time synchronization and certificate status verification functionalities that otherwise would be required for a certificate-enabled protocol implementation. SCVP, however, does not afford to offload the remaining public-key operations of the certificate-based handshake. Contrary to our handshake delegation architecture, SCVP, therefore, still requires constrained devices to implement public-key cryptography. Moreover, by exchanging certificate information with the validation server on a per handshake basis, SCVP further increases the overall transmission overhead of the certificate-based handshake.

Finally, we note that the design of our handshake delegation architecture is inspired by the architectural model of the Kerberos protocol [NYHR05]: Comparable to a Kerberos KDC, we employ a central entity, i.e., the delegation server, to facilitate authentication of the intended communication end-points. In addition, our handshake delegation procedure similarly relies on the exchange of protected session tickets. In contrast to Kerberos, we, however, specifically design our architecture with memory-constrained devices and inter-domain communication scenarios in mind. Our handshake delegation procedure, e.g., does not rely on timestamps in the session tickets and forgoes the need for global time synchronization. As a result, our architecture can achieve lower ROM overheads than Kerberos in the context of constrained devices [Miy10]. This is especially true when considering the need to employ additional end-to-end security mechanisms for the secure transmission of application data in case of Kerberos. Moreover, it is important to note that, contrary to Kerberos, our handshake delegation architecture does not require constrained devices and remote communication end-points to belong to the same (federated) trust domain.

5.5.2 Authorization Frameworks for the IP-based IoT

Concerning the authorization of network interactions in the IP-based IoT, the CoAP protocol specification [SHB14] proposes the use of local Access Control Lists (ACLs) on a constrained device as a simple mechanism to restrict access on a per-end-point and a per-resource basis. With an increasing number of authorized end-point and resource combinations, such local ACLs, however, quickly cause extensive memory and management overheads [RP12]. Role-based (e.g., [GMW10]) or attribute-based (e.g., [OASIS13]) authorization frameworks afford to reduce these overheads by introducing intermediate levels in the authorization hierarchy. This enables an assignment of access rights to a set of abstract roles or attributes instead of directly assigning

Approach	Token verification	Transm. overhead	Revocation	Time sync.
CapBAC	$-$ (PK crypto)	$-$ (per request)	$\approx$	required
DCapBAC	$-$ (PK crypto)	$-$ (per request)	$-$	required
Seitz et al.	$\approx$ (Sym. crypto)	$-$ (per request)	$-$	required
IoT-OAS	$-$ (Netw. interact.)	$-$ (per request)	$+$	not needed
DCAF	$\approx$ (Sym. crypto)	$-$ (per session)	$\approx$	optional

Table 5.2 High-level comparison of the discussed capability-based authorization frameworks. These frameworks exhibit inferior ($-$), similar ($\approx$), or improved ($+$) properties when compared to our handshake delegation architecture. In case the authors did not quantify a property, the presented result denotes an estimation based on the provided architectural descriptions.

them to the communication end-points. The end-points, in turn, are assigned roles or attributes according to their intended access rights. Based on these roles or attributes, the constrained device then has to determine if an end-point is allowed to access the requested resource. The corresponding policy evaluation procedure, however, often incurs extensive computation and memory overheads [SSG13].

By delegating the derivation of the authorization decision to a trusted network entity, capability-based security [RP12] – as employed in our handshake delegation architecture – allows to largely prevent the above authorization-related overheads at a constrained device. To this end, capability-based security binds the authorization decision to a cryptographically verifiable authorization token that typically contains a reference to the authorized communication end-point, the authorized resources, and the granted access rights. Consequently, when receiving such an authorization token, the constrained device only needs to verify the authenticity and the freshness of the received token as well as to match the contained authorization information to the requesting end-point and the requested resource for *policy enforcement* purposes.

Several authorization frameworks that are based on capability-based security have recently been presented in the context of constrained devices (see Table 5.2 for a comparison overview). With CapBAC [RP12] and DCapBAC [HRPJ$^+$15], the authors propose to use public-key signatures and certificates in order to identify the involved network entities in the authorization tokens. In contrast, our handshake delegation architecture solely relies on symmetric-key cryptography. Thus, CapBAC and DCapBAC are prone to cause significantly higher transmission and token verification overheads at a constrained device than our presented architecture.

Similarly, Seitz et al. [SSG13] presented an authorization framework that transfers authorization information on a per-request basis. In contrast to CapBAC and DCapBAC, this approach employs efficient symmetric-key cryptography for token protection purposes. By transferring authorization tokens on a per-request instead of a per-session basis, their approach, however, still causes increased transmission overheads compared to our handshake delegation architecture. Furthermore, while our architecture allows for efficient, explicit authorization revocation, Seitz et al. rely on timestamps and short-lived authorization tokens for revocation purposes. Consequently, their approach requires constrained devices to have synchronized clocks.

Cirani et al. [CPG$^+$15] recently proposed IoT-OAS, an adapted OAuth authorization architecture for the IoT that delegates the granting and the verification of OAuth access tokens [Har12] from a constrained device to a dedicated, trusted OAuth service. In contrast to our handshake delegation architecture, IoT-OAS, however, incurs significant transmission overheads in the context of constrained devices. This is primarily because IoT-OAS requires a constrained device to contact the OAuth service on a per-request basis for token verification purposes. Conversely, our handshake delegation procedure conveys all authorization-related information in the session ticket when (re-)establishing a secure connection. This enables the constrained device to perform policy enforcement without the need for further network interactions.

Finally, Gerdes et al. [GBB14, GBB15] recently presented DCAF, a capability-based framework that integrates the transfer of an authorization token with the symmetric-key-based DTLS handshake. Similar to our presented architecture, DCAF thereby binds the exchanged authorization information to the established session context. Their approach differs from ours by requiring a constrained device to request an authorization token prior to the connection establishment and to transfer the corresponding authorization information to the communication partner during a symmetric-key-based handshake. In contrast, with our handshake delegation procedure, a constrained device only has to perform an abbreviated session resumption handshake. DCAF, therefore, is likely to cause a higher transmission overhead.

5.6 Conclusion

In this chapter, we analyzed the impact of public-key cryptography on the RAM and ROM requirements of a certificate-based DTLS implementation for constrained devices. As similarly indicated by the evaluation results for HIP DEX, our analysis revealed extensive memory overheads that render a comprehensive, public-key-enabled protocol implementation infeasible for a wide range of memory-constrained devices. To still enable these devices to communicate securely via standard end-to-end IP security protocols, we introduced the handshake delegation architecture that allows to delegate the initial public-key-based connection establishment handshake to an unconstrained delegation server. As a result, constrained devices can rely on an abbreviated session resumption handshake and efficient symmetric-key cryptography for the protection of application data. Moreover, by leveraging the central role of the delegation server during the initial connection establishment, handshake delegation also allows to authorize new and revoke existing connections. Finally, our architecture is not limited to DTLS, but also affords a memory-efficient mode of operation and limited authorization capabilities for HIP DEX and Minimal IKEv2.

As the evaluation results show, our handshake delegation architecture achieves an overall RAM and ROM reduction of about 31 kB (i.e., 64 %) compared to a certificate-based DTLS implementation. Likewise, the handshake delegation procedure reduces the computation overhead by 97 % and the transmission overhead by 68 % in the context of constrained devices. With these results, the presented architecture achieves similarly low computation and transmission overheads as a purely symmetric-key-based DTLS handshake while providing additional authorization capabilities.

Overall, we conclude that the handshake delegation architecture provides a comprehensive, yet compact solution for authentication, authorization, and secure data transmission for the IoT. Still, the evaluation results also indicate non-negligible packet fragmentation at the 6LoWPAN layer when performing the handshake of standard end-to-end IP security protocols inside a constrained node network. In the following chapter, we will show that such packet fragmentation is vulnerable to potential DoS attacks. Moreover, we will present lightweight defense mechanisms that protect constrained devices against the identified fragmentation-based DoS attacks.

6

Secure 6LoWPAN Fragmentation

The evaluation results in Chapters 4 and 5 show that the IoT security protocol adaptations DTLS and HIP DEX commonly involve handshake messages that exceed the maximum frame size of constrained link layer technologies such as IEEE 802.15.4. To still afford the transmission of these messages inside a constrained node network, the 6LoWPAN adaptation layer provides a *fragmentation mechanism* for exceedingly large IPv6 packets (see Section 2.4). As we identify in this chapter, this mechanism, however, is vulnerable to potential DoS attacks. This enables an adversary to block the establishment of secure end-to-end connections despite our previous efforts.

Our contributions in this chapter are twofold. First, we present two design-level DoS attacks that we identified during our security analysis of the 6LoWPAN fragmentation mechanism in the context of constrained devices. The *fragment duplication attack* enables an eavesdropping adversary to reactively block the reassembly of an overheard fragmented IPv6 packet by sending a single forged 6LoWPAN fragment to a target device. Similarly, the *buffer reservation attack* enables an adversary without overhearing capabilities to proactively block the packet reassembly at a reassembling node by maliciously reserving the reassembly buffer via a single protocol-compliant 6LoWPAN fragment. As a second contribution, we then introduce two complementary, lightweight defense mechanisms that mitigate the identified fragmentation attacks at the 6LoWPAN layer. The *content-chaining scheme* protects against the fragment duplication attack by offering efficient per-fragment data origin authentication. In addition, the *split buffer approach* fosters competition for the scarce buffer resources at a reassembling node between an adversary and legitimate fragment senders. In combination with a dedicated packet discard strategy that penalizes suspicious sending behavior, this allows to effectively mitigate the buffer reservation attack. The evaluation results confirm the practicability of the identified 6LoWPAN fragmentation attacks and show the effectiveness of the presented defense mechanisms at moderate computation, energy, memory, and transmission trade-offs.

The remainder of this chapter is structured as follows. In Section 6.1, we provide an overview of previously identified fragmentation attacks as a basis for our work in this chapter. Next, Section 6.2 describes the underlying attacker model of our security analysis and Section 6.3 presents the detailed analysis results. We then introduce the design of our two defense mechanisms. Section 6.4 thereby focuses on the content-chaining scheme, whereas Section 6.5 describes the split buffer approach. In Section 6.6, we discuss the security considerations of the presented defense mechanisms. We then present the evaluation results in Section 6.7. Finally, Section 6.8 discusses related work and Section 6.9 concludes this chapter with a summary. We note that the contents of this chapter is based on our published work in [HHW+13].

6.1 Overview of Existing Fragmentation Attacks

Fragmentation has been shown to be harmful in a variety of network scenarios. Bittau et al. [BHL06], e.g., exploit fragmentation at the IEEE 802.11 MAC layer to considerably increase the efficiency of breaking the encryption of WEP-protected wireless channels. Similarly, Albrecht et al. [APW09] exploit ciphertext fragmentation in order to recover plaintext information from SSH-protected tunnels. Moreover, fragmentation-based vulnerabilities have extensively been investigated in the context of the IPv4 and IPv6 protocols. We note that our following discussion focuses on these IP-level fragmentation attacks as these are strongly related to the attacks that we identify with respect to the 6LoWPAN fragmentation mechanism. For our discussion, we classify these existing IP-level fragmentation attacks according to their root cause, i.e., implementation deficiencies and design-level vulnerabilities.

6.1.1 Attacks Based on Implementation Deficiencies

One of the most well-known fragmentation attacks exploiting implementation deficiencies of the IPv4 fragmentation mechanism is the “Ping of Death” [CERT96]. It is based on sending IPv4 fragments to a victim host, which, when reassembled, exceed the maximum IPv4 packet size of 65535 byte. This attack caused vulnerable systems to crash on packet reassembly. Similarly, the “Teardrop” attack [CERT97] exploits the fact that several operating systems handled overlapping IPv4 fragments incorrectly, causing high computational load or crashes of the victim host. Notably, such implementation-specific attacks can commonly be prevented by patching the vulnerable reassembly routines. We, therefore, do not specifically consider these types of attacks in this chapter and instead focus on design-level vulnerabilities.

6.1.2 Attacks Based on Design-Level Vulnerabilities

In contrast to the above implementation-related attacks, design-level IP fragmentation attacks exploit inherent vulnerabilities in the IPv4 or IPv6 protocol specification. Correspondingly, most design-level IP fragmentation attacks resulted in revised protocol specifications. These specification updates aim at mitigating or preventing the identified fragmentation attacks via enhanced fragmentation and reassembly policies. We highlight these improvements in our following discussion as the design of the 6LoWPAN fragmentation mechanism employs the most recent policies for the

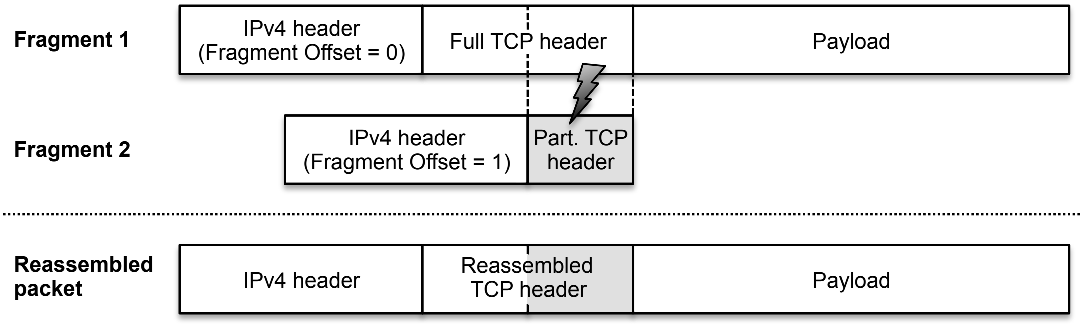


Figure 6.1 Packet filter evasion based on overlapping fragments. The on-path packet filter inspects the first fragment that appears to be legitimate and applies the same filter decision to the second packet fragment. The second fragment, however, overwrites, e.g., the TCP flags in the header of the transport layer during packet reassembly at the victim host (marked in gray). This effectively conceals the actual TCP flag combination from the packet filter.

IPv6 protocol. To structure our discussion of design-level IP fragmentation attacks, we further differentiate these attacks based on the exploited packet property, i.e., *overlapping fragment content* and *fragmented packets with missing packet fragments.*

6.1.2.1 Attacks Exploiting Overlapping Fragment Content

In 1995, Ziemba et al. [ZRT95] showed that overlapping IPv4 fragment content can be exploited by an adversary to evade on-path packet filters such as firewalls. Ptacek et al. [PN98] subsequently presented similar attacks targeting Intrusion Detection Systems (IDSs). The basic idea behind these fragmentation attacks is that *non-reassembling* network elements typically apply their filter rules based on the header information contained in the *first* IPv4 fragment and perform the resulting action on all remaining packet fragments. The reassembly algorithm outlined in the IPv4 protocol specification [Pos81], however, prefers the most recent fragment content over previously received content in case of overlapping fragments as depicted in Figure 6.1. This enables an adversary to circumvent filter rules at a network element that, e.g., block inbound TCP handshakes based on the SYN flag in the TCP header.

To mount this type of fragmentation attack, the adversary generates an initial IPv4 fragment that contains legitimate header information such as the ACK flag in the TCP header. A subsequent fragment, however, indicates a fragment offset that causes the corresponding fragment content to overlap with the initial IPv4 fragment (see Figure 6.1). This overlapping content then contains the header information that the adversary intends to be processed at the victim host, e.g., the TCP SYN flag.

With the goal to prevent the above attack, the packet filter and reassembly policies of the IPv4 protocol were updated to discard packet fragments that overlap at the transport layer. These updated policies utilize the fact that the payload of the first IPv4 fragment always starts with the transport protocol header [Pos81]. Thus, static policy rules based on the fragment offset suffice to prevent overlapping fragments from overwriting transport header information [ZRT95, Mil01]. One of these static rules, e.g., drops IPv4 fragments with a fragment offset of 1 in case of TCP as this offset would overwrite the second byte of the TCP header (see Figure 6.1).

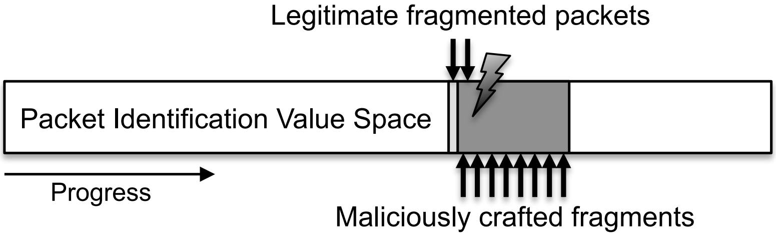

Legitimate fragmented packets
Packet Identification Value Space
Progress
Maliciously crafted fragments

Figure 6.2 Blocking packet reassembly with overlapping fragments. The adversary maliciously occupies parts of the packet identification value space at a victim host that will be used in the near future to identify associated fragments of legitimate packets during packet reassembly.

The original IPv6 specification [DH98] did not prevent overlapping fragment content either, i.e., like the IPv4 standard. As a result, Davies et al. [DKS07] also identified overlapping fragment content to cause overwriting vulnerabilities in the context of IPv6 packet fragmentation. IPv6 packets, however, may carry optional IPv6 extension headers in their *fragmentable* part. Thus, reassembly policies based on static fragment offsets would not suffice to prevent overwriting of the transport protocol header. To still protect IPv6-enabled hosts and network elements against overlapping fragment content, the IPv6 reassembly policy was updated to discard the *entire* IPv6 packet in case of overlapping fragment content [Kri09]. Importantly, this revised policy also was adopted by the 6LoWPAN standard [MKHC07].

Despite the above modifications to the IPv4 and IPv6 protocol specifications, the authors of [Gon11, GH11, GH13] recently discovered that overlapping fragments can still be exploited by an adversary in order to mount a DoS attack against a victim host. To this end, the adversary must first guess the current *packet identification value* of a legitimately fragmented IP packet, i.e., the IP header information allowing a reassembling host to correlate fragments that belong to the same overall IP packet. The adversary then sends multiple spoofed IP fragments to the victim host containing packet identification values that will soon be used by the legitimate communication partners (see dark gray area in Figure 6.2). These malicious IP fragments cause overlapping fragment content at the victim host once the legitimate IP fragments with matching packet identification values arrive. As a result, the victim host potentially reassembles the fragmented IP packet incorrectly (IPv4) or disposes of the entire legitimate IP packet (IPv6). This enables the adversary to block the correct reassembly of legitimate fragmented IP packets at the victim host.

6.1.2.2 Attacks Exploiting Fragmented Packets with Missing Fragments

In addition to overlapping fragment content, fragmented IP packets with missing packet fragments have also been shown to be a viable attack vector against IP-enabled hosts. Specifically, IP fragmentation requires a reassembling host to allocate part of its memory resources for packet reassembly purposes, i.e., until the *fully reassembled* IP packet can be passed up the network stack for further processing. These memory resources are either limited by the size of a dedicated memory pool or by the overall memory resources that are available at the reassembling host. As the authors of [KPS03, DKS07, HMC07, Gon11] identify, an adversary can exploit this circumstance to mount a fragmentation-based DoS attack against a victim host by maliciously occupying *all* of the available reassembly buffer resources.

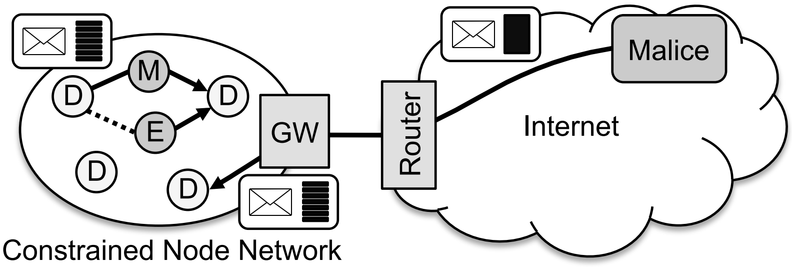


Figure 6.3 The attacker model for our security analysis of the 6LoWPAN fragmentation mechanism. The adversaries Eve (E) and Mallory (M) are co-located with constrained devices (D) inside a constrained node network. Malice is situated outside the target network. Arrows indicate specific packet forwarding paths. The dashed line represents overhearing.

To mount this type of attack, the adversary floods a victim host with multiple *incomplete* fragmented IP packets that never reassemble to a complete IP packet. Once its reassembly buffer resources are exhausted, the victim host then no longer can reassemble newly arriving packets until the reassembly procedure disposes of the partially received packets. Hence, missing packet fragments also enable an adversary to block the reassembly of legitimate fragmented IP packets at a victim host.

Based on this understanding of existing IP-level fragmentation attacks, we now present the design-level DoS attacks that we identified during our security analysis of the 6LoWPAN fragmentation mechanism in the context of constrained devices. We begin this discussion with an introduction of the assumed attacker model.

6.2 The Assumed Attacker Model

For our security analysis of the 6LoWPAN fragmentation mechanism, we used the Internet Threat Model (see Section 3.1.1) as a basis. This threat model, however, describes a *worst case* attack scenario by assuming that the adversary has full control over the network infrastructure employed by the victim host. To allow for a more fine-granular analysis of the attack capabilities required to mount a fragmentation attack at the 6LoWPAN layer, we, therefore, further distinguish between three types of adversaries within the Internet Threat Model, i.e., Eve, Mallory, and Malice.

As depicted in Figure 6.3, we assume *Malice* to be a network-external adversary. *Eve and Mallory*, contrarily, are situated inside the target network. Moreover, we assume Eve and Mallory to actively participate in the routing topology. To achieve such participation, Eve or Mallory may simply be placed within radio range of a legitimate node and join the routing topology if the target network does not restrict network admission. Contrarily, a protected target network that, e.g., employs link layer security, first requires the adversary to gain access to the corresponding security keys to achieve network admission for Eve or Mallory. To this end, the adversary may, e.g., try to extract the security keys from a legitimate constrained device of the target network [HBH$^+$05, BBD06]. Alternatively, the adversary may also attempt to remotely take control over a legitimate constrained device and enforce adversarial behavior. Such remote exploitation has recently been shown to also be viable in the context of constrained devices. Specifically, smart fridges and televisions were reported to have been utilized maliciously in recent SPAM attacks [Pro14a, Pro14b].

With respect to the attack capabilities of Eve and Mallory, we note that these network-internal adversaries can send network packets to any constrained device inside the target network. As illustrated in Figure 6.3, their attack capabilities, however, differ due to their relative position to the forwarding path of 6LoWPAN-fragmented packets. Specifically, we assume Eve to be located *besides* the fragment forwarding path. As a result, she can overhear legitimate packets and send packets in reaction to overheard information. In contrast, we assume Mallory to be located *on* the fragment forwarding path. Thus, in addition to Eve's capabilities, Mallory can also alter, delay, reorder, or simply drop legitimate packets. Mallory's capabilities, therefore, allow her to mount at least the attacks that are viable for Eve. For this reason, we do not mention Mallory explicitly when discussing Eve.

Regarding the network-external adversary *Malice*, we assume this adversary to be equipped with significantly more resources than a constrained device and to be interconnected via a high-bandwidth network connection. This resource advantage enables Malice to flood constrained devices with numerous IPv6 packets. Moreover, Malice may send large IPv6 packets that exceed the frame size of the constrained link layer technology employed by the target network. Such exceedingly large IPv6 packets then must be fragmented at the 6LoWPAN layer of the interconnecting gateway before transmission inside the constrained node network (see Figure 6.3).

6.2.1 Remarks About the Network-External Attacks

The network-external adversary Malice is restricted to IP-based communication via the gateway and *cannot* directly employ 6LoWPAN fragments in her attacks. This prevents Malice from exploiting potential design-level vulnerabilities in the 6LoWPAN fragmentation mechanism. As a result, Malice has to resort to flooding-based attacks. She, however, can further amplify these attacks by sending large IPv6 packets that require 6LoWPAN fragmentation at the interconnecting gateway.

Notably, such flooding-based attacks can typically be detected and protected against via standard security mechanisms at the gateway such as authenticated tunnels or rate limitation approaches. Authenticated tunnels allow to frustrate the above flooding attack by enabling the gateway to exclude network-external hosts from communication once it detects an excessive amount of large IPv6 packets. Similarly, rate limitation of large inbound IPv6 packets at the gateway prevents constrained devices from being overloaded by vast amounts of 6LoWPAN fragments. This allows to mitigate the adverse effects of Malice's resource advantage over constrained devices.

The above defense mechanisms, however, still leave the constrained node network unprotected against *network-internal* adversaries who do not resolve to flooding-based attacks and that appear to be benign network participants. We, thus, focus our security analysis on the challenging case of the *standard-compliant* adversaries Eve and Mallory that exploit vulnerabilities in the 6LoWPAN protocol design and the resource constraints in the target networks. Moreover, we make no assumptions about the hardware resources of Eve or Mallory. Hence, even resource-constrained adversaries can mount the 6LoWPAN fragmentation attacks described below.

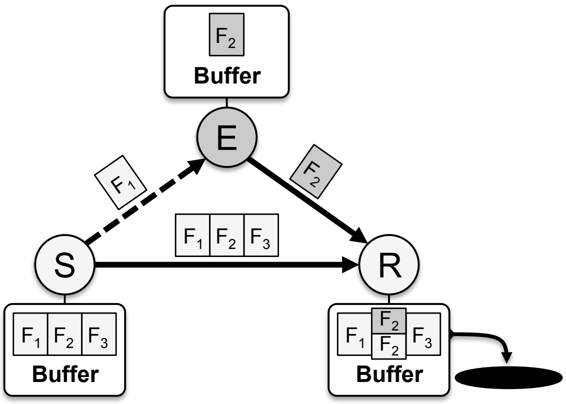


Figure 6.4 The fragment duplication attack. Eve (E) overhears 6LoWPAN fragments from the legitimate sender (S) and injects a spoofed fragment with altered fragment content. Due to overlapping fragment content, the recipient (R) drops the entire IPv6 packet.

6.3 The 6LoWPAN Fragmentation Attacks

In this section, we present the two design-level DoS attacks that we identified during our security analysis of the 6LoWPAN fragmentation mechanism, i.e., the *fragment duplication attack* and the *buffer reservation attack*. Moreover, we discuss their relation to the previously described IP-level fragmentation attacks in Section 6.1.2. We note that, for the remainder of this chapter, the term “fragmentation” refers to the IPv6 packet fragmentation *at the 6LoWPAN layer* if not mentioned otherwise.

6.3.1 The Fragment Duplication Attack

The fragment duplication attack leverages the circumstance that the recipient of a 6LoWPAN fragment cannot unambiguously identify at the 6LoWPAN layer if this fragment belongs to the indicated fragmented IPv6 packet or if it was maliciously crafted by an adversary. Moreover, the attack exploits the fact that the 6LoWPAN standard recommends IPv6 packets with overlapping 6LoWPAN fragment content to be discarded. Combined, these factors enable Eve to *selectively* block the reassembly of a fragmented IPv6 packet by sending a single *duplicate* 6LoWPAN fragment.

To show how such a selective attack would proceed, we assume that Eve aims at preventing DTLS-protected end-to-end communication between two legitimate end-points. For this, Eve inspects the wireless medium for 6LoWPAN-fragmented DTLS handshake messages, e.g., by checking the overheard FRAG1 fragments for DTLS header information. Once, Eve discovers a 6LoWPAN fragment with DTLS-related packet content, she injects a spoofed FRAGN fragment (see Figure 6.4). This FRAGN contains *arbitrary* fragment payload and a 6LoWPAN fragmentation header that links her FRAGN to the fragmented DTLS message via a spoofed datagram tag (see Section 2.4.2 for detailed information about the 6LoWPAN fragmentation header). As the fragment recipient cannot distinguish such spoofed FRAGNs from legitimate ones, it cannot decide which fragments to use during its packet reassembly at the 6LoWPAN layer. Moreover, due to the fragment overlap of the spoofed and the legitimate FRAGNs, a standard-compliant recipient has to discard the entire IPv6 packet, thus affording Eve to block the overheard DTLS handshake message.

6.3.1.1 General Interference via the Fragment Duplication Attack

Besides such selective blocking of the correct 6LoWPAN packet reassembly, Eve can also unconditionally block the delivery of *any* fragmented IPv6 packet in her vicinity. To this end, she merely needs to inject duplicate FRAGNs for each overheard fragmented packet. Eve then achieves general interference with the 6LoWPAN packet reassembly of all fragmented packets within her wireless radio range. Moreover, Eve may exploit such general interference in an *energy exhaustion attack*. Specifically, we observe that upper layer protocols typically employ retransmissions to recover lost application data. This, e.g., is the case for confirmable messages of the CoAP protocol [SHB14]. By also blocking the correct reassembly of these retransmissions, Eve then can further deplete the potentially scarce energy resources of the legitimate fragment sender as well as of the forwarding nodes on the communication path.

With respect to the previously discussed IP-level fragmentation attacks, we note that the fragment duplication attack is similar to the IP-level fragmentation attacks based on overlapping fragment content presented in Section 6.1.2.1. However, in contrast to these attacks, Eve does not require guessing of the 6LoWPAN datagram tag due to the broadcast nature of the wireless medium and the multi-hop topology of constrained node networks. Instead, she can overhear the wireless medium and react to overheard packets. This enables Eve to mount a notably cheap attack that only requires her to send a single duplicate 6LoWPAN fragment per overheard IPv6 packet in order to prevent the correct packet reassembly at the victim node. This stands in stark contrast to the fragmentation attack discussed in Section 6.1.2.1 that requires guessing of the current IP packet identification value as well as the transfer of multiple packet fragments to increase the probability of a correct guess.

6.3.2 The Buffer Reservation Attack

In contrast to the fragment duplication attack, the buffer reservation attack exploits the scarce memory resources of constrained devices and leverages the fact that a fragment recipient cannot determine a-priori if *all* 6LoWPAN fragments of a fragmented IPv6 packet will eventually arrive. As a result, the recipient of a 6LoWPAN fragment must *optimistically* store the 6LoWPAN fragments of an IPv6 packet until this packet has been received completely. For in-place packet reassembly, this requires the recipient to *reserve* buffer resources that suffice for the entire expected IPv6 packet content. We note that the corresponding size information is indicated in the 6LoWPAN header of each received fragment as described in Section 2.4.2.

Importantly, the fragment recipient has to discard the received 6LoWPAN fragments from other fragmented IPv6 packets once its available reassembly buffer resources are exhausted. Eve can exploit this fact to block the reassembly of fragmented IPv6 packets at a target device by *reserving* all available reassembly resources with maliciously crafted 6LoWPAN fragments. For our discussion of the buffer reservation attack, we assume that the devices inside the constrained node network only have a single 6LoWPAN buffer for reassembly purposes. This, e.g., is the case when employing the Contiki OS [DGV04] as the underlying operating system for constrained devices. Moreover, we note that, if a constrained device has sufficient memory re-

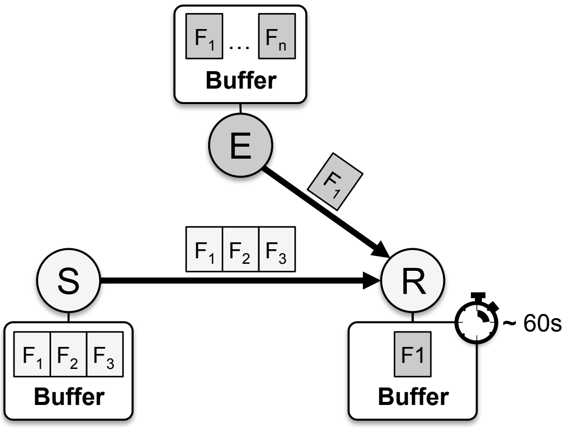


Figure 6.5 The buffer reservation attack. Eve (E) sends a single 6LoWPAN fragment to the resource-constrained recipient (R). Eve, however, does not complete the partial IPv6 packet with the remaining 6LoWPAN fragments. As a result, the reassembly buffer at the recipient is maliciously occupied until the reassembly timeout expires. During this time, the recipient cannot process other fragmented packets that originate from a legitimate sender (S).

sources to maintain multiple 6LoWPAN reassembly buffers, this linearly increases the effort that an adversary has to put into mounting the buffer reservation attack.

6.3.2.1 The Basic Buffer Reservation Attack

When mounting the buffer reservation attack, Eve generates a fragmented IPv6 packet with arbitrary payload as illustrated in Figure 6.5. She then, however, only sends the first 6LoWPAN fragment, i.e., the FRAG1, to the target device. If the reassembly buffer of the target device is not yet occupied by another fragmented IPv6 packet, the received FRAG1 reserves the target's reassembly buffer for Eve's fragmented IPv6 packet. Eve now refrains from sending the remaining FRAGN fragments to ensure that her fragmented IPv6 packet never reassembles at the target device. As a result, Eve's attack FRAG1 blocks the reassembly buffer at the target device until the 6LoWPAN reassembly timeout expires. We note that this reassembly timeout can last up to 60 seconds according to the 6LoWPAN standard [MKHC07].

The above attack only persists for the duration of a single 6LoWPAN reassembly timeout. To continuously block the packet reassembly at the target device, Eve either needs to constantly *flood* the target device with FRAG1s or she has to *time* her attack fragments according to the reassembly timeout. Flooding the target device with FRAG1s denotes a crude form of attack that may easily be detected by monitoring the wireless medium. The timing-based attack, contrarily, is more lenient on Eve's resources and is significantly more difficult to detect when the monitoring wireless transmissions inside a constrained node network. For this reason, we now describe how Eve can perform the timing-based buffer reservation attack.

6.3.2.2 The Timing-Based Buffer Reservation Attack

To mount the continuous timing-based buffer reservation attack, Eve must ensure that the target device receives her FRAG1 of a subsequent buffer reservation attack

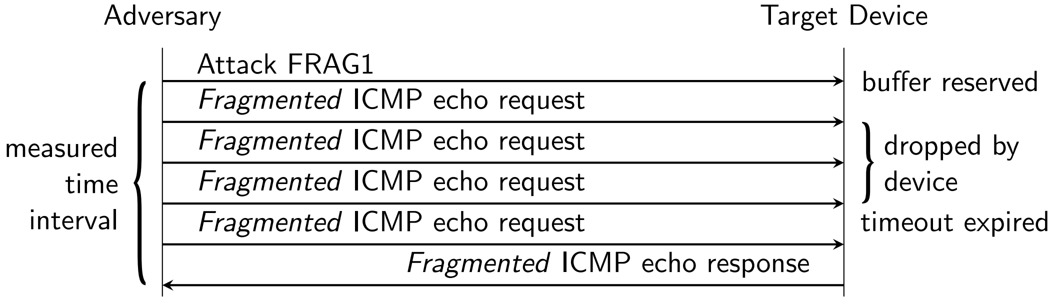

Adversary
Target Device
Attack FRAG1
buffer reserved
Fragmented ICMP echo request
Fragmented ICMP echo request
Fragmented ICMP echo request
Fragmented ICMP echo request
measured time interval
dropped by device
timeout expired
Fragmented ICMP echo response

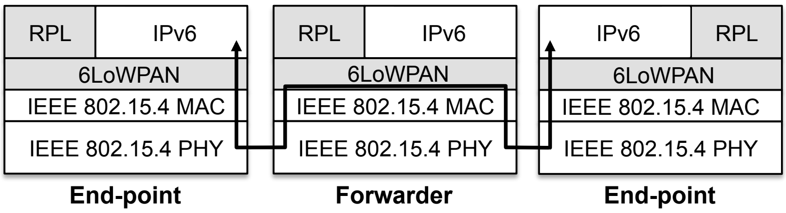
Figure 6.6 Probing procedure for the continuous timing-based buffer reservation attack. Eve first sends an attack FRAG1 to reserve the reassembly buffer at the target device. She then sends a sequence of fragmented ICMP echo requests that trigger a response from the target device. By measuring the time between sending the FRAG1 and receiving the first ICMP echo response, Eve can estimate the time interval required for the buffer reservation attack.

immediately after the reassembly timeout of the current attack has expired. Eve, thus, has to probe the target device in preparation of her attack in order to learn the employed reassembly timeout and to approximate the fragment transmission time to this device as shown in Figure 6.6. To this end, Eve first sends an attack FRAG1 that reserves the reassembly buffer at the target device. She then sends a sequence of complete fragmented IPv6 packets that trigger a response from the target device, e.g., ICMP echo requests with a large data field. These messages represent control packets for Eve that allow her to verify if the buffer reservation attack succeeded and how long it persists. In case of a successful attack, Eve only receives a reply from the target device for ICMP messages that arrived after the reassembly timeout expired. This enables Eve to estimate the reassembly timeout of the target node and the packet propagation time by measuring the duration between the FRAG1 and the first ICMP reply reduced by half the reported RTT. Based on this information, Eve then is able to continuously reserve the reassembly buffer at the target device by iteratively mounting the buffer reservation attack as described above.

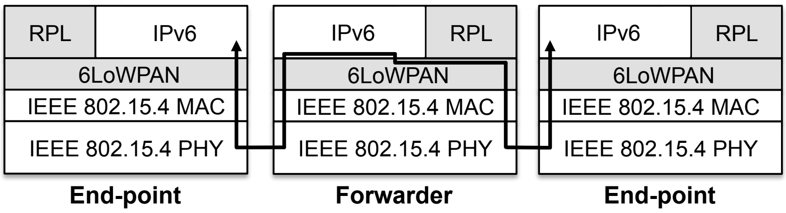
Regarding the previously discussed IP-level fragmentation attacks, we note that the buffer reservation attack exploits the same properties of the packet reassembly mechanism as the IP-level fragmentation attack based on missing packet fragments presented in Section 6.1.2.2. The impact of the 6LoWPAN-based attack, however, is significantly aggravated by the circumstance that constrained devices commonly are equipped with very limited RAM resources. This enables Eve to maliciously reserve the entire reassembly buffer of a target device for the duration of the reassembly timeout by sending merely a single protocol-compliant 6LoWPAN fragment.

6.3.3 Susceptibility to the 6LoWPAN Fragmentation Attacks

As described in the previous sections, the fragment duplication attack and the buffer reservation attack allow Eve to exploit the 6LoWPAN fragmentation facilities to mount two design-level DoS attacks against devices that are located inside the same constrained node network as Eve. Notably, the susceptibility to these attacks is limited to constrained devices that try to reassemble Eve's malicious 6LoWPAN fragments. While this property would restrict the identified attacks to the IPv6 packet



(a) Mesh-under routing



(b) (Enhanced) route-over routing

Figure 6.7 The packet processing path inside a constrained node network that employs mesh-under routing or (enhanced) route-over routing. While forwarding nodes can remain oblivious to the 6LoWPAN packet fragmentation in case of mesh-under routing, these node are required to act on 6LoWPAN-fragmented IPv6 packets in case of (enhanced) route-over routing.

destination for regular IP-level fragmentation[1], the 6LoWPAN standard requires constrained devices *on the forwarding path* to reassemble traversing fragmented packets depending on the employed 6LoWPAN fragment forwarding mechanism. Hence, these devices are also susceptible to the identified 6LoWPAN fragmentation attacks.

We now briefly introduce the different fragment forwarding mechanisms that are supported by the 6LoWPAN protocol specification and then analyze the susceptibility of constrained devices based on the key properties of these approaches.

6.3.3.1 A Brief Introduction to 6LoWPAN Fragment Forwarding

Opposed to IPv6 fragments, 6LoWPAN-fragmented IPv6 packets only contain IPv6 header information in the FRAG1 fragment (see Figure 2.8). FRAGNs, in contrast, do not contain routable IP-level information. This omission of the IPv6 header information in the FRAGNs stems from the goal to keep the overhead of fragmented IPv6 packets low. As a result, only the fragment content of the FRAG1 can be routed immediately to the IPv6 packet destination. FRAGNs first need to be associated to the corresponding FRAG1 for fragment forwarding purposes. This particular design trait has implications on the 6LoWPAN fragment forwarding inside a constrained node network. The following three fragment forwarding mechanisms are commonly distinguished in the context of 6LoWPAN: (i) mesh-under, (ii) route-over, and (iii) enhanced route-over routing [MKHC07, SB10]. We now outline these forwarding mechanisms and then discuss their susceptibility in the following sections.

With **mesh-under routing**, the fragment forwarding nodes derive the forwarding decision at the link layer and do not consult the IPv6 layer for routing purposes (see Figure 6.7a). As a result, *only the final destination* inside the constrained node

---

[1] With IPv6, packet fragmentation and packet reassembly must only be performed by the communication end-points according to the IPv6 protocol specification [DH98].

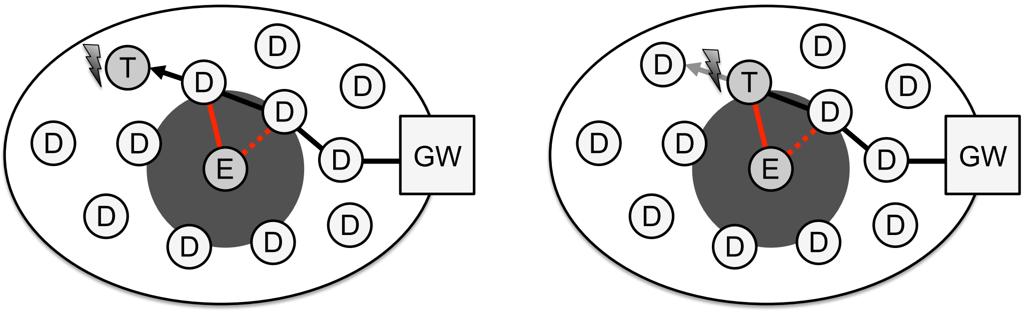
network has to *reassemble* a fragmented IPv6 packet, whereas the forwarding nodes can remain oblivious to the packet fragmentation at the 6LoWPAN layer. Moreover, as the forwarding nodes are able to derive the forwarding decision on a per-fragment basis, the *forwarding path may differ* across the individual 6LoWPAN fragments.

Contrary to mesh-under routing, **route-over routing** derives the forwarding decision at the network layer. To this end, each receiving node must first *reassemble* the entire IPv6 packet at the 6LoWPAN layer. The node then passes the reassembled packet to the IPv6 layer for further processing (see Figure 6.7b). The IPv6 layer, in turn, checks if the packet is destined for the local device or for another communication end-point. In the latter case, the IPv6 layer looks up the next hop and passes the packet down to the 6LoWPAN layer again. Here, the IPv6 packet is re-compressed and re-fragmented. Importantly, this packet processing procedure implies that all 6LoWPAN fragments of an IPv6 packet are transferred along the *same forwarding path* as the intermediate hops apply the routing decision on a per-packet basis.

To afford a similar forwarding efficiency as the mesh-under approach, **enhanced route-over routing** forgoes the packet reassembly at the forwarding nodes as an improvement to route-over routing. More precisely, the fragment recipient derives the forwarding decision immediately after a FRAG1 has been received, i.e., *without prior reassembly* of the entire IPv6 packet. To this end, the 6LoWPAN layer reconstructs the IPv6 header from the FRAG1 and derives the forwarding decision at the IPv6 layer via cross-layer interactions. The 6LoWPAN layer then stores this forwarding decision for the transmission of the associated FRAGNs and forwards the FRAG1 to the next hop on the forwarding path. Thus, FRAGNs can be forwarded individually along a virtual circuit that is laid out by the FRAG1 without the need for intermediate packet reassembly. In other words, all 6LoWPAN fragments of a fragmented IPv6 packet are transmitted along the *same forwarding path*.

6.3.3.2 Susceptibility in Case of Mesh-under Routing

With mesh-under routing, fragment forwarding nodes inside the constrained node network do not reassemble Eve's maliciously crafted 6LoWPAN fragments. Instead, they forward Eve's attack fragments without realizing that she is actually mounting



(a) Mesh-under and enhanced route-over **(b)** Route-over

Figure 6.8 Susceptibility of constrained devices (D) against the identified 6LoWPAN fragmentation attacks depending on the employed fragment forwarding mechanism. Mesh-under and enhanced route-over routing allow Eve (E) to target constrained devices (T) beyond her one-hop neighborhood. With route-over routing, she is limited to her one-hop neighborhood.

an attack. As shown in Figure 6.8a, this enables Eve to target the final fragment destination inside the constrained node network with the identified 6LoWPAN fragmentation attacks. More precisely, Eve is bound to the specific destination of the overheard 6LoWPAN fragments when mounting the fragment duplication attack. She, however, does not have such restrictions in case of the buffer reservation attack. In fact, Eve then only needs to know the IPv6 address of the target device.

6.3.3.3 Susceptibility in Case of Route-over Routing

In case of route-over routing, Eve can interfere with the 6LoWPAN packet reassembly of the constrained devices that are located in her one-hop neighborhood. This is due to the fact that each node on the forwarding path reassembles Eves maliciously crafted 6LoWPAN fragments. As a result, both identified 6LoWPAN fragmentation attacks immediately affect the first hop on the forwarding path towards the final destination of Eve's attack fragments (see Figure 6.8b). Conversely, Eve cannot target the packet reassembly of nodes that are located topologically farther away than her one-hop neighborhood as these nodes would never receive her attack fragments.

6.3.3.4 Susceptibility in Case of Enhanced Route-over Routing

Enhanced route-over-based forwarding nodes do not reassemble Eve's maliciously crafted 6LoWPAN fragments, but directly forward her attack fragments towards the final destination inside the constrained node network. This enables Eve to mount the identified attacks against the same constrained devices as for mesh-under routing. Hence, the reach of her attack extends beyond constrained devices in her one-hop neighborhood, i.e., the vulnerable devices of the basic route-over routing approach.

6.4 The Content-Chaining Scheme

To protect constrained devices against the 6LoWPAN fragmentation attacks that we identified above, we now present the design of our two complementary, lightweight defense mechanisms, i.e., the *content-chaining scheme* and the *split buffer approach*. In this section, we focus our discussion on the content-chaining scheme and refer to Section 6.5 for the split buffer approach. We start our introduction of the content-chaining scheme with a brief review of why existing security mechanisms often do not suffice to protect constrained devices against the *fragment duplication attack*.

6.4.1 Partial and Non-Solutions

The fragment duplication attack enables Eve to block the successful reassembly of overheard fragmented IPv6 packets by sending a single duplicate 6LoWPAN fragment. Constrained devices could defend against this attack if they were able to identify the 6LoWPAN fragment combination that represents the original IPv6 packet. To this end, a reassembling node could, e.g., verify *upper layer protocol information* such as a message integrity checksum for each reassembled fragment combination.

This simple approach, however, exhibits multiple shortcomings. First, it requires *all* upper layer protocols to employ integrity protection mechanisms that can withstand an existential forgery attack. The goal of this attack would be for Eve to craft overlapping 6LoWPAN fragment content that validates against a given integrity checksum but differs from the original fragment content. Checksums such as the one's complement sum of TCP or UDP, yet, commonly do not provide sufficient cryptographic strength to withstand such a forgery attack [HS13]. Instead, cryptographic integrity protection mechanisms would be required for all upper layer protocols. Such cryptographic mechanisms, however, typically are only used in network security protocols and are not generally available. Second and most importantly, verifying upper layer protocol information requires the recipient of a fragmented IPv6 packet to optimistically store all received 6LoWPAN fragments, including those fragments that were sent by Eve. Hence, by sending multiple duplicate fragments, Eve could overload the reassembly buffer at the fragment recipient. The recipient then could no longer process subsequently arriving 6LoWPAN fragments, thus preventing the reassembly of legitimate 6LoWPAN-fragmented IPv6 packets in the first place.

From the above discussion, we derive that constrained devices must be able to differentiate between legitimate and spoofed 6LoWPAN fragments on a per-fragment basis, i.e., below the network layer. *Link layer security mechanisms* partially realize this requirement as they provide confidentiality protection and data origin authentication for individual link layer frames. With *network-wide keys* as, e.g., employed in the context of ZigBee IP [ZBIP13], these mechanisms, however, only afford to prevent a *network-external* adversary, who is in the radio range of the target network, from injecting packets and from overhearing legitimate communication. Moreover, these mechanisms do *not* enable the fragment recipient to distinguish between different network participants as all devices in the constrained node network share the same link layer protection key. Thus, link layer security based on network-wide keys commonly does not suffice to prevent Eve from mounting the fragment duplication attack as she also is in the possession of the network-wide key (see Section 6.2).

Complementary to such security mechanisms at the data link layer, we, therefore, now introduce the content-chaining scheme as a lightweight defense mechanism against the fragment duplication attack at the 6LoWPAN layer.

6.4.2 High-level Overview of the Content-Chaining Scheme

The content-chaining scheme protects constrained devices against the fragment duplication attack by enabling reassembling nodes and fragment forwarders to cryptographically verify that a received 6LoWPAN fragment indeed belongs to the indicated fragmented IPv6 packet. To achieve this protection in an efficient manner, the design of the content-chaining scheme is based on the following two observations. First, while the 6LoWPAN standard does not define a specific sending order for the 6LoWPAN fragments of a fragmented IPv6 packet, *in-order transmission* starting from the FRAG1 has recently been recognized as highly advisable, especially in constrained node networks that employ route-over-based fragment forwarding [Bor13, ZBIP13]. Second, a fragment recipient does *not* need to be able to authenticate the data origin across *multiple* fragmented IPv6 packets. Instead, it suffices if the recipient can unambiguously identify associated 6LoWPAN fragments

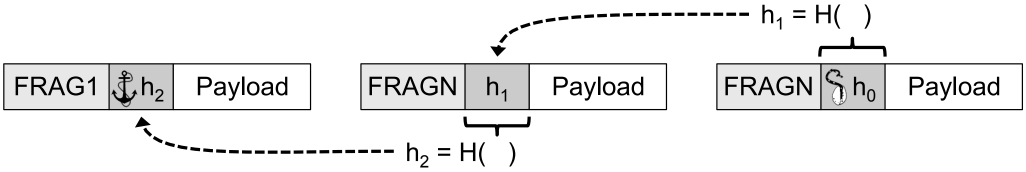


Figure 6.9 In a basic approach, a hash chain element is added to each 6LoWPAN fragment during the 6LoWPAN fragmentation procedure. The hash chain elements, however, are not cryptographically bound to the 6LoWPAN fragment content. In case of out-of-order fragments, Eve can exploit this fact to forge 6LoWPAN fragments that correctly validate at a verifier.

for *each* fragmented IPv6 packet. Even with this weaker security property, Eve can no longer attribute her attack fragments to an overheard fragmented IPv6 packet.

Based on these observations, the basic idea behind the content-chaining scheme is to cryptographically bind the FRAGNs of a 6LoWPAN-fragmented IPv6 packet to the corresponding FRAG1. After reception of the FRAG1, the fragment recipient then is able to unambiguously verify if a received FRAGN belongs to the same overall IPv6 packet as indicated in its 6LoWPAN fragmentation header. This allows the recipient to immediately discard Eve's attack fragments after reception.

To achieve an efficient cryptographic binding between 6LoWPAN fragments without relying on device-specific keys[2], we leverage the concept of hash chains (see Section 3.2.4.1) in our content-chaining scheme. To recollect, the key idea behind a hash chain is for the sender to generate a chain of interlinked one-time authentication tokens that a verifier can validate based on the knowledge of a single element.

We now first discuss why the mere inclusion of a hash chain element in each 6LoWPAN fragment does not suffice to protect against the fragment duplication attack and then describe how the content-chaining scheme averts the shortcomings of this approach.

6.4.3 A Basic Fragment-Chaining Approach

In a basic fragment-chaining approach, the sender of a fragmented IPv6 packet generates a hash chain as described in Section 3.2.4.1 during the 6LoWPAN fragmentation procedure. This hash chain has the same number of elements as the fragmented IPv6 packet has 6LoWPAN fragments. As illustrated in Figure 6.9, the sender then adds these hash chain elements to the 6LoWPAN fragments in reverse order, i.e., from the anchor element in the FRAG1 to the seed value in the last 6LoWPAN fragment. A fragment recipient, in turn, stores the anchor element included in the FRAG1 and subsequently verifies the received FRAGNs by validating the included hash token.

Merely including hash chain elements in the 6LoWPAN fragments, however, does not suffice to protect constrained devices against the fragment duplication attack. This is because FRAGNs may also be received out-of-order on the forwarding path or at the final 6LoWPAN fragment destination. Eve could take advantage of this fact by generating forged FRAGNs that cannot be distinguished from legitimate fragments. To do so, she could replay an overheard hash chain element h_i in a maliciously

---

[2] Device-specific keys would require centralized online coordination [BN07], public-key infrastructures (e.g., based on certificates), or pre-deployment keying material (e.g., based on probabilistic key pre-distribution [EG02, CY07]) for all possible peers inside the constrained node network.

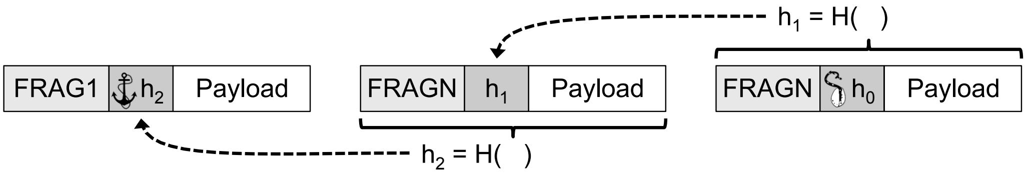


Figure 6.10 A content chain for an IPv6 packet that is fragmented into three 6LoWPAN fragments. The content token in the FRAG1 commits to the overall IPv6 packet content as the iterative construction of the content chain involves the actual fragment content.

crafted FRAGN. In addition, she could also compute a previously unseen hash chain element h_{j+k} ($k > 0$) based on an overheard element h_j of an out-of-order fragment. To this end, Eve would only need to apply the hash function $H(\cdot)$ to h_j for k times: $h_{j+k} = H^k(h_j)$. She could then add the element h_{j+k} to a forged FRAGN.

Due to these abilities of Eve, a fragment recipient still cannot distinguish between legitimate and attack fragments when receiving overlapping FRAGNs. This is because an overlapping FRAGN could either be a legitimate out-of-order fragment, for which Eve previously sent a forged fragment with a computed element h_{j+k}, or it could be Eve's attack fragment that contains a replayed element h_i. As a result of this ambiguity, Eve would still be able to mount the fragment duplication attack.

6.4.4 Construction of a Content Chain

To afford secure fragment verification despite out-of-order FRAGNs, our content-chaining scheme additionally considers the actual *fragment content* in the construction of a content chain as shown in Figure 6.10. More precisely, the sender uses a random value in combination with the fragment content of the last *FRAGN* as the seed value when starting the construction of a content chain during the 6LoWPAN fragmentation procedure. The sender then adds the resulting content token to the previous *FRAGN* and computes the hash digest over the fragment content as depicted in Figure 6.10. After iteratively applying this procedure to the entire IPv6 packet, the FRAG1 contains a content token that transitively commits to the overall IPv6 packet content. This allows a fragment recipient to verify that subsequent FRAGNs belong to the same IPv6 packet after receiving the associated FRAG1.

Importantly, the construction of a content chain prevents Eve from either creating forged FRAGNs with replayed token information or from computing valid content tokens for overheard out-of-order fragments. This is because content tokens cryptographically commit to a specific chain of 6LoWPAN fragment contents. As a result, content tokens require Eve to send attack fragments with content that matches the original fragment content. This, however, does not lead to an attack as equal content does *not* require the recipient to decide for one of the overlapping fragments.

6.4.5 Verifying the Tokens of a Content Chain

A device that receives all 6LoWPAN fragments of a protected fragmented IPv6 packet also possesses the necessary information to verify the content tokens that are contained in the received 6LoWPAN fragments. This is because content chains are self-contained on a per-packet basis. Specifically, a fragment-verifying node neither requires knowledge about previous fragmented IPv6 packets to validate the content tokens of a recent fragmented IPv6 packet nor does it need out-of-band information about device-specific keys for verification purposes. This enables the final 6LoWPAN fragment destination as well as the fragment forwarders in case of (enhanced) route-over routing (see Section 6.3.3) to verify received 6LoWPAN fragments.

To validate the content tokens of protected 6LoWPAN fragments, a fragment recipient processes a received FRAG1 normally and stores the included content token for verification purposes. When subsequently receiving a FRAGN that appears to belong to the same overall IPv6 packet, the recipient computes the hash over the received fragment content, i.e., including the contained content token, and matches the resulting hash digest to its stored token. If both values match, the recipient accepts the 6LoWPAN fragment and processes its fragment content at the 6LoWPAN layer. Moreover, the recipient replaces the stored content token with the verified token. This final step affords a verification of the next FRAGN with only a single hash operation. Contrary, if the computed hash digest differs from the stored token, the recipient considers the received FRAGN spoofed and drops it immediately.

6.4.5.1 Verification of Out-of-Order Fragments and Fragment Loss

When receiving an out-of-order FRAGN, a fragment-verifying node may not be able to directly verify the 6LoWPAN fragment as the preceding fragment and, therefore, the commitment to the content of the just received FRAGN might still be missing. To minimize non-malicious reception of such out-of-order FRAGNs, we require fragment forwarders in case of (enhanced) route-over routing to only forward 6LoWPAN fragments that have been verified successfully. Moreover, forwarding nodes as well as the final 6LoWPAN fragment destination store out-of-order FRAGNs without prior verification until all previous FRAGNs have been verified successfully.

We note that, while these policies afford verification of out-of-order fragments, they also enable Eve to overload the reassembly buffer of a fragment-verifying node with FRAGNs that appear to be out-of-order fragments of a legitimate IPv6 packet. This attack is limited to Eve's one-hop neighborhood in case of (enhanced) route-over routing and targets the final 6LoWPAN fragment destination for mesh-under routing. To frustrate Eve's attack, fragment-verifying nodes discard the fragment with the highest datagram offset when reaching a buffer overload situation for a fragmented IPv6 packet. This fragment least likely is the result of non-malicious fragment reordering in case of (enhanced) route-over routing as forwarding devices only relay verified 6LoWPAN fragments. However, for mesh-under-based networks, there is a chance that the discarded out-of-order fragment actually was legitimate as individual fragments may be routed along different forwarding paths with varying propagation delays. Hence, in these networks, the 6LoWPAN reassembly buffer size at a reassembling node should allow to account for a limited number of Eve's attack fragments before the corresponding legitimate 6LoWPAN fragment arrives.

Besides out-of-order FRAGNs, 6LoWPAN fragments may also be lost on the forwarding path. According to the 6LoWPAN standard [MKHC07], such fragment loss causes the entire fragmented IPv6 packet to be discarded after the reassembly timeout expired. Consequently, the content-chaining scheme is not required to be robust against token loss. Instead, if reliable communication is provided by an upper layer protocol, the loss of a 6LoWPAN fragment triggers a retransmission at this protocol layer. This retransmission then is fragmented at the 6LoWPAN layer similar to the original IPv6 packet and, thus, also is protected with the content-chaining scheme.

6.4.6 Overhead Reduction Techniques

The content-chaining scheme trades increased computation, memory, and transmission overheads for a gain in protection against the fragment duplication attack. This trade-off also persists when no attacks are imminent. To remedy this circumstance, we introduce an *attack notification extension* in Section 6.4.6.1 that affords a default-off policy for the content-chaining scheme. Moreover, we show how *truncated content tokens* (Section 6.4.6.2) and *AES hardware support* (Section 6.4.6.3) allow to reduce the computation, memory, and transmission overheads in case the attack notification extension triggers the activation of the content-chaining scheme. For a detailed security discussion of these overhead reduction techniques, we refer to Section 6.6.2.

6.4.6.1 Attack Notification Extension

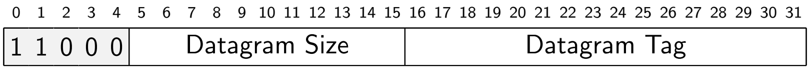
The attack notification extension supplements the content-chaining scheme with a simple attack detection mechanism for the fragment duplication attack. Moreover, it enables the recipient of overlapping 6LoWPAN fragments to request the legitimate fragment sender to activate the content-chaining scheme. Due to this reactive approach, the content-chaining scheme then incurs zero run-time overheads during normal operation and effectively protects constrained devices in case of an attack.

The attack notification extension relies on constrained devices that also reassemble fragmented IPv6 packets without the content-chaining scheme in order to detect the fragment duplication attack. These devices compare the content in their reassembly buffer to the received 6LoWPAN fragment[3]. In case of overlapping but matching fragment content, the reassembling device drops the newly received 6LoWPAN fragment and proceeds without further actions. However, if the overlapping content differs, this device sends an ICMP notification message to the original fragment sender in order to alert about the impending attack. This ICMP message should employ a yet unused ICMP type number, e.g., 42, that specifically indicates the reception of overlapping fragment content. In addition to sending this ICMP notification message, the reassembling device also discards the corresponding IPv6 packet from its reassembly buffer as it cannot determine the legitimate fragment combination. As soon as the sender of the fragmented IPv6 packet receives such an ICMP notification message, it activates the content-chaining scheme for the transmission of subsequently fragmented IPv6 packets. As a result, the adversary is *unable* to mount the fragment duplication attack against future fragment transmissions.

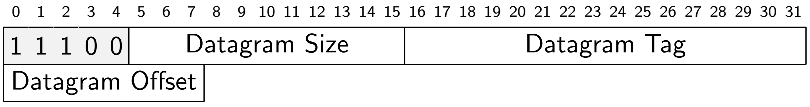
To prevent the content-chaining scheme from remaining active indefinitely due to a single fragment duplication attack, we allow the sender to deactivate this defense

---

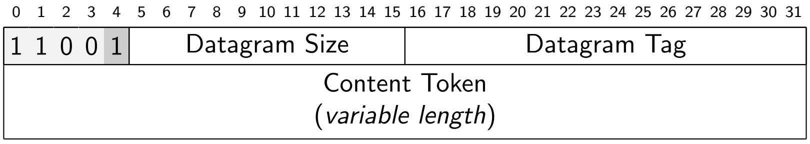
[3] This operation is based on a memory comparison and, thus, only incurs a marginal overhead.



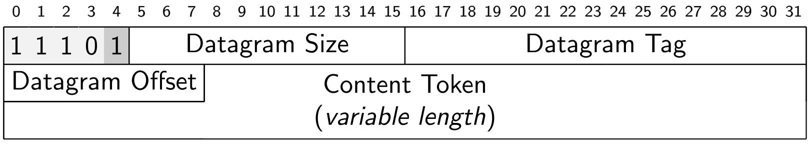
(a) Original FRAG1 header



(b) Original FRAGN header



(c) FRAG1 header with our content-chaining scheme



(d) FRAGN header with our content-chaining scheme

Figure 6.11 6LoWPAN fragmentation headers with and without our content-chaining scheme. The 6LoWPAN header type is indicated by the header fields marked in gray. The difference of the new header types for the protected 6LoWPAN fragments is highlighted in dark gray.

mechanism after a pre-configured timeout of, e.g., 15 minutes[4]. Notably, this deactivation allows the adversary to continue the fragment duplication attack as soon as the deactivation timeout has expired. Such attack continuation by the adversary, however, would cause the reassembling device to trigger another ICMP notification message and the content-chaining scheme would be re-activated. As a result, the adversary again would be unable to mount the fragment duplication attack.

Importantly, the final 6LoWPAN fragment destination and fragment forwarding nodes have to be able to distinguish protected from unprotected 6LoWPAN fragments for interoperability purposes. Protected 6LoWPAN fragments, therefore, use dedicated 6LoWPAN fragmentation headers that are based on the existing FRAG1 and FRAGN headers. As illustrated in Figure 6.11, the only difference of these new fragmentation headers are the header type values and the inclusion of a header field for the content token information. Fragment-verifying nodes then only verify content tokens for those 6LoWPAN fragments with a fragment header type that indicates the use of the content-chaining scheme and otherwise process the fragment normally.

6.4.6.2 Truncated Content Tokens

When the content-chaining scheme protects 6LoWPAN fragments in its *active state*, it causes an increased transmission overhead compared to the standard 6LoWPAN fragmentation mechanism. This overhead stems from the content token information that is included in each protected 6LoWPAN fragment. To decrease the transmission overhead in its active state, we allow the size of the content tokens to be configurable

---

[4] The length of the deactivation timeout is a trade-off between a timespan with overhead resulting from an active content-chaining scheme but no adversary present and packet loss due to a continuing attack with a deactivated content-chaining scheme.

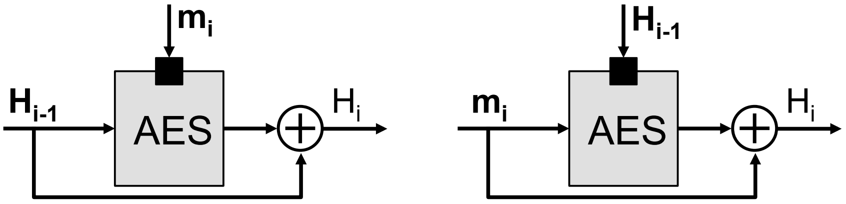
to a smaller size than the native output length of the underlying cryptographic primitive, which, e.g., amounts to 20 byte in case of SHA-1 [NIST12b]. Specifically, the content-chaining scheme can *truncate* the hash digest as an alternative to adopting it in its unmodified form in order to allow for smaller tokens of either 8 or 16 byte.

To afford the verification of variable-size content tokens, a fragment-verifying node must be able to determine the length of the token information that is included in a received 6LoWPAN fragment. To this end, the content-chaining scheme could, e.g., add a new token size indication field to 6LoWPAN fragmentation header. We, however, refrain from such explicit size indication in order to forgo the corresponding per-fragment transmission overhead. Instead, we require the content token size to be configured consistently on a network-wide basis depending on the security requirements of the specific constrained node network. Such network-wide configuration can, e.g., be realized during the bootstrapping phase of a constrained device (see Section 3.3.2) or at run-time via the extended context distribution facilities of the IPv6 Neighbor Discovery mechanism for constrained devices (see Section 2.3.2).

6.4.6.3 Leveraging AES Hardware Support

Besides the transmission overheads stemming from the content tokens, the content-chaining scheme also incurs computation, RAM, and ROM overheads. These overheads strongly depend on the underlying cryptographic primitive. Hence, if the cryptographic primitive was supported in hardware, this would allow to considerably reduce the computation overhead for the general-purpose MCU as well as the memory requirements for the implementation of the employed primitive. With these considerations in mind, we observe that most constrained devices with IEEE 802.15.4 radio modules are equipped with hardware support for the AES block cipher. Moreover, we note that this hardware support often is exposed to the operating system via well-defined hardware-level Application Programming Interfaces (APIs) [cc2420, cc2520].

To leverage this AES hardware support in the design of the content-chaining scheme, we allow content tokens to be computed via an AES-based hash function instead of a standard hash function such as SHA-1. To this end, we employ a length-padded Merkle-Damgård construction [Mer90, Dam90] as presented in Section 3.2.4. The central cryptographic component of this construction is the iterated application of a block-cipher-based one-way compression function. However, as the AES encryption and decryption functions are bijective, these functions do not natively provide the necessary one-way compression property. Still, by employing feedback-based modes of operation such as Davies-Meyer (DM) [BÖS11] or Matyas-Meyer-



(a) Davies-Meyer (DM) **(b)** Matyas-Meyer-Oseas (MMO)

Figure 6.12 One-way compression functions. The DM and MMO modes of operation afford the use of the AES block cipher to build a hash function via the Merkle-Damgård construction.

Oseas (MMO) [CZ12] (see Figure 6.12), the required one-way compression function can effectively be built from the AES block cipher. Hence, we leverage these modes of operation for the AES-based hash functions in the content-chaining scheme. Moreover, we allow for the network-wide configuration of the cryptographic primitive based on the same configuration mechanisms discussed in the previous section.

6.5 The Split Buffer Approach

In the previous sections, we focused our discussion on the content-chaining scheme. Complementary to this defense mechanism against the fragment duplication attack, we now present the split buffer approach that affords protection against the buffer reservation attack. As before, we start our discussion with a brief review of why existing security mechanisms commonly do not suffice to prevent this type of attack.

6.5.1 Partial and Non-Solutions

The buffer reservation attack enables an adversary to proactively reserve the scarce reassembly buffer resources at a target device by sending well-timed 6LoWPAN fragments that never reassemble to a complete IPv6 packet. As a result, existing security mechanisms that are located *above* the 6LoWPAN layer never get to inspect these attack fragments and, thus, do not allow to mitigate the buffer reservation attack. Moreover, link layer security mechanisms as, e.g., employed in ZigBee IP [ZBIP13], do not enable the fragment recipient to distinguish between the different network participants inside a constrained node network. This enables Eve to spoof her attack fragments in order to prevent detection by the target device when mounting the buffer reservation attack. To protect constrained devices against the buffer reservation attack, we now introduce our split buffer approach at the 6LoWPAN layer.

6.5.2 High-level Overview of the Split Buffer Approach

The basic idea behind the split buffer approach is to frustrate the buffer reservation attack by significantly increasing the effort that an adversary has to put into mounting the attack. More precisely, we aim at forcing the adversary into sending *complete* 6LoWPAN-fragmented IPv6 packets *in short fragment bursts* in order to reserve a large amount of reassembly buffer resources. As a result of such sending behavior, the adversary then is no longer able to exploit the amplification effects of a malicious buffer reservation, thus rendering the buffer reservation attack unattractive.

To increase the effort that the adversary has to put into mounting the buffer reservation attack, the split buffer approach consists of the following two mechanisms. First, the split buffer approach partitions the reassembly buffer into *fragment-sized buffer slots.* By assigning these buffer slots on a per-fragment instead of a per-packet basis, the split buffer approach prevents an adversary from reserving the entire reassembly buffer with a single 6LoWPAN fragment. An adversary, however, may still occupy all buffer slots with a multitude of maliciously crafted 6LoWPAN fragments. To mitigate such malicious buffer occupation, we extend the split buffer

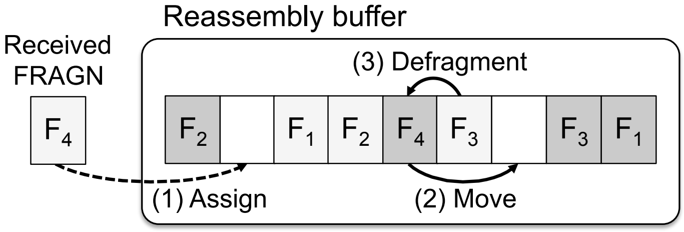


Figure 6.13 The split buffer of a reassembling node. When receiving a new 6LoWPAN fragment, the reassembling node assigns this fragment to one of its unoccupied buffer slots (1). Once all 6LoWPAN fragments for a fragmented IPv6 packet have been received, the reassembling node rearranges the corresponding fragments according to their indicated fragment offsets (2 and 3). As the 6LoWPAN header is removed during this defragmentation procedure, the defragmented part of the reassembly buffer finally contains the reassembled IPv6 packet.

approach with a *packet discard strategy* that severely penalizes suspicious sending behavior. We now proceed with a detailed description of the fragment-size buffer slots in Section 6.5.3 and then present the packet discard strategy in Section 6.5.4.

6.5.3 Fragment-sized Buffer Slots

For the design of the split buffer approach, we observe that a single reassembly buffer forces a reassembling node to make a decision whether to replace a partially received IPv6 packet with a newly received packet as soon as the first 6LoWPAN fragment arrives. This prevents the reassembling node from optimistically storing 6LoWPAN fragments that originate from *multiple* senders and to defer the decision which fragmented IPv6 packet to discard to a point when the concurrently received IPv6 packets *actually* cause a buffer overload situation. As a result, the adversary can mount the buffer reservation attack by merely *pretending* to send a complete fragmented IPv6 packet. Moreover, even multiple reassembly buffers would not mitigate the buffer reservation attack as the adversary could still reserve all available buffer resources by sending multiple incomplete IPv6 packets that each indicates a large overall size in the 6LoWPAN fragmentation header (see Section 2.4.2).

If a reassembling node instead was to store *individual* 6LoWPAN fragments of *multiple* IPv6 packets in its reassembly buffer, Eve's attack fragments would compete with legitimate fragments for the available buffer resources on a *per-fragment* basis. Such per-fragment competition would be based on the buffer space that the received 6LoWPAN fragments actually occupy. Hence, an adversary would have to follow up on her pretense to maliciously occupy the buffer resources at the reassembling node.

To enable such per-fragment competition, the split buffer approach partitions the available memory resources for the reassembly buffer into *fragment-sized buffer slots* as illustrated in Figure 6.13. These buffer slots have the maximum size of a 6LoWPAN fragment for a given constrained node network. A reassembling node then no longer has to reserve the entire reassembly buffer when receiving a 6LoWPAN fragment that belongs to a new fragmented IPv6 packet. Instead, the node can assign the received 6LoWPAN fragment to one of its unoccupied buffer slots (see step 1 in Figure 6.13). Once all 6LoWPAN fragments of a fragmented packet have been received, the reassembling node rearranges the associated fragments according to their indicated

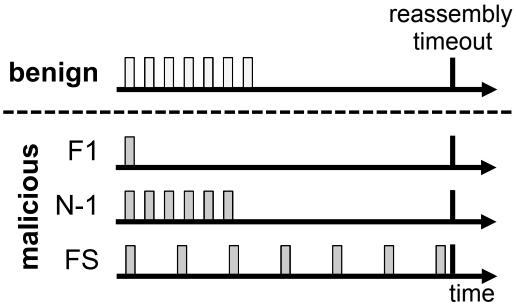


Figure 6.14 Legitimate and malicious sending behaviors for 6LoWPAN-fragmented IPv6 packets. While legitimate senders transmits 6LoWPAN fragments in short succession, an adversary may exclusively send FRAG1s (F1), transmit fragment bursts excluding the last FRAGN (N-1), or spread fragment transmission across the reassembly timeout at the target device (FS).

fragment offsets (see steps 2 and 3 in Figure 6.13). During this *defragmentation procedure*, the reassembling node also removes all 6LoWPAN header information from the defragmented fragments. Thus, upon completion, the defragmented part of the reassembly buffer contains the reassembled IPv6 packet. The node then passes this reassembled packet up to the IPv6 layer, where it is processed normally.

By storing 6LoWPAN fragments from multiple IPv6 packets in an interleaved fashion, the split buffer approach may potentially run into an overload situation. This overload situation is reached, when the last available buffer slot is assigned to a newly received 6LoWPAN fragment, but no IPv6 packet in the split buffer is completed as a result. In this case, the reassembling node has to decide which partially received IPv6 packet it wants to discard in order to free the corresponding buffer slots for the reassembly of the remaining partially received IPv6 packets in the split buffer.

To derive this decision in an informed manner, the packet discard strategy of the split buffer approach additionally considers the *observed sending behavior* of the already received 6LoWPAN fragments in the decision making process. This allows to prioritize the subsequent collection of those partial IPv6 packets with unsuspicious sending behavior. We now continue with a description of how the packet discard strategy leverages the observed sending behavior in order to dispose of a packet.

6.5.4 Packet Discard Strategy

For our packet discard strategy, we observe that legitimate fragment senders commonly send the 6LoWPAN fragments of a fragmented IPv6 packet in short succession as illustrated in Figure 6.14. This is because extended buffering of 6LoWPAN fragments would unnecessarily block the scarce sending buffer resources of a legitimate fragment sender concerning subsequent transmissions of upper layer payload data. As shown in Figure 6.14, an adversary, contrarily, may employ one of the following three fundamentally different attack behaviors when mounting the buffer reservation attack[5]: i) exclusive transmission of FRAG1s (F1) as initially discussed in Section 6.3.2, ii) transmission of all but the last 6LoWPAN fragments in short succession (N-1), and iii) fragment spreading across the reassembly timeout (FS).

---

[5] Further attack behaviors then denote variations or combinations of these sending behaviors.

With respect to these attack behaviors, we note that the exclusive transmission of FRAG1s (F1) requires the smallest resource commitment from the adversary. The long transmission gap following the FRAG1, however, indicates a potential attack or the loss of the remaining 6LoWPAN fragments in case of benign sending behavior. Thus, a reassembling node should preferably retain the other partially received IPv6 packets in its split buffer when a buffer overload situation occurs. In contrast, fragment bursts (N-1) are less suspicious than only sending a single FRAG1 as this sending behavior closely resembles benign behavior. However, the long gap after the first 6LoWPAN fragment burst still indicates an impending attack or the loss of the last remaining FRAGN in case of a legitimate fragment sender. Thus, a reassembling node again should prioritize the reassembly of other fragmented IPv6 packets in case of a buffer overload situation. Fragment spreading, in turn, appears most legitimate compared to the other two attack behaviors as the intermediate delay between the individual 6LoWPAN fragments may also stem, e.g., from extensive duty cycle periods at the previous hop on the forwarding path. By spreading the fragment transmission across the reassembly timeout, the adversary, however, is only able to fill the buffer reassembly resources of the target node at a slow pace.

The packet discard strategy leverages the above observations by computing *per-packet scores* that capture the *percentage of completion* as well as the *continuity in the sending behavior* for each partial IPv6 packets in the split buffer. In case of a buffer overload situation, the packet discard strategy then disposes of the partial IPv6 packet with the lowest packet score. We now proceed with a detailed discussion of how the packet discard strategy computes these per-packet scores. We thereby first focus on the percentage of completion in Section 6.5.4.1 and then describe how the packet discard strategy accounts for the sending behavior in Section 6.5.4.2.

6.5.4.1 Considering the Percentage of Completion

To prioritize the partial IPv6 packets that are most likely to complete promptly after a buffer overload situation has been resolved, the packet discard strategy scores each IPv6 packet in the split buffer based on its percentage of completion. To this end, it updates the packet score of a partially received IPv6 packet based on the running sum computation depicted in Algorithm 1 when receiving the ith 6LoWPAN fragment.

```
procedure ComputePacketScore(fragment)
    packet = packet indicated by fragment
    if packet not in split buffer then
        $\text{score}_{\text{packet}} = \frac{\text{fragment size}}{\text{packet size}}$
    else
        $\text{score}_{\text{packet}} = \text{score}_{\text{packet}} + \frac{\text{fragment size}}{\text{packet size}}$
    end if
end procedure
```

Algorithm 1 Computation of the packet score upon 6LoWPAN fragment arrival at a reassembling node based on the percentage of completion.

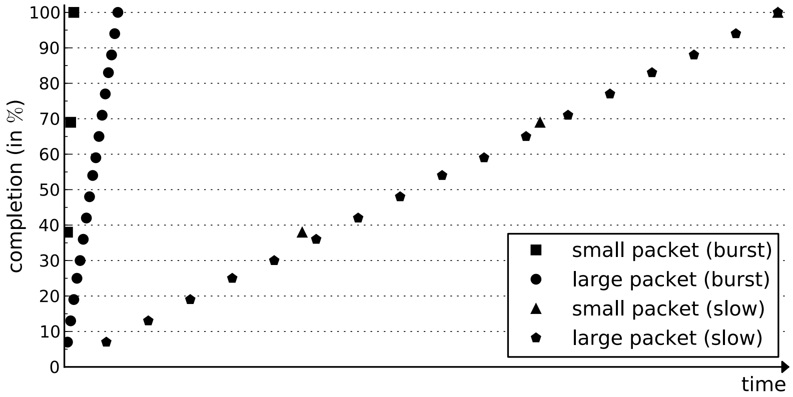


Figure 6.15 Packet score at a reassembling node based on the percentage of completion. Fragmented packets, which are sent in short bursts, quickly gain a high packet score. In contrast, fragmented packets, which are transmitted at a low sending rate and reserve the reassembly resources for extended periods of time, only slowly increase the packet score.

As illustrated in Figure 6.15, Algorithm 1 leads to a quick increase of the packet score for fragmented IPv6 packets that are transmitted in short 6LoWPAN fragment bursts independent from the overall packet size. As discussed in Section 6.5.4, these packets also are most probably the result of benign sending behavior. In contrast, fragmented IPv6 packets that are transmitted at a low sending rate only increase the score at a slow pace. Hence, in buffer overload situations, IPv6 packets that are sent in short bursts are likely to have a higher score than slowly arriving attack packets. This circumstance renders low sending rates unattractive for the adversary.

6.5.4.2 Considering the Sending Behavior

An adversary may also change the fragment sending behavior after achieving a high packet score for the occupied buffer slots at the target device, e.g., by first sending a fragment burst and then spreading the remaining fragments across the reassembly timeout. To penalize such change in behavior, we additionally incorporate the time domain in the packet discard strategy. The basic idea of this extension is to decrease the packet score according to the intensity of the change in sending behavior. Hence, minor fluctuations that may be attributable to common network effects are penalized less than harmful or malicious sending behavior. To capture the change in fragment sending behavior, we additionally consider the *average time elapsed between two consecutive 6LoWPAN fragments* (a) of a fragmented IPv6 packet and the *time elapsed since the reception of the last 6LoWPAN fragment* (l) of this packet. More precisely, if the currently elapsed time l differs significantly from the expected time

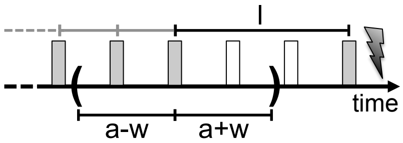


Figure 6.16 The packet discard strategy of the split buffer approach penalizes fragmented packets with a significant change in sending behavior. To this end, the discard strategy checks if the time that elapsed since the reception of the last 6LoWPAN fragment (l) exceeds the average time between the preceding 6LoWPAN fragments (a) by more than a grace period w.

```
procedure ComputePacketScore(fragment, $w$)
    packet = packet indicated by fragment
    if packet not in split buffer then
        $\text{score}_{\text{packet}} = \frac{\text{fragment size}}{\text{packet size}}$
    else
        $a$ = average time between received fragments of packet
        $l$ = time since last fragment reception of packet
        if $a - w < l < a + w$ then
            $\text{score}_{\text{packet}} = \text{score}_{\text{packet}} + \frac{\text{fragment size}}{\text{packet size}}$
        else
            $\text{score}_{\text{packet}} = \frac{\text{score}_{\text{packet}}}{2^{\max\{1;\lfloor l/a \rfloor\}}}$
        end if
    end if
    return $\text{score}_{\text{packet}}$
end procedure
```

Algorithm 2 Adapted computation of the packet score upon fragment arrival at a reassembling node with penalty of significant changes in the sending behavior.

of fragment reception a, the packet discard strategy considerably reduces the score of the corresponding partially received IPv6 packet in the split buffer.

To only penalize significant changes in the sending behavior and, thus, to allow for small variations regarding the fragment reception times, we introduce a time window w in the packet discard strategy as shown in Figure 6.16. This time window allows to calibrate the maximum tolerable change in sending behavior to the specific network characteristics. Hence, as long as the sending behavior does not change considerably ($a - w < l < a + w$), the score is calculated based on the percentage of completion. However, if the reassembling node receives a 6LoWPAN fragment earlier than expected ($l \leq a - w$), it decreases the packet score by half. Likewise, if the 6LoWPAN fragment arrives later than expected ($l \geq a + w$), the score is halved for each presumably intermittent 6LoWPAN fragment ($\lfloor l/a \rfloor$). To incorporate these penalties in the packet score, we adapt its computation as illustrated in Algorithm 2. We now describe how the split buffer approach incorporates these computed packet scores of the packet discard strategy in case of a buffer overload situation.

6.5.4.3 Integration with the Split Buffer Approach

The packet discard strategy only triggers the calculation of the packet score for the IPv6 packet in the split buffer that corresponds to a just received 6LoWPAN fragment. This is to achieve a low per-fragment computation overhead. Hence, when a discard decision should be derived, the packet scores of the remaining packets in the split buffer must be updated to reflect the score at the time when the decision takes place. The comparison, thus, is either based on the previously computed or the reduced packet score depending on the elapsed time since the reception of the last fragment for each partially received IPv6 packet in the split buffer (see Algorithm 3).

```
procedure ComputeComparisonScore(packet, $w$)
    $a$ = average time between received fragments of packet
    $l$ = time since last fragment reception of packet
    if $a - w < l < a + w$ then
        $\text{score}_{\text{comparison}} = \text{score}_{\text{packet}}$
    else
        $\text{score}_{\text{comparison}} = \frac{\text{score}_{\text{packet}}}{2^{\max\{1; \lfloor l/a \rfloor\}}}$
    end if
    return $\text{score}_{\text{comparison}}$
end procedure
```

Algorithm 3 Computation of the comparison score for each partially received IPv6 packet in the split buffer. The packet with the lowest score will be discarded.

After computing the score for each partial IPv6 packet in the split buffer according to Algorithm 3, the packet discard strategy disposes of all 6LoWPAN fragments that belong to the IPv6 packet with the lowest packet score. If multiple packets have the same low score, the packet discard strategy randomly flushes one of these packets. This is to ensure that an adversary cannot exploit specific design traits of the packet discard strategy to craft incomplete packets with a low score that are always preferred over others. Finally, the freed buffer slots enable the reassembling node to continue with the reassembly of the remaining packets in the split buffer.

To summarize, the split buffer approach prioritizes competing fragmented IPv6 packets that are transmitted in short fragment bursts. Moreover, its packet discard strategy severely penalizes significant changes in the fragment sending behavior. Combined, these properties considerably limit the practicability of the buffer reservation attack and effectively force an adversary into flooding the target device with short bursts of fragmented IPv6 packets. Hence, the split buffer approach successfully prevents the adversary from exploiting the amplification effects that stem from the reservation of the 6LoWPAN reassembly buffer and requires the adversary to have sufficient resources to mount a brute-force flooding attack in order to incapacitate the target device. At the same time, the split buffer approach also allows for a more efficient use of the reassembly buffer resources for *legitimate communication* compared to a single or multiple reassembly buffers of the same size. This is because the reassembly buffer resources are assigned on a per-fragment basis. As a result, a reassembling node can even process interleaved IPv6 packets that, combined, would exceed the reassembly buffer resources. We note that such interleaved processing is highly desirable as congestion of the reassembly buffer has been identified as a reoccurring problem in the context of constrained devices [LCC11, PS13, WRTTG13].

6.6 Security Considerations

We now discuss potential attacks that the adversaries Eve or Mallory (see Section 6.2) can mount against the presented defense mechanisms. To structure this discussion, we differentiate between attacks against (i) the basic content-chaining scheme, (ii) its overhead reduction techniques, and (iii) the split buffer approach.

To recollect, Mallory can at least mount the attacks that are viable for Eve. Hence, we do not mention her explicitly when discussing Eve in the following discussion.

6.6.1 Attacks Against the Basic Content-Chaining Scheme

The content-chaining scheme cryptographically binds the overall content of a fragmented IPv6 packet to the corresponding FRAG1 at the 6LoWPAN layer. In this section, we outline attacks that can be mounted despite such cryptographic binding.

6.6.1.1 Spoofed FRAG1 Fragments

Eve may overhear a legitimate FRAG1 with a valid content token and send spoofed FRAG1s that include the overheard token. Despite employing our content-chaining scheme, a fragment recipient could not distinguish between the legitimate and Eve's FRAG1s as content tokens only assure the legitimacy of the subsequent FRAGNs. As a result, the fragment recipient would need to drop the entire IPv6 packet due to indistinguishable overlapping FRAG1 fragments. We note that this property denotes a trade-off of the content-chaining scheme that stems from our design decision to forgo the considerable overhead of cryptographically binding the anchor element of a content chain to an on-path verifiable device identity, e.g., via public-key signatures.

To still enforce a decision for a single FRAG1 in case of duplicate FRAG1 fragments, a fragment recipient only considers the first received FRAG1 to be valid. Subsequently received FRAG1s with matching packet attribution but diverging fragment content are dropped by the recipient immediately. This policy prevents Eve from attacking a fragment recipient with spoofed FRAG1s in all network scenarios, where the direct transmission of a FRAG1 between legitimate nodes is faster than the transmission via the off-path adversary Eve. In networks that employ mesh-under routing, Eve may, therefore, still be able to attack the final destination in the constrained node network if she knows a faster forwarding path towards this target device. In contrast, for (enhanced) route-over-based networks, Eve must be in direct communication range of a next hop on the forwarding path and transmit her FRAG1s prior to the legitimate fragment forwarder. Especially for enhanced route-over routing, this significantly limits Eve's capability to mount the fragment duplication attack as forwarding nodes immediately relay FRAG1s after reception.

6.6.1.2 Spoofed Out-Of-Order Fragments

Eve may also try to take advantage of the fact that out-of-order fragments cannot be validated immediately at a fragment recipient. To this end, Eve would send spoofed 6LoWPAN fragments with a large fragment offset for an overheard legitimate 6LoWPAN fragment. Such attack fragments would occupy the scarce buffer resources at the fragment recipient until all previous fragments have been received and verified successfully. Only then could the fragment recipient discard Eve's attack fragments as the content tokens included in her spoofed fragments would be identified as invalid. Still, it is important to note that the fragment discard policy of the content-chaining scheme (see Section 6.4.5.1) causes the fragments with the highest offset to be dropped in case of a buffer overload situation. This frustrates Eve's attack as her attack fragments would be disposed of first. Alternatively, Eve may also send 6LoWPAN fragments with a fragment offset that is close to the current offset of the targeted IPv6 packet. Such attack fragments, however, are quickly discarded upon reception of the corresponding legitimate 6LoWPAN fragments.

6.6.1.3 On-Path Packet Modification

An on-path adversary Mallory may intercept all 6LoWPAN fragments of a traversing IPv6 packet. This would allow her to modify the original packet content and to compute a valid content chain before forwarding the altered IPv6 packet towards its final destination inside the constrained node network. Notably, the content token in the FRAG1 would still only commit to a single 6LoWPAN fragment combination. Hence, Mallory's attack would not result in indistinguishable overlapping 6LoWPAN fragments at the target device. Instead, her attack resembles dropping of the original and creating a new IPv6 packet. These, however, are inherent capabilities of an on-path adversary. Moreover, we note that we consider the integrity protection and the detection of modified IPv6 packet content to be the task of upper layer protocols.

6.6.2 Attacks Targeting the Overhead Reduction Techniques

The attack notification extension allows to activate the content-chaining scheme by sending feedback to the fragment sender. Moreover, truncation and the use of an AES-based hash function afford further overhead reductions for content tokens. We now discuss potential attacks against these overhead reduction techniques.

6.6.2.1 Malicious Attack Notification Messages

The ICMP messages used to notify the sender of a fragmented IPv6 packet about an impending fragment duplication attack are not authenticated. Thus, Eve could force the devices inside a constrained node network to activate the content-chaining scheme by maliciously sending attack notification messages. The resulting computation and transmission overheads inside the constrained node network, however, would be equal to the overheads of the content-chaining scheme without the attack notification extension. Hence, this extension does not expose a new vector of attack compared to the content-chaining scheme without this overhead reduction technique.

6.6.2.2 Dropping of Attack Notification Messages

An on-path adversary Mallory may aim at preventing the content-chaining scheme from being activated in order to successfully mount the fragment duplication attack. To this end, Mallory would merely need to drop traversing attack notification messages. However, if Mallory's goal was to block a fragmented IPv6 packet, she could achieve the same results more easily by simply dropping the traversing 6LoWPAN fragments in the first place. Consequently, the attack notification extension also does not open a new vector of attack with respect to an on-path adversary.

6.6.2.3 Cryptographic Strength of AES-Based, Truncated Content Tokens

The truncation of the tokens in the content-chaining scheme allows to reduce the token size to 8 or 16 byte. Such truncation, however, also reduces the cryptographic strength of the employed content tokens. Still, it is important to note that even truncated content tokens achieve a comparable level of security to, e.g., the cryptographic message integrity codes specified for IEEE 802.15.4 as these codes have a configurable length of 4, 8, or 16 byte [IEEE11b]. Moreover, also when considering current advances concerning the computation of a second pre-image [KS05], i.e., valid fragment content for a token commitment, the level of security only diminishes modestly from 2^{64} to about 2^{63} for content tokens of 8 byte. This relatively low loss in cryptographic strength primarily stems from the small input space of the content tokens, i.e., fragment content of up to 72 byte and the previous token in the content chain. Importantly, the remaining cryptographic strength currently still prevents an adversary from computing a valid content token on-the-fly as the maximum timespan for forging tokens is limited to the length of the 6LoWPAN reassembly timeout.

Similarly, the pre-computation of a look-up table for the short-lived content tokens is hard to achieve for an adversary, even in case of a token length of 8 byte. This is because our content chain construction depends on the previous content token and the actual fragment content. As a result, the look-up structure would need to account for a significant amount of information. Moreover, even if a sender periodically transmitted fragmented IPv6 packets with equal packet payload, an adversary would not gain an advantage from overhearing valid content tokens. This is because the seed value of a content chain is chosen randomly by the sender for each fragmented IPv6 packet. Content tokens, therefore, differ across fragmented IPv6 packets with equal packet payload and cannot be exploited by an adversary.

Finally, Bradford et al. [BG06] recently showed that hash chains with small domains, i.e., a small input space, are prone to contain cycles. Such cycles would enable an adversary to shorten or extend the fragmented packet with the fragment content of a cycle. The overall length of an IPv6 packet, however, is indicated in each 6LoWPAN fragment and, thus, is fixed after reception of the first fragment at a verifying node.

6.6.3 Attacks Against the Split-Buffer Approach

The split buffer approach segments the 6LoWPAN reassembly buffer into fragment-sized buffer slots to foster competition for the scarce buffer resources between an adversary and legitimate fragment senders. Moreover, the packet discard strategy of the split buffer approach employs heuristics to detect malicious sending behavior. In this section, we outline attacks that could be mounted against these mechanisms.

6.6.3.1 Unfair competition for the scarce reassembly buffer resources

Eve may try to establish an advantage in her competition with legitimate fragment senders for the available buffer slots of the split buffer approach by maintaining an initial non-zero packet score at the target device. To this end, she may send 6LoWPAN fragments that belong to a large IPv6 packet at a low transmission rate.

As soon as Eve detects that another device sends a fragmented IPv6 packet, she could change her sending behavior to a high transmission rate for the remaining fragments in order to block the competing fragmented IPv6 packet from being reassembled at the target device. Our packet discard strategy accounts for such behavior by penalizing senders that suddenly change their transmission rate. As the packet score is halved for each fragment that is received early, Eve's score would quickly decrease by a significant amount. This enables newly received fragmented IPv6 packets to compete with Eve's attack fragments despite the lack of an initial score.

6.6.3.2 Imposing a low packet score

An on-path adversary Mallory may forward legitimate fragmented IPv6 packets with varying artificial delays between the associated 6LoWPAN fragments in order to impose a low packet score at a reassembling node. As a result, Mallory's forwarded packets would likely be dropped by the reassembling node in case of a buffer overload situation. Mallory, however, could achieve the same result more effectively by immediately dropping received 6LoWPAN fragments instead of forwarding them towards the final destination. Consequently, adding artificial delays on the communication path does not constitute a new vector of attack with respect to an on-path adversary.

6.7 Evaluation

For our evaluation, we extended the 6LoWPAN layer of the Contiki OS [DGV04] in version 2.5 with the required functionality for the content-chaining scheme and for the split buffer approach. We then evaluated the presented defense mechanisms on Tmote Sky motes[6] as the underlying hardware platform for constrained devices. Hence, constrained devices in our evaluation setup were equipped with an 8 MHz MSP430 MCU, 10 kB of RAM, 48 kB of ROM, and an IEEE 802.15.4 radio interface. With respect to Contiki OS, we used the standard configuration for the Tmote Sky platform where possible and only decreased the size of the RPL neighborhood table from 20 to 6 entries in order to free additional memory resources that were required to increase the size of the 6LoWPAN reassembly buffer from 240 to 1280 byte. This change allowed us to evaluate the behavior and the overhead of our presented defense mechanisms for different IPv6 packet sizes that cause 6LoWPAN packet fragmentation but do not yet require fragmentation at the IPv6 layer[7]. Due to this modification, the maximum node connectivity in our evaluation setup, however, was limited to 6 neighbors per constrained device. As different deployment scenarios will require their own adapted configurations of the network stack, we primarily focus our discussion on the worst-case overheads observed for a packet size of 1280 byte and highlight the results for the buffer size of 240 byte in case of significant deviations.

Concerning the cryptographic primitives, the SHA-1[8] provided the baseline for the hash operations in the content-chaining scheme. Our implementation of a DM-based and an MMO-based hash function then allowed us to quantify the computation and memory impacts of AES hardware support[9] in the context of this defense mechanism.

---

[6] In Section 2.2.1, we give a brief overview and a comparison of the platforms used in this thesis.

[7] IPv6 packets below 1280 byte must not be fragmented according to the IPv6 standard [DH98].

[8] The implementation of the SHA-1 hash function was based on the *relic toolkit* [relic].

[9] AES hardware support was provided by the CC2420 radio interface of the TMote Sky platform.

In our evaluation, we did not consider the computation overhead caused by active link layer security, but included the transmission overhead resulting from the *maximum length* of the security header at the link layer. To this end, we reduced the available 6LoWPAN payload size by 21 byte. Moreover, the IPv6 and UDP headers were compressed to the extend discussed in Section 2.4.1 via the LOWPAN_IPHC and LOWPAN_NHC compression mechanisms. In this, we also assumed that the prefix of the IPv6 source address was compressible due to the availability of the corresponding 6LoWPAN compression context inside the constrained node network. As a result, IPv6 packets with a size above 98 byte, i.e., excluding lower layer headers, were fragmented by the 6LoWPAN layer prior to their transmission and reassembled at the fragment recipient as described in Section 2.4.2. FRAG1s then contained up to 88 byte of IPv6 header information and IPv6 packet payload, whereas FRAGNs carried between 1 and 72 byte of IPv6 packet payload. Notably, by accounting for the additional link layer header overhead and by only considering modest header compression ratios, the evaluation results indicate worst-case overheads regarding the number of 6LoWPAN fragment transmissions pertaining to an IPv6 packet.

6.7.1 Effective Defense Against the Identified Attacks

We now analyze the practicability of the fragment duplication and the buffer reservation attack. Moreover, we show that the content-chaining scheme and the split buffer approach effectively protect against these 6LoWPAN fragmentation attacks.

6.7.1.1 Evaluating the Fragment Duplication Attack

To substantiate the practicability of the fragment duplication attack, we employed a network setup consisting of two wirelessly interconnected constrained devices, i.e., a sender and a receiver. The sender ran a simple Contiki application that transmitted a constant stream of fragmented UDP packets with a pre-defined packet content. Moreover, the sender's 6LoWPAN layer was modified to *duplicate* and to re-transmit one of the legitimate 6LoWPAN fragments with an altered fragment payload for each transmitted UDP packet. This additional fragment simulated the spoofed attack fragment from an eavesdropping adversary Eve. The receiver, in turn, counted the successfully reassembled UDP packets containing the expected UDP packet content.

With this evaluation setup, we then compared the following two scenarios. In the first scenario, the sender and the receiver ran an otherwise unmodified 6LoWPAN layer. This scenario allowed us to evaluate the practicability of the fragment duplication attack. In the second scenario, we additionally activated the content-chaining scheme in order to show its effectiveness against the identified attack. In both scenarios, the sender transmitted 100 UDP packets of 240 and 1280 byte at the 6LoWPAN layer, respectively. We note that we observed equal results for both packet sizes.

The results for the first evaluation scenario showed that the unmodified 6LoWPAN fragmentation mechanism of the Contiki OS reassembled the fragmented UDP packets despite overlapping fragment content. More precisely, the attack fragment overwrote the legitimate fragment content. As the UDP checksum was disabled due to our 6LoWPAN header compression setup, this modification of the packet content went unnoticed by the operating system. Still, our UDP application at the receiver correctly reported a Packet Delivery Ratio (PDR) of 0 % as the reassembled packet

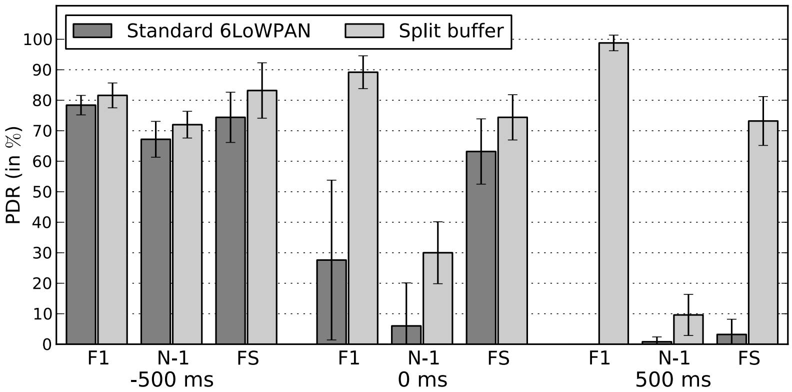


Figure 6.17 Successful reception of legitimate UDP packets (1280 byte) at the application layer of the target device during the buffer reservation attack. The split buffer approach consistently outperforms the unmodified 6LoWPAN fragmentation mechanism of the Contiki OS independent from the adversary's sending behavior and the sender's transmission offset.

content differed from the expected packet content. Notably, we observed the same low PDR at the application layer when enabling the UDP checksum or when discarding packets with overlapping fragments as mandated by the 6LoWPAN standard.

Contrary to the unprotected 6LoWPAN fragment transmissions, the content-chaining scheme achieved a PDR of 100 %. Each received 6LoWPAN fragment was verified correctly by the receiver. Based on these results, we conclude that the fragment duplication attack indeed is practical to mount for an in-network adversary Eve. Moreover, the content-chaining scheme effectively mitigates the identified attack.

6.7.1.2 Evaluating the Buffer Reservation Attack

For the evaluation of the buffer reservation attack, we employed a network setup consisting of three constrained devices: a sender, an adversary, and a target device. The sender legitimately transmitted a constant stream of fragmented UDP packets to the target device, whereas the target device counted the successfully reassembled UDP packets at the application layer. The adversary, in turn, sent its attack fragments in parallel to the transmissions between the sender and the target device. As depicted in Figure 6.17, we thereby distinguished between three relative reception times of attack and legitimate fragments at the target device to account for the various reassembly buffer states. In particular, the sender transmitted its packets 500 ms before, simultaneous to (i.e., 0 ms), and 500 ms after the first attack fragment from the adversary. While the -500 ms offset allowed for legitimate transmissions without malicious interference, the 500 ms offset represents a situation where the adversary successfully occupied the reassembly buffer at the target device. In contrast, the fragment order was not pre-determined in case of simultaneous transmissions.

Based on this evaluation setup, we then compared the following two scenarios. In the first scenario, we equipped the target device with an unmodified 6LoWPAN layer of the Contiki OS. This allowed us to confirm the practicability of the buffer reservation attack. In the second setup, we additionally activated the split buffer approach on the target device to demonstrate its effectiveness against the buffer reservation attack. We thereby set the window value w to 250 ms in order to account for potentially high jitter with respect to the reception of legitimate fragmented packets.

To also evaluate the adequacy of the packet discard strategy, we additionally differentiated between the following three malicious sending behaviors of the adversary (see Figure 6.17): i) exclusive transmission of FRAG1s (F1), ii) fragment bursts excluding the last FRAGN (N-1), and iii) fragment spreading across the reassembly timeout (FS). For each combination of the above configurations, we then measured the PDR at the application layer of the target device for 10 runs with 25 legitimate packets and packet sizes of 240 and 1280 byte, respectively. We note that the packet size only showed a negligible impact on the evaluation results in case of no protection and further improved the performance of the split buffer approach for packets of 240 byte. We, thus, focus our discussion on the results for packets of 1280 byte.

As illustrated in Figure 6.17, the split buffer approach consistently outperformed the unmodified 6LoWPAN reassembly mechanism of the Contiki OS. In fact, even when attack fragments should not have interfered with the reception of legitimate fragments (see "-500 ms" in Figure 6.17), the PDR was slightly higher for the split buffer approach than for the unmodified 6LoWPAN layer. As the main reason for this performance gain, we identified that periodic control messages of the RPL protocol also required fragmentation in our evaluation setup due to the consideration of the additional link layer security overhead and the modest header compression ratios. The single 6LoWPAN reassembly buffer, however, did not allow for two packets to be reassembled at the same time. As a result, the interleaved reception of the fragmented RPL control messages interfered with the correct reassembly of our fragmented UDP packets at the target device despite the relatively small size of the RPL messages, i.e., 2 6LoWPAN fragments. In contrast, the split buffer approach afforded an interleaved fragment reception as it assigns buffer resources on a per-fragment instead of a per-packet basis. This circumstance effectively demonstrates the efficient utilization of the available buffer resources with the split buffer approach.

As shown with the "500 ms" case in Figure 6.17, the buffer reservation attack succeeded for all attack behaviors in case of an unmodified 6LoWPAN reassembly mechanism, i.e., when the legitimate fragment was received after the first attack fragment. In these situations, the PDR dropped to as low as 0 %. Contrarily, the PDR remained at up to 98 % with the split buffer approach depending on the specific attack behavior. Notably, the adversary had to transmit packets consisting of a large number of fragments in short bursts in order to decrease the PDR with the split buffer approach (see "N-1" in Figure 6.17). In this case, the packet discard strategy, however, quickly decreased the score of the overall attack packet once the delay of the last missing attack fragment was detected as a significant deviation from the previous sending behavior. In fact, we observed that all attack fragments were purged from the split buffer when a new fragmented packet was received by the target device about 350 ms after the reception of the N-1^{th} attack fragment. Importantly, such quick discarding considerably decreases the amplification effects of a malicious buffer reservation compared to the otherwise achievable reservation time of up 60 s.

For simultaneous fragment transmissions, the split buffer approach showed similar results as observed above (compare "0 ms" to "500 ms" in Figure 6.17). Also here, our approach persistently outperformed the unmodified 6LoWPAN reassembly mechanism. However, it is worth noting that the results for the *unmodified* 6LoWPAN reassembly mechanism differed noticeably from the expected PDR of about 50 % regarding the direct competition for the single reassembly buffer at the target device.

When further investigating this observation, we found that the lower layers in the network stack of the adversary and the legitimate sender buffered up to 6 fragments before transmission over the wireless link. In case of fragment spreading (FS), this led to situations where the transmission of the attack fragments were delayed such that legitimate fragments were received first by the target device. The legitimate fragments then claimed the reassembly buffer at the target device, thus leading to a PDR that exceeded the expected value with 63 % for the unmodified 6LoWPAN reassembly mechanism (see "FS" for "0 ms" in Figure 6.17). Similarly, we observed that the adversary sent at least one attack fragments prior to the legitimate sender in the majority of the measurement runs for exclusive FRAG1 transmissions (F1) and fragment bursts (N-1). Correspondingly, the PDR of the legitimate fragment transmissions dropped below the expected value with 27 % and 6 %, respectively.

The above transmission behaviors similarly applied to our measurement runs for the split buffer approach. In case of fragment bursts, the adversary, however, was able to send its attack fragments prior to the legitimate fragments in fewer cases than for the unmodified 6LoWPAN reassembly mechanism. As a result, the split buffer approach achieved an increased PDR of 30 %. With equally distributed competition between the adversary and the legitimate sender, we would expect this value to be around 50 % for both the unmodified 6LoWPAN reassembly mechanism and our split buffer approach. In contrast, the observed advantage of the adversary did not have an impact on the PDR of 89 % for exclusive FRAG1 transmissions. This is because the split buffer approach allowed for an interleaved reception of a limited number of 6LoWPAN fragments. Moreover, the packet discard strategy disposed of the single attack fragment in the split buffer when a buffer overload situation occurred. Consequently, also for simultaneous fragment transmissions, the split buffer approach requires the adversary to send a large number of fragments in short bursts in order to perceivably decrease the PDR of legitimately fragmented packets (see "N-1" in Figure 6.17). The long delay after the N-1^{th} attack fragment then, however, causes the packet discard strategy to quickly decrease the packet score of the received attack fragments as similarly discussed for the "500 ms" case above.

Overall, we conclude that unprotected 6LoWPAN fragmentation transmissions are vulnerable to the buffer reservation attack. In contrast, the split buffer approach mitigates this attack by forcing the adversary into sending short successions of a large number of fragments. Combined with the packet discard strategy, this prevents the adversary from exploiting the amplification effects that stem from the reservation of the 6LoWPAN reassembly buffer. In addition, the split buffer approach also affords an interleaved reception of fragmented IPv6 packets. Hence, our presented approach additionally improves fragment processing under normal network conditions.

6.7.2 Transmission Overhead

Besides the practicability of the identified 6LoWPAN fragmentation attacks and the effectivity of the presented defense mechanisms, we also analyzed the computation, energy, memory, and transmission overheads of our solutions to evaluate the trade-offs for the gained protection. We now discuss the observed transmission overheads and examine the remaining evaluation results in the subsequent sections.

To evaluate the transmission overhead of the presented defense mechanisms, we analyzed the wireless transmission of 6LoWPAN-fragmented packets between two con-

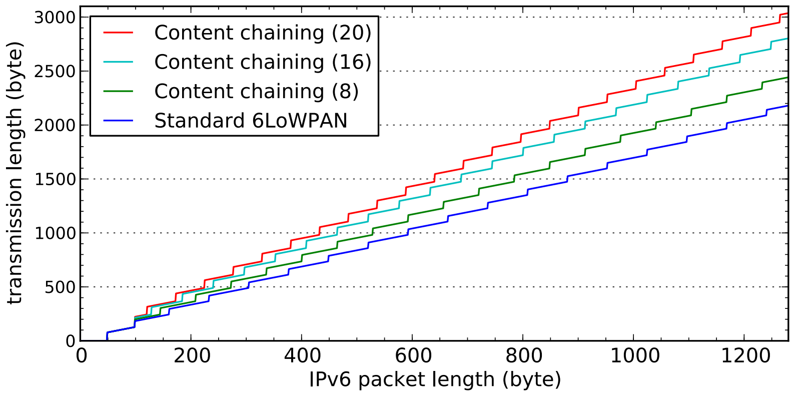


Figure 6.18 Transmission overhead of the content-chaining scheme for different token and packet sizes (with all headers) compared to the standard 6LoWPAN fragmentation mechanism. The content token size in byte is shown in brackets. Truncated content tokens achieve a significant reduction of the transmission overhead, especially for large fragmented packets.

strained devices with the *wireshark* tool. In doing so, we focused on the maximum length of FRAG1s and FRAGNs with (i) the standard 6LoWPAN fragmentation mechanism, (ii) the content-chaining scheme, and (iii) the split buffer approach. We then extrapolated the observed results for larger IPv6 packet sizes. The *split buffer approach* is a local defense mechanism and, thus, incurs the same transmission overheads as the standard 6LoWPAN fragmentation mechanism. The *content-chaining scheme*, contrarily, requires the transmission of additional token information. Hence, the following discussion focuses on the results for the content-chaining scheme.

Importantly, unfragmented IPv6 packets do *not* contain token information. Hence, packets with IPv6 header information and IPv6 packet payload of up to 98 byte did not cause an additional transmission overhead with the content-chaining scheme. Fragmented IPv6 packets, however, resulted in a minimum token overhead of 8 byte per transmitted 6LoWPAN fragment. Correspondingly, we observed an additional overhead of 262 byte for the protection of a fragmented IPv6 packet of 1280 byte (see the difference between the blue and the green lines in Figure 6.18). 160 byte thereby stemmed from the 8 byte content tokens. Moreover, the token-induced decrease of the available fragment payload space caused the transmission of two extra 6LoWPAN fragments with additional link layer and 6LoWPAN header information of 102 byte. This reduction in payload space is illustrated by the shorter step length for the content-chaining results in Figure 6.18. To put these numbers into perspective, the content-chaining scheme increased the overall transmission overhead by about 12.02 % compared to standard 6LoWPAN fragmentation. This is because an IPv6 packet of 1280 bytes already caused transmissions of 2180 byte due to link layer and 6LoWPAN header overheads, i.e., even without the content-chaining scheme.

To further quantify the benefit of truncated content tokens, we also compared the transmission overheads of the content-chaining scheme for token sizes of 8, 16, and 20 byte. Focusing on IPv6 packets of 1280 byte, the evaluation results show that tokens of 16 byte required the transmission of 23 fragments instead of 18 for the standard 6LoWPAN fragmentation mechanism and, therefore, increased overall transmissions from 2180 byte to 2803 byte. Likewise, content tokens of 20 byte increased the transmission overhead to a total of 25 fragments and 3037 byte. Hence, by truncating content tokens to 8 byte, we can reduce the additional transmission overhead of the content-chaining scheme by about 57.9 % compared to 16 byte content tokens and by about 69.4 % compared to 20 byte content tokens (see Figure 6.18).

To summarize, the split buffer approach does not cause any additional transmission overheads compared to the standard 6LoWPAN fragmentation mechanism. Moreover, the content-chaining scheme incurs moderate transmission overheads with content tokens of 8 byte. Importantly, by employing the attack notification extension of the content-chaining scheme (see Section 6.4.6.1), these transmission overheads only occur while actively protecting against the fragment duplication attack. During normal network operation, this extension affords the content-chaining scheme to remain inactive, thus reducing the additional transmission overhead to zero.

6.7.3 Computation Overhead

For the evaluation of the computation overhead, we first analyzed the *per-fragment* performance impact of the underlying hash function on the token generation and verification of the content-chaining scheme. We then evaluated the *per-packet* computation overhead of the content-chaining scheme and the split buffer approach.

6.7.3.1 Per-Fragment Overhead of the Content-Chaining Scheme

To evaluate the performance impact of the underlying hash function on the content-chaining scheme, we measured the per-fragment computation overhead of SHA-1 on a single constrained device to establish a baseline for comparison purposes. We then repeated our experiments for the DM-based and the MMO-based hash functions (see Section 6.4.6.3). We thereby increased the payload size from 8 to 72 byte in 8 byte steps and ran 100 measurements for each hash function and input size combination.

As depicted in Figure 6.19, both AES-based hash functions consistently outperformed SHA-1 independent from the fragment payload size. By employing the DM-based hash function instead of SHA-1, the per-fragment computation overhead for the content-chaining scheme decreased by as much as 82.59 % (see input length of 8 byte in Figure 6.19). In contrast, the DM-based and MMO-based hash functions

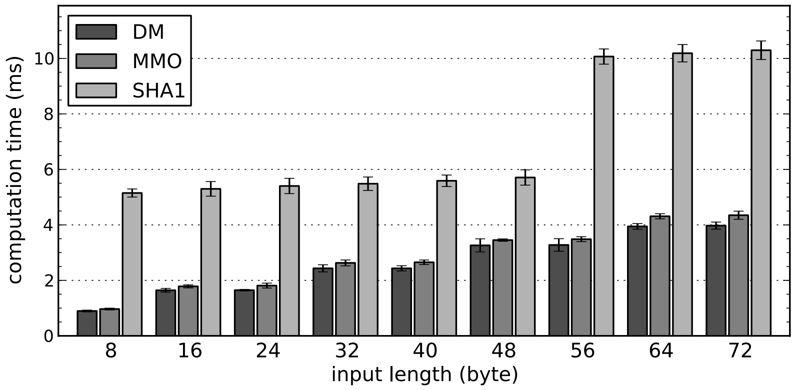


Figure 6.19 Computation overhead for the generation or the verification of a single content token. The bars represent the average over 100 measurement runs for each input size. The error bars illustrate the observed the standard deviation. Both AES-based hash functions consistently outperform SHA-1. Moreover, the DM-based and MMO-based hash functions only exhibit a modest performance difference.

only exhibited a modest performance difference. While the average computation time for the generation or verification of a single content token increased from 0.90 ms to 4.72 ms for growing payload sizes for the DM-based hash function, this overhead increased from 0.97 ms to 5.15 ms for the MMO-based hash function. Hence, the DM-based hash function showed a small performance advantage when employing a AES-based hash function with hardware support in the content-chaining scheme.

Interestingly, the computation overhead of the cryptographic primitives did *not* increase with each 8 byte step regarding the input length. This is especially visible for the SHA-1 hash function (compare input lengths from 8 to 48 byte in Figure 6.19) but also applies to the AES-based hash functions (e.g., compare input lengths of 16 and 24 byte in Figure 6.19). The reason for these relatively stable results despite an increasing input length is the internal block size and the iterative construction with which the cryptographic primitives operate. Moreover, we note that the observed computation overhead increased from an input length of 8 to 16 byte for the AES-based hash functions and from 48 to 56 byte in case of SHA-1 despite their internal block size of 16 byte and of 64 byte, respectively. This is because the hash function input has to be extended by an additional block if the length padding information does not fit into the last input block, thus causing another hash function iteration.

To summarize, by leveraging the AES hardware support with a DM-based or an MMO-based hash function, the computation overhead of the content-chaining scheme can be reduced significantly compared to the use of a standard hash function such as SHA-1 without hardware support. Moreover, the DM-based hash function marginally outperformed the MMO-based hash function. We note that, for this reason, we employed the DM-based hash function in the subsequent performance measurements.

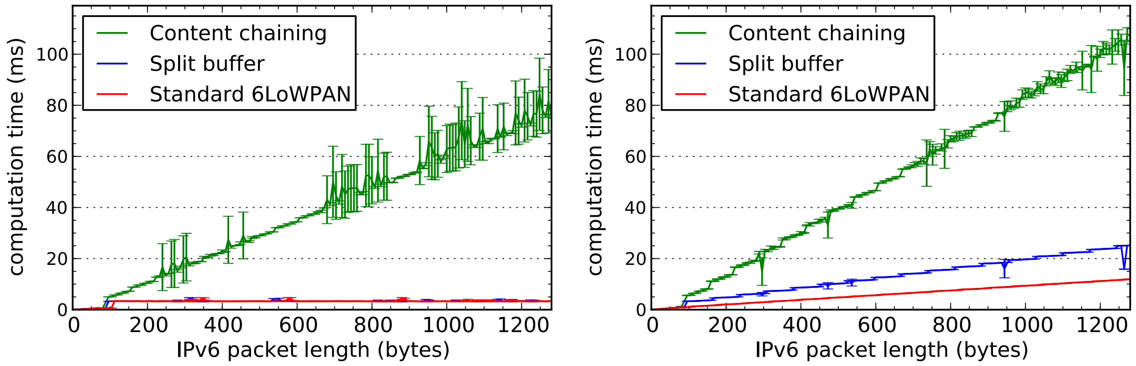
6.7.3.2 Overall Per-Packet Computation Overhead

For the evaluation of the overall per-packet computation overhead, we measured the processing time at the 6LoWPAN layer of two wirelessly connected constrained devices for (i) the standard 6LoWPAN fragmentation mechanism, (ii) the content-chaining scheme, and (iii) the split buffer approach. One device thereby acted as the sender, whereas the other device represented the recipient. Moreover, we increased the packet size from 56[10] to 1280 byte in 8 byte steps and performed 10 measurement runs for each fragmentation mode and packet size combination. We recollect that only packets exceeding 98 byte required fragmentation at the 6LoWPAN layer.

As illustrated by the largely overlapping evaluation results for the split buffer approach and for the standard 6LoWPAN fragmentation mechanism in Figure 6.20a, the split buffer approach only incurs a marginal computation overhead at the sender compared to an unmodified implementation of the 6LoWPAN fragmentation mechanism. However, the split buffer approach adds a processing overhead of up to 13.19 ms for fragmented packets of 1280 byte at the fragment recipient compared to an unmodified reassembly mechanism (see Figure 6.20b). This overhead primarily stems from the defragmentation procedure during packet reassembly (see Section 6.5.3) and the additional buffer management of the split buffer approach.

---

[10] We chose 56 byte as the smallest packet size as it represents a minimal IPv6 packet with an IPv6 header of 40 byte, a UDP header of 8 byte, and up to one full octet of IPv6 packet payload.



(a) Sender side **(b)** Receiver side

Figure 6.20 Processing time at the 6LoWPAN layer for the standard 6LoWPAN fragmentation mechanism, the content-chaining scheme, and the split buffer approach. The error bars denote the standard deviation. The split buffer approach only causes a modest overhead, whereas the content-chaining scheme incurs a noticeable overhead at the sender and at the receiver.

The content-chaining scheme incurs a noticeable computation overhead at the sender and at the receiver. These overheads mainly originate from the construction of the content chain at the sender and the token verification at the recipient. As a result, the content-chaining scheme adds a maximum of 64.22 ms at the sender side and of 95.23 ms at the receiver side for the processing of fragmented packets with a size of 1280 byte. Moreover, while the sender and the receiver both perform the same cryptographic operations per fragment, a comparison of Figure 6.20a and Figure 6.20b shows an observable increase in computation overhead on the receiver side. This additional computation overhead stems from the fact that the content-chaining scheme also facilitates the buffer management functionality of the split buffer approach to handle out-of-order fragments. Consequently, the computation overhead of the content-chaining scheme also includes the majority of the overhead for the split buffer approach. Furthermore, the fragment recipient has to search the reassembly buffer slots for unverified out-of-order fragments in case of fragment reordering. This also increases the overhead on the receiver side compared to the sender side. Lastly, we note that the sporadically high standard deviation is the result of outlier hash operations with run-times above 30 ms during which the hardware reported as busy.

During our evaluation, we did not explicitly measure the computation overheads of our defense mechanisms at a *fragment forwarder*. Still, the above evaluation results allow to analytically derive overhead approximations for these on-path devices. The computation overhead at a fragment forwarder thereby strongly depends on the fragment forwarding mechanism employed in the constrained node network. More precisely, in case of mesh-under routing, a fragment forwarder is oblivious to the packet fragmentation at the 6LoWPAN layer. Consequently, neither the content-chaining scheme nor the split buffer approach cause additional overheads for this fragment forwarding mechanism. In contrast, the computation overhead of the content-chaining scheme is similar to the overhead at the receiver for enhanced route-over routing and largely resembles the sum of the sender and the receiver overheads in case of route-over routing. This is because forwarders with enhanced route-over verify the content tokens and immediate forward the individual 6LoWPAN fragments, whereas route-over-based fragment forwarders reassemble and fragment each traversing fragmented packet. For the same reason, the split buffer approach results in the same

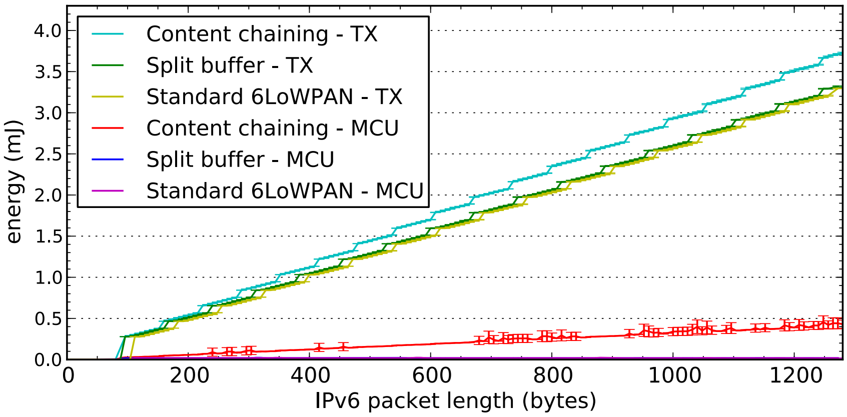


Figure 6.21 Energy estimation for the computations (MCU) and transmissions (TX) resulting from our presented defense mechanisms on a sending node. The estimated power consumption of the standard 6LoWPAN fragmentation mechanism is given as a baseline. The content-chaining scheme causes non-negligible energy overheads, whereas the split buffer approach exhibits comparable results to the standard 6LoWPAN fragmentation mechanism.

computation overhead at a fragment forwarder as at the receiver in case of route-over routing and does not incur any additional overhead for enhanced route-over.

Overall, the split buffer approach exhibits modest computation overheads that are limited to the communication end-points inside a constrained node network for all fragment forwarding mechanisms except for route-over routing. Importantly, these overheads not only denote a trade-off for a gain in robustness against the buffer reservation attack, but also facilitate an improved processing of interleaved packets as discussed in Section 6.7.1.2. The content-chaining scheme, however, incurs noticeable computation overheads at the sender and the fragment verifying nodes. This overhead is a trade-off for a gain in protection against the fragment duplication attack. Still, the attack notification mechanism (see Section 6.4.6.1) allows to limit these computation overheads to situations when constrained devices actually profit from spending their processing resources for the content-chaining scheme.

6.7.4 Energy Evaluation

Especially the content-chaining scheme causes both, increased computation and transmission overheads, compared to the standard 6LoWPAN fragmentation mechanism. To evaluate the combined impact of these overheads, we additionally estimated the energy resources required by the MCU for the packet processing at the 6LoWPAN layer as well as by the radio module for the transmission of the corresponding 6LoWPAN fragments. To this end, we employed the same network setup as described in Section 6.7.3.2 and leveraged the energy estimation facilities provided by the Contiki OS [DEFT11]. We assumed an idealized energy consumption of 1.8 mA for the MCU and of 19.5 mA for the radio module at a supply voltage of 3 V as indicated in the Tmote Sky datasheet [tmote]. Moreover, we restricted our energy evaluation to the fragment sender in order to prevent overhead misattribution for inbound 6LoWPAN fragments that, e.g., belong to the RPL protocol.

As illustrated in Figure 6.21, the split buffer approach exhibits highly similar results to the standard 6LoWPAN fragmentation mechanism regarding the energy

Functionality	ROM (byte)	RAM (byte)
Contiki OS	37788	8478
+ Split buffer	40052 [+2264]	9014 [+ 536]
+ Content chaining	41108 [+3320]	9402 [+ 924]
+ Both mechanisms	41750 [+3962]	9502 [+1024]

Table 6.1 ROM and RAM requirements of the split buffer approach and the content chaining scheme in byte. Numbers in brackets denote the added overhead by our presented defense mechanisms when compared to the Contiki OS base overhead. Combined, these mechanisms require about 1 kB of RAM and just below 4 kB of ROM.

expenditure for both computations and transmissions. In fact, our energy estimation indicates a combined additional energy overhead for the split buffer approach that is as low as 0.72 % for fragmented IPv6 packets of 1280 byte. In contrast, the content-chaining scheme shows an overall energy overhead increase of about 25.30 % compared to the standard 6LoWPAN fragmentation mechanism for fragmented IPv6 packets of 1280 byte. This increased energy overhead stems in equal parts from the additional computation (i.e., 0.42 mJ) and transmission overheads (i.e., 0.42 mJ).

Based on these results, we conclude that the split buffer approach has a marginal impact on the lifetime of energy-constrained devices, thus affording it to remain active continuously. In contrast, the content-chaining scheme causes comparably high energy overheads. This energy expenditure would reduce the overall lifetime of an energy-constrained device by up to 25 % assuming that this device transmitted a *constant* stream of 6LoWPAN-fragmented IPv6 packets. However, we do not expect constrained devices to continuously be involved in the transmission of upper layer application data that requires packet fragmentation at the 6LoWPAN layer. As a result, the energy expenditure of the content-chaining scheme in real-world deployment scenarios would significantly be reduced. Moreover, it is important to note that the attack notification extension of the content-chaining scheme additionally allows to decrease its energy expenditure to near zero during network normal operation.

6.7.5 RAM and ROM Overhead

Constrained devices often are equipped with only very limited memory resources. The Tmote Sky platform, e.g., provides constrained devices with 10 kB of RAM and 48 kB of ROM. Hence, to estimate the RAM and ROM impact of our presented defense mechanisms, we analyzed the Contiki OS binaries that we employed during the evaluation with the *msp430-size* tool[11]. The first binary then contained an unmodified 6LoWPAN layer, whereas the remaining three binaries additionally included the required functionality for (i) the content-chaining scheme, (ii) the split buffer approach, and (iii) a combination of both presented defense mechanisms.

As shown in Table 6.1, the split buffer approach increases the ROM overhead by 2264 byte compared to the Contiki OS base overhead. Similarly, the content-chaining scheme adds about 3320 byte to the overall ROM requirements of the Contiki OS. In the latter case, about 878 byte result from the implementation of the DM-based hash function for the content-chaining scheme. This overhead would further grow

---

[11] The *msp430-size* tool is part of the GCC toolchain for the MSP430 MCU [mspgcc].

to 922 byte for the MMO-based hash function and to 3232 byte in case of SHA-1. Interestingly, a major part of the ROM overhead that is caused by the split buffer approach and the content-chaining scheme stems from the buffer management functionality that is shared between both mechanisms. This fact is reflected by the low additional ROM overhead of 1698 byte and 642 byte for the combined defense mechanisms when compared to the individual overheads for the split buffer approach and the content chaining scheme, respectively. Consequently, when one of our defense mechanisms is already deployed on a constrained device, the other defense mechanism only adds a marginal ROM overhead. Notably, while not optimized for minimal ROM overhead, such a combined deployment requires just below 4 kB of ROM.

With respect to the RAM requirements, the split buffer approach and the content-chaining scheme require about 536 byte and 924 byte additional memory when compared to the Contiki base overhead, respectively. For the split buffer approach, this overhead stems from the need to over-provision each buffer slot such that it can hold a 6LoWPAN fragment of maximum length including 6LoWPAN header information, i.e., a total of 81 byte. As a result, the memory overhead for a reassembly buffer that supports IPv6 packets of 1280 byte increases by 18.13 %. The remaining overhead results from additional per-packet management information that is required for the packet discard strategy and for the maintenance of the individual buffer slots. This additional overhead, however, is not only a trade-off for a gain of security, but also enables a constrained device to process interleaved fragmented IPv6 packets during normal operation. The content-chaining scheme, in turn, primarily adds an overhead of 8 byte for the last verified token to this per-packet management information.

To conclude, the content-chaining scheme successfully protects constrained devices against the fragment duplication attack. In doing so, it incurs noticeable computation, energy, and transmission overheads. Importantly, the attack notification extension allows to reduce these overheads to zero unless a constrained device actually profits from spending the required resources for the content-chaining scheme due to the presence of an in-network adversary. The split buffer approach, likewise, mitigates the buffer reservation attack. In contrast to the content-chaining scheme, it, however, only incurs marginal computation and energy overheads and does not impact the transmission overhead of fragmented packets. This affords the split buffer approach to be activated continuously. Moreover, both the content-chaining scheme and the split buffer approach only require modest memory resources, especially when a constrained device already employs one of the introduced mechanisms. Overall, the presented defense mechanisms, therefore, provide resource-conscious protection against the identified fragment duplication and the buffer reservation attacks.

6.8 Related Work

For our discussion of related work, we distinguish between the following three research directions: (i) fragmentation attacks and defense mechanisms at the 6LoWPAN and the IP layer, (ii) efficient authentication schemes for packet streams, and (iii) related non-cryptographic approaches. Moreover, we refer to Section 6.1 for a detailed discussion of previously identified IP-level fragmentation attacks.

6.8.1 Fragmentation Attacks and Defense Mechanisms

Both, research and standardization, previously investigated potential fragmentation attacks at the 6LoWPAN and the IP layer as well as proposed corresponding defense mechanisms. Regarding the 6LoWPAN layer, Kim [Kim08] lists several well-known IP-level fragmentation attacks and speculates that these attacks may also be applicable in the context of the 6LoWPAN fragmentation mechanism. Moreover, Kim suspects that replaying of fragmented IPv6 packets might be harmful. Triggered by this work, we conducted a thorough security analysis of the 6LoWPAN fragmentation mechanism and identified two design-level fragmentation attacks. During our analysis, we, however, could not confirm that replaying unaltered, fragmented IPv6 packets indeed is harmful. Instead, the identified fragment duplication attack requires the adversary to send at least one spoofed fragment with modified fragment content. Moreover, the defense mechanisms proposed by Kim, i.e., timestamps and nonces, do not suffice to protect against the identified fragmentation attacks as an eavesdropping adversary can simply spoof such non-cryptographic fragment content.

To prevent an adversary from exploiting overlapping IP fragments via correct guessing of the packet identification value (i.e., the datagram tag in case of 6LoWPAN), Gilad et al. [GH13] propose to generate these values with a pseudo-random function instead of employing a simple counter-based approach. While mitigating the considered IP-level attack by reducing the efficiency of guessing, this mechanism does not protect constrained devices against the fragment duplication attack as Eve can overhear legitimate fragmented packets and, thus, is not required to guess. Moreover, the authors propose to avoid fragmentation at and below the IP layer by employing path MTU discovery mechanisms. The avoidance of such fragmentation, however, may often not be possible in constrained node networks due to the exceedingly low frame size and the lack of fragmentation support at many upper layer protocols.

Concerning attacks based on missing packet fragments, Kaufman et al. [KPS03] present defense mechanisms that are specifically designed for the IKEv2 protocol. More precisely, the authors propose to re-design the initial IKEv2 handshake messages such that their message size does not cause packet fragmentation. Moreover, Kaufman et al. propose to include a cookie-based DoS protection mechanism in these initial handshake messages. After a successful cookie verification, the IKEv2 layer then clears the communication partner for IP-level fragmentation of the subsequent IKEv2 handshake messages. Importantly, while these protocol mechanisms allow to restrict IP-level fragmentation to weakly verified end-points, they are highly dependent on the employed upper layer protocol and require bidirectional data flows for the cookie exchange. In contrast, our presented defense mechanisms are protocol-independent and also support exclusively uni-directional data flows. Moreover, it is important to note that the severe frame-size limitations in constrained node networks may require even small handshake messages to be fragmented, thus rendering the proposed defense mechanisms by Kaufman et al. ineffective for such networks.

Gont [Gon11] proposes to decrease the reassembly timeout and to use multiple independent buffer pools to handle fragmented packets with missing packet fragments. Specifically, the author suggests to maintain one buffer pool for IPsec and another one for all remaining protocols. To handle overload situations for the IPsec buffer pool, Gont introduces flushing policies that prefer partial IPsec packets for which a number of validity checks can be performed to those that fail these checks or that do not yet provide sufficient packet content for validation purposes. The provisioning of multiple independent buffer pools, however, is less memory efficient than our split buffer approach as memory is still allocated on a per-packet instead of a per-fragment basis. Moreover, while the flushing policies proposed by Gont are tailored to IPsec, our packet discard strategy can be applied to a variety of upper layer protocols.

6.8.2 Efficient Authentication Schemes for Packet Streams

Several authentication schemes have been proposed that aim at providing efficient data origin authentication for packet streams. Similar to our work in this chapter, Gennaro et al. [GR97] propose to employ hash chains to authenticate digital streams of finite length. Our content-chaining scheme differs from their approach by forgoing the use of public-key cryptography to bind the anchor element of the generated hash chain to a cryptographic sender identity. Instead, a fragment recipient considers the anchor element in the first received FRAG1 as legitimate and discards subsequently arriving FRAG1s that indicate the same IPv6 packet. Moreover, Wong et al. [WL98] propose the use hash trees for authentication purposes in order to afford the immediate verification of out-of-order packets. We consciously decided against such a tree-based construction for our content-chaining scheme as it significantly increases the per-fragment computation and transmission overheads in constrained node networks. Instead, we introduce fragment forwarding and fragment discard policies that allow to efficiently handle out-of-order 6LoWPAN fragments.

A rich body of research investigates efficient stream authentication mechanisms that are resilient to packet loss. These mechanisms most notably include hash chain schemes with delayed token disclosure [PTSC00, PST$^+$02], with redundant traversal paths [GM01], and with erasure coding [PCS02, PM03] as well as one-time signature schemes [Per01]. While all of these schemes can be employed to verify the data origin of 6LoWPAN-fragmented IPv6 packets, they incur significant verification delays or considerably increase the computation and transmission overheads compared to our content-chaining scheme. Moreover, we note that the protection of the 6LoWPAN fragmentation mechanism does not need to be robust against fragment loss. This is because the loss of a single 6LoWPAN fragment invalidates the entire associated IPv6 packet according to the 6LoWPAN standard. Hence, the increased overhead of these loss-resilient schemes would be spent without achieving an additional gain.

6.8.3 Related Non-Cryptographic Approaches

Related non-cryptographic approaches do not specifically focus on the protection of constrained devices against the identified 6LoWPAN fragmentation attacks. Instead, they aim at improving the forwarding and reassembly properties for packets that are fragmented at the 6LoWPAN layer. We now discuss to which extend these approaches also provide protection against our identified fragmentation attacks.

Thubert et al. [TH14] recently proposed recoverable 6LoWPAN fragments that require a reassembling node to explicitly confirm fragment reception via an additional 6LoWPAN acknowledgement packet. Negatively acknowledged fragments then are retransmitted by the original fragment sender. Without protection by our content-chaining scheme, such fragment recovery, however, still is susceptible to the fragment duplication attack. This is because an adversary can also attack fragment retransmissions. Still, a reassembling node may leverage the fragment recovery mechanism to probe for malicious sending behavior in case of the buffer reservation attack by requesting missing packet content via an acknowledgement packet. Such probing, however, would be triggered *after* the detection of suspicious sending behavior and would involve at least one RTT to determine if the potentially malicious sender actually (re-)transmitted the requested 6LoWPAN fragments. At this point, the packet discard strategy of our split buffer approach already penalized malicious sending behavior. Thus, the additional transmission overhead caused by the fragment recovery mechanism does not result in significantly improved security properties compared to our split buffer approach. Moreover, we note that 6LoWPAN fragment recovery often is considered inadvisable as it breaks the best-effort semantics of IP packet delivery and interferes with retransmissions at the upper layers in the network stack.

With the goal to increase throughput and energy efficiency for bulk data transfers, pipelining and burst forwarding were proposed as enhancements at the data link layer. Regarding 6LoWPAN fragmentation, pipelining [RCBG10, DC14] allows all fragments that belong to a specific IPv6 packet to be forwarded to the final destination inside the constrained node network in a single fragment stream. Thus, while Eve would be able to overhear legitimate 6LoWPAN fragments, she could not inject spoofed attack fragments before the fragmented IPv6 packet is reassembled. This prevents her from mounting the fragment duplication attack. Currently proposed pipelining approaches, however, only consider the sequential download of bulk data from one constrained device at a time and monopolize the network resources on the forwarding path for a specific bulk transfer. Hence, pipelining typically is not applicable in the context of uncoordinated end-to-end transmissions. This stands in stark contrast to our content-chaining scheme that does not require coordination and is independent from the specific capabilities of the underlying link layer.

Burst forwarding as proposed by Dunquennoy et al. [DOD11] relaxes the concept of pipelining and allows 6LoWPAN fragments of a specific IPv6 packet to span across multiple bursts as well as a burst to contain 6LoWPAN fragments from multiple IPv6 packets. This, however, enables Eve to overhear legitimate 6LoWPAN fragments and to inject spoofed attack fragments between two bursts. In contrast to our content-chaining scheme, burst forwarding, therefore, still allows Eve to mount the fragment duplication attack. Moreover, contrary to our split buffer approach, neither of the above approaches affords protection against the buffer reservation attack as an adversary remains able to maliciously occupy the reassembly buffer at a target device. Still, we note that flash interleaving as proposed by Dunquennoy et al. in the context of burst forwarding would allow to increase the size of the reassembly buffer and, thus, the effort that an adversary has to put into mounting the buffer reservation attack. Flash interleaving, therefore, effectively complements our split buffer approach in case the reassembling node also is equipped with flash memory.

Subsequent to our work, Teo et al. [TASS13] proposed to maintain multiple dynamically created reassembly buffers in order to allow for an interleaved reception of 6LoWPAN-fragmented IPv6 packets. However, as discussed in Section 6.3.2, this approach only marginally increases the effort that Eve has to put into mounting the buffer reservation attack as she can still reserve all available reassembly buffer resources with only few 6LoWPAN fragments that each indicate a high overall IPv6 packet size. Moreover, maintaining multiple independent reassembly buffers is less memory efficient than our split buffer approach as reassembly resources are assigned on a per-packet instead of a per-fragment basis. Hence, our split buffer approach also exhibits advantageous properties during normal network operation.

Finally, we note that our work in this chapter is inspired by early buffer allocation and packet discarding schemes for packet-based IP communication over ATM networks. Specifically, we adopt the idea of sharing a single reassembly buffer for interleaved fragmented packets from CSVP as proposed by Wu et al. [WM95]. Moreover, our packet discard strategy follows the notion of penalizing misbehaving users as presented by Chan et al. [CWK97] for their fair packet discarding strategy.

6.9 Conclusion

Packet fragmentation at the 6LoWPAN layer is a common condition when performing the handshake of standard end-to-end IP security protocols such as DTLS with a device that is located inside a constrained node network. In this chapter, we analyzed if an adversary can exploit such packet fragmentation to block the establishment of secure end-to-end connections. Our analysis revealed two 6LoWPAN fragmentation attacks that are notably cheap to mount, thus allowing an adversary to even employ tightly resource-constrained devices in her attack: Both the fragment duplication attack and the buffer reservation attack only require the transmission of a single 6LoWPAN fragment to block the successful packet reassembly at a target device.

To protect constrained devices against the identified 6LoWPAN fragmentation attacks, we introduced two complementary, lightweight defense mechanisms. The content-chaining scheme enables a constrained device to cryptographically verify that a received 6LoWPAN fragment belongs to the indicated IPv6 packet on a per-fragment basis. As a result, the adversary is no longer able to mount the fragment duplication attack. The split buffer approach fosters per-fragment competition for the scarce buffer resources at a reassembling node between an adversary and legitimate fragment senders. In addition, the packet discard strategy of the split buffer approach severely penalizes suspicious changes in the fragment sending behavior. In combination, this prevents an adversary from exploiting the amplification effects that stem from the reservation of the 6LoWPAN reassembly buffer and, thus, frustrates the buffer reservation attack. Finally, the evaluation results confirm the practicability of the identified 6LoWPAN fragmentation attacks. Moreover, the results also show that the presented defense mechanisms effectively mitigate the identified attacks at moderate computation, energy, memory, and transmission trade-offs.

Importantly, the impact of the identified 6LoWPAN fragmentation attacks also extends beyond the connection establishment handshake of end-to-end IP security protocols such as DTLS, HIP DEX, and Minimal IKEv2. In fact, all types of IPv6 packet transmissions inside a constrained node network that exceed the frame size of the employed link layer are vulnerable to the identified fragmentation attacks. Such large IPv6 packet transmissions can occur for a wide range of application data, e.g., in case of incompressible sampling data or firmware updates [TH14]. Consequently, our presented defense mechanisms also facilitate the protection of other types of network traffic that we did not specifically consider within the scope of this thesis.

7

Discussion and Conclusion

IP technology for networked embedded devices enables the seamless interconnection of objects from the physical world in the Internet of Things (IoT). The achieved connectivity, however, also exposes this new class of network entities to similar network attacks as conventional IP-enabled hosts or services. The severity of these attacks is considerably aggravated in the IoT as attacks in the virtual world suddenly can have detrimental physical impact. Thus, effective network security is a vital aspect for the secure interconnection of embedded devices, hosts, and services in the IoT. The device and network constraints in the embedded domain and the resource asymmetry in the IoT, however, challenge the design of existing security solutions.

Our main goal with this thesis is to address the emerging protocol design challenges in the context of end-to-end security for the IP-based IoT and to contribute to the on-going efforts of adapting IP technology to IoT requirements. In our protocol analyses, we, therefore, specifically considered the IoT security protocol adaptations DTLS, HIP DEX, and Minimal IKEv2 that are currently proposed for standardization at the IETF. In addition, we analyzed the security properties of the 6LoWPAN adaptation layer as the main underlying transport mechanism of these protocols in the embedded domain. The key findings of our protocol analyses are:

i) The use of public-key cryptography in the handshake of the DTLS, HIP DEX, and Minimal IKEv2 protocols causes a significant computation overhead on networked embedded devices. This overhead renders the deployment of these protocols in their current state inefficient and even insecure. Specifically, we observe that the long processing times of public-key operations significantly hamper the availability and response time of networked embedded devices. Hence, such computationally expensive handshake operations should only rarely be employed and must be accounted for in the overall protocol design.

ii) A chief design goal of the DTLS, HIP DEX, and Minimal IKEv2 protocol adaptions is to preserve the protocol semantics of TLS, HIPv2, and IKEv2, respectively. TLS, HIPv2, and IKEv2, however, were developed with extensibility and flexibility in mind. Thus, the handshake messages of these protocols commonly contain dispensable protocol information. This information, however, constitutes undesirable transmission overhead in the embedded domain.

iii) DTLS, HIP DEX, and Minimal IKEv2 protocol implementations are prone to exhibit significant RAM and ROM requirements when employing public-key primitives in the performed protocol handshakes. We find that these memory requirements render a comprehensive, public-key-enabled protocol implementation infeasible for a wide range of memory-constrained embedded devices.

iv) The large message size in the DTLS, HIP DEX, and Minimal IKEv2 handshakes commonly causes packet fragmentation at the 6LoWPAN layer. We identify two design-level DoS attacks against the employed fragmentation mechanism. An adversary can exploit these attacks to block the correct reassembly of fragmented handshake messages at the 6LoWPAN layer and, thus, prevent the DTLS, HIP DEX, and Minimal IKEv2 handshakes from completing.

As part of our four core contributions, we introduced resource-conscious protocol mechanisms and a security architecture that resolve the identified protocol efficiency and security issues. Section 7.1 briefly summarizes these solutions. We then highlight the impact of our contributions in Section 7.2. Finally, we close this thesis with ideas for future research in Section 7.3 and concluding remarks in Section 7.4.

7.1 Contributions and Achievements

In this section, we provide a high-level overview of our presented solutions, outline the achieved results, and highlight their contribution to the main goal of this thesis. To recollect, the main goal of this thesis comprises of the two key aspects *security protocol efficiency* and *protocol security* as previously discussed in Section 1.1.

7.1.1 Tailored Protocol Handshake Mechanisms

With our first contribution, we address the identified protocol efficiency and security issues that stem from the high processing overhead of public-key cryptography. To this end, we developed three complementary protocol extensions for HIP DEX and showed how these extensions can similarly be applied to DTLS and Minimal IKEv2.

The *flexible session resumption mechanism* reduces the need for public-key cryptography throughout the lifetime of a networked embedded device. The main idea behind session resumption is for two communication end-points to only perform a public-key-based handshake during the initial connection establishment. The end-points then store the established session state across connections and leverage this state to efficiently re-establish the initial session context via subsequent session resumption handshakes, i.e., without the need for public-key cryptography. In addition, our session resumption mechanism also allows one handshake peer to securely offload its session state to its communication partner. This allows to relieve the offloading peer from the memory burden of storing session state across connections.

The long processing time of public-key operations on networked embedded devices enables even a single adversary to target these computationally expensive handshake operations in a DoS attack. To protect constrained devices against such attacks, we introduced a *collaborative puzzle-based DoS protection mechanism.* The main goal of this mechanism is to adapt the cryptographic puzzle approach of HIP DEX to the device constraints in the embedded domain and the resource asymmetry in the IoT. To this end, we developed an attack detection mechanism that allows the target device to only request a puzzle-based resource commitment from its handshake peer in case of a potential DoS attack. As a result, a legitimate handshake peer only has to invest its possibly scarce resources during an on-going attack. Moreover, we devised a collaborative difficulty selection strategy that enables the target device to adjust its requested resource commitment based on additional information from an on-path gateway. This allows the device to account for the resource asymmetry between networked embedded devices and conventional hosts or services in the IoT.

The high computation overhead of public-key cryptography in the context of networked embedded devices causes the processing time of the different handshake messages to vary significantly. To prevent premature retransmissions stemming from such variation in processing time, we introduced an *adaptive retransmission mechanism* that refrains from the use of fixed-length retransmission timeouts. Instead, our devised mechanism employs multiple dynamically adjustable worst-case estimates for message retransmission purposes. More precisely, messages that only trigger inexpensive handshake operations at the handshake peer are retransmitted based on a simple network delay-based timeout. Retransmissions of expensive handshake messages, in contrast, incorporate feedback from the handshake peer for an additionally processing-based timeout. This allows to largely prevent premature retransmissions that would otherwise be prevalent in the context of networked embedded devices.

The evaluation results show that our flexible session resumption mechanism reduces the computation overhead by up to 91.5 % and transmissions by up to 43.0 % compared to a standard protocol handshake. Moreover, our collaborative puzzle-based DoS protection mechanism accounts for the resource asymmetry in the IoT and successfully defends networked embedded devices against more powerful adversaries. Lastly, our adaptive retransmission mechanism handles packet loss at a moderate transmission overhead and affords a timely handshake conclusion. With these results, we conclude that our presented solutions successfully improve both aspects of our main goal in this thesis, i.e., *security protocol efficiency* and *protocol security.*

7.1.2 Message Wire-Format Compression

In our second contribution, we specifically focussed on the identified message conciseness issues of the HIP DEX protocol. For this purpose, we introduced a novel compression layer called Slimfit that adapts the HIP message wire-format for transmission inside the embedded domain. As its main building blocks, Slimfit (i) elides the message content that is statically defined in the HIP DEX protocol specification, (ii) rearranges the HIP message wire-format via general parameter compression mechanisms to achieve a higher compression ratio, and (iii) introduces compression profiles that allow to adapt the parameter-content-specific compression mechanisms to future signaling information. Notably, by designing Slimfit as a separate layer in the network stack, it can transparently be combined with an existing HIP DEX implementation. In addition, our Slimfit layer can also be deployed without HIP DEX at an interconnecting gateway. The gateway then acts a compressor and decompressor for traversing HIP DEX messages. As a result, communication end-points and on-path network elements that are located outside the embedded domain can remain oblivious to our compressed HIP message wire-format. This design trait affords incremental deployment of our Slimfit layer on a per-network basis.

The evaluation results show that our Slimfit layer achieves compression ratios between 1.36 and 1.55 depending on the handshake message and reduces the overall message fragmentation of the HIP DEX handshake by about 25 %. Moreover, Slimfit also decreases the number of retransmissions and even marginally *reduces* the handshake processing overhead. Notably, we identify a modest ROM overhead as the only tradeoff for the above overhead reductions. Based on these results, we conclude that Slimfit effectively contributes to the sub-goal *security protocol efficiency*.

7.1.3 Handshake Delegation Architecture

Our third contribution concerns the extensive RAM and ROM requirements that render a comprehensive, public-key-enabled end-to-end security protocol implementation infeasible for a wide range of memory-constrained embedded devices. To still enable such memory-constrained devices to communicate securely via standard end-to-end IP security protocols, we introduced the handshake delegation architecture for the DTLS protocol. Moreover, we showed that handshake delegation also affords a memory-efficient mode of operation in the context of HIP DEX and Minimal IKEv2.

The key idea behind the handshake delegation architecture is to offload the initial public-key-based connection establishment handshake to an off-path, trusted delegation server. The unconstrained nature of the delegation server then allows to use public-key cryptography for peer authentication and key agreement purposes when performing the connection establishment on behalf of a constrained device. Moreover, by subsequently handing over the established connection context to the constrained device via session resumption functionality, this device no longer has to implement expensive public-key cryptography. Instead, it can rely on an abbreviated session resumption handshake and efficient symmetric-key cryptography for the protection of application data. In addition to such lightweight key provisioning, handshake delegation also provides for an authorization framework in the context of embedded devices. To this end, a human operator leverages the central role of the delegation server during the initial protocol handshake to control which other communication end-points a networked embedded device is allowed to interact with.

The evaluation results show that our handshake delegation architecture achieves an overall RAM and ROM reduction of about 64 % compared to a certificate-based DTLS protocol implementation. Likewise, the handshake delegation procedure reduces the computation overhead by 97 % and the transmission overhead by 68 % in the context of constrained devices. With these result, our presented architecture achieves similarly low computation and transmission overheads as a purely symmetric-key-based DTLS handshake while providing additional authorization capabilities. Overall, we, therefore, conclude that handshake delegation provides a comprehensive, yet compact solution for authentication, authorization, and secure data transmission for the IP-based IoT. Hence, handshake delegation successfully addresses to the sub-goal *security protocol efficiency* and, regarding the provided authorization capabilities, also contributes to the sub-goal *protocol security*.

7.1.4 Secure 6LoWPAN Fragmentation

With our fourth and final contribution, we address the identified DoS attacks against the 6LoWPAN fragmentation mechanism, i.e., the fragment duplication and the buffer reservation attack. The fragment duplication attack enables an eavesdropping adversary to reactively block the reassembly of an overheard fragmented IPv6 packet by sending a single forged 6LoWPAN fragment to the target device. Similarly, the buffer reservation attack enables an adversary without overhearing capabilities to pro-actively block packet reassembly by maliciously reserving the 6LoWPAN reassembly buffer with a single 6LoWPAN fragment. To defend against these attacks, we introduced two complementary security mechanisms at the 6LoWPAN layer.

The *content-chaining scheme* protects networked embedded devices against the fragment duplication attack. To this end, content chaining affords per-fragment authentication by cryptographically binding the content of a fragmented IPv6 packet to its first 6LoWPAN fragment via an efficient hash-chain construction. This prevents the adversary from maliciously attributing spoofed 6LoWPAN fragments to a legitimate fragmented IPv6 packet. We complement this hash-chain construction with three overhead reduction techniques that allow to deactivate the content-chaining scheme during normal network operation and that reduce its computation and transmission overheads when protecting against an on-going attack in its active state.

The *split buffer approach* allows to mitigate the buffer reservation attack at a target device. As the name suggests, the split buffer approach segments the 6LoWPAN reassembly buffer into fragment-sized buffer slots. This segmentation then fosters per-fragment instead of per-packet competition for the scarce buffer resources at a target device between the adversary and legitimate fragment senders. In addition, the packet discard strategy for the split buffer approach severely penalizes suspicious sending behavior. In combination, this prevents an adversary from maliciously reserving the scarce reassembly resources at the 6LoWPAN layer of a target device.

The evaluation results confirm the practicability of the identified 6LoWPAN fragmentation attacks and show the effectiveness of our proposed defense mechanisms at moderate computation, energy, memory, and transmission trade-offs. Hence, we conclude that the above defense mechanisms effectively address the sub-goal *protocol security* with respect to the fragmentation facilities provided at 6LoWPAN adaptation layer. As such, our fourth contribution builds the foundation for our previous three contributions by securing the 6LoWPAN layer as the key underlying transport mechanism for DTLS, HIP DEX, and Minimal IKEv2 in the embedded domain.

7.2 Impact at the Time of Writing

We now provide a brief overview of the impact of our work at the time of writing. For the sake of clarity, we distinguish between academia, industry, and standardization.

Regarding the *academic impact*, we published all our core contributions at international scientific conferences [HHW$^+$13, HWZ$^+$13, HSR$^+$14] and workshops [HHW11, HRW12, HZS$^+$13, HHHW13]. As an indicator of the corresponding impact, we note that each of our publications has already been cited by several other researchers.

In addition, we presented parts of our work in the context of project presentations and invited talks from *industry*. For example, we recently held a keynote at a workshop of the *CPS.HUB/NRW*, a federal initiative to increase the competitive strength of the state of North Rhine-Westphalia in the field of cyber-physical systems [cpshub]. Here, we introduced the key technical enablers for IP technology in the IoT and discussed network security concerns related to our work in this thesis.

Moreover, we proposed selected solutions of our four core contributions for *standardization* at the IETF. These solutions include the flexible session resumption mechanism [HGS13], an early version of the collaborative puzzle-based DoS protection mechanism [HHH13], and the adaptive retransmission mechanism [MH14]. To further raise awareness of our work, we presented the flexible session resumption mechanism at the IETF 88 meeting of the DTLS In Constrained Environments (DICE) WG [DICE13]. Notably, the adaptive retransmission mechanism recently replaced the original aggressive retransmission strategy in the HIP DEX protocol specification due to its improved retransmission properties [MH14]. In this context, our Slimfit compression layer likewise was considered for adoption in the HIP DEX specification. Slimfit, however, was considered independent protocol functionality and we were encouraged to submit a separate document that specifies the devised compression scheme. We note that the write-up of this document is on-going work.

Besides the above standardization documents, we also contributed to current standardization efforts at the IETF in the context of the IoT via active participation in the corresponding working groups. For example, we presented an early version of the handshake delegation architecture at a pre-meeting of the Authentication and Authorization for Constrained Environments (ACE) WG [ACE14] with the goal to contribute to the considered problem space and to demonstrate a potential solution. Finally, our feedback regarding the DTLS profile for the IoT helped to improve the corresponding standardization document [TF15] in the context of DICE WG.

7.3 Future Research

Based on our contributions in this thesis and our gained understanding of end-to-end security in the IP-based IoT, we identified a number of future research challenges. We now briefly outline these challenges and sketch possible solutions.

7.3.1 Cross-Domain Device-to-Device Authorizations

With the handshake delegation architecture, we introduced a framework for intra- and inter-domain authorization in the IP-based IoT. As illustrated in Figure 7.1, we thereby considered the following two types of network interactions:

i) A networked embedded device exchanges information with another networked embedded device from the same administrative domain.

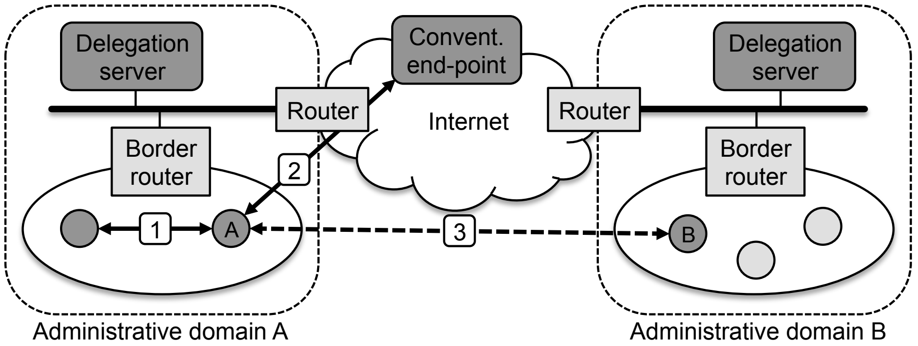


Figure 7.1 The presented handshake delegation architecture provides a framework for intra-domain authorizations between networked embedded devices (1) and inter-domain authorizations with conventional communication end-points (2). Authorizations involving memory-constrained embedded devices from remote administrative domains (3) are future work.

ii) A networked embedded device interacts with an unconstrained communication end-point such as a local workstation or a Cloud-based service.

These network interactions cover the majority of use cases that are currently considered within the context of the ACE WG at the IETF [SGS$^+$15]. Several of these use cases, however, also imply direct device-to-device communication across administrative domains. In one of the considered building automation scenarios, e.g., a smoke sensor from one building occupant is described to trigger the fire alarms of all other occupants. Our presented architecture currently only provides limited support for this type of interactions as handshake delegation either requires the remote end-point to support public-key cryptography or necessitates the delegation server to be in possession of the device-specific delegation keys of both embedded devices.

To address this limitation, we envision the following extension of our handshake delegation architecture. When authorizing a new connection between two networked embedded devices A and B from distinct administrative domains (see case 3 in Figure 7.1), the delegation server of domain A first establishes a connection with the delegation server from domain B. During this handshake, both delegation servers authenticate each other. In case of sufficient access rights, the delegation server of domain B then encrypts a session ticket for device B and transfers this session ticket to A's delegation server. This delegation server, in turn, forwards the received session ticket to device A along with its own session context. Devices A and B now can re-establish the connection based on the delegated session context as described in Chapter 5. We note that we consider the detailed design and a thorough security analysis of this extended handshake delegation architecture to be future work.

7.3.2 Selective End-to-End Payload Security for the IoT

Our contributions in this thesis primarily focus on protocol efficiency and security issues during the connection establishment phase of the DTLS, HIP DEX, and Minimal IKEv2 protocols in the context of the IoT. During our research, we also identified design-level issues with respect to the *protection of application data* that go beyond recent efforts to reduce the header overhead of payload security in the context of DTLS [RSH$^+$13, RSD14] and the IPsec protocol suite [GMSS10, RDC$^+$11,

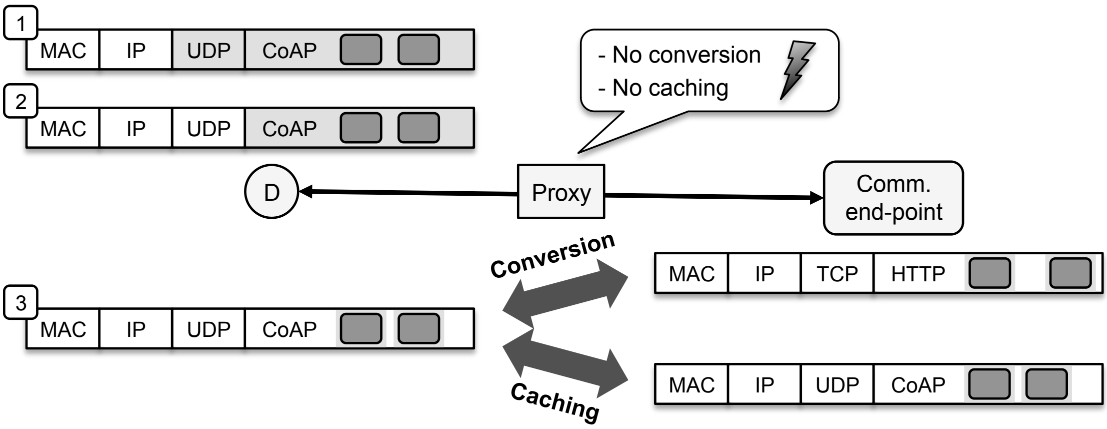


Figure 7.2 The network-layer and transport-layer protection of standard end-to-end IP security (marked in light gray for 1 and 2, respectively) prevent cross-protocol conversion middleboxes or caching proxies from operating on traversing network packets. In contrast, selective end-to-end payload security (3) only encrypts confidential and integrity-protects immutable application data (marked in dark gray) and, thus, affords provisioning of these on-path middlebox functions.

RVJ12, MG14]. Specifically, we observe that middleboxes often constitute important on-path auxiliaries in the IoT. *Cross-protocol conversion* between CoAP and HTTP at an intermediary proxy, e.g., is presumed to foster adoption of CoAP by affording incremental deployment of this new application layer protocol for existing HTTP-enabled hosts or services [CLR$^+$15]. Likewise, *caching proxies* enable access to information from sleeping devices and allow to reduce transmissions inside the embedded domain [SHB14]. Such additional on-path services commonly require access to traversing packets at the application layer. As illustrated in Figure 7.2, this requirement stands in direct conflict with the network-layer or transport-layer protection that is commonly provided by existing end-to-end IP security solutions.

With the goal to support cross-protocol conversion and caching proxies, we conducted initial research based on the idea of *selective* end-to-end payload security. More precisely, we investigated the integration of an object-based security mechanism such as JOSE [Bar14] with the HIP DEX protocol as a replacement of the original IPsec-protected payload channel. We note that we performed this initial research in collaboration with a student in the context of his Bachelor's thesis [Bog13].

The key idea behind the proposed approach is to extend the existing protocol design with two additional capabilities. First, a new negotiation mechanism enables the handshake peers to agree on the use of object-based payload security. Second, a novel API allows an application to leverage the established session context to selectively protect its data before transmission. More precisely, an application uses this new API to encrypt confidential information and to protect the integrity of immutable information. As shown in Figure 7.2, public mutable information remains unprotected. As a result of such selective payload protection, intermediary proxies can perform (restricted) cross-protocol conversions and are able to cache information from sleeping devices. In addition, we also envision caching support for groups of recipients. Here, the main challenge, however, is that all group members must be in possession of the same payload protection keys to be able to decrypt and verify the integrity of application data from a caching proxy. We identify the design of this group key mechanism and the integration of the proposed approach with DTLS and

Minimal IKEv2 as important future research. Still, we note that our initial results already strongly indicate the general feasibility of the proposed approach.

7.3.3 End-Point-Assisted In-Network Security

Our four core contributions in this thesis specifically concern network security mechanisms that are provided by the communication end-points themselves. In existing IP networks, this type of network security is commonly supplemented with in-network security solutions such as firewalls and intrusion detection systems. We expect middleboxes that provide such in-network security to similarly complement our considered end-to-end security solutions in the context of the IoT. Notably, one would expect such middleboxes to operate on detailed and precise information due to their critical role in the network. The IP network architecture and its supporting protocols, however, provide middleboxes with little more than IP addresses, port numbers, and protocol IDs as easily accessible information. Moreover, deep packet inspection typically only affords a probabilistic classification of data flows and commonly fails in case of encrypted application data. In-network security, therefore, regularly has to rely on ambiguous and forgeable information in order to identify communication end-points and applications as the origin of a data flow and to decide which flows to permit and which ones to block. This circumstance, however, impairs the effectivity of in-network security for the IoT in particular and for IP networks in general.

To tackle this issue of inadequate context information on the communication path, we previously proposed a novel end-to-middle signaling extension for the HIP protocol handshake, called SEAMS [HZH$^+$12]. This extension enables on-path middleboxes to actively participate in the handshake by requesting context information such as the device type or the employed application from the handshake peers. The handshake peers then look up the requested end-point contexts and signal this information within the handshake. As the signaled contexts are cryptographically bound to the handshake peers, middleboxes can verify and use the requested information to provide more secure and richer middlebox functions. Our SEAMS extension, however, heavily relies on the use of public-key cryptography and, thus, would cause excessive overheads in the context of networked embedded devices. We consider the development of a similar approach to SEAMS for the IoT interesting future work.

7.4 Concluding Remarks

The recent exposure of several high-profile cyber-attacks including pervasive monitoring, corporate data theft, and industrial sabotage emphasizes the fundamental need for comprehensive network security in an increasingly interconnected world. We expect this requirement to become even more pronounced in the IoT, where cyber-attacks are prone to have large-scale physical impact and to involve highly sensitive private information. At the same time, the device and network constraints in the embedded domain as well as the resource asymmetry in the IoT render the design of efficient and secure network security solutions especially challenging.

In this thesis, we showed that many of these emerging protocol design challenges can effectively be resolved via resource-conscious protocol extensions and additional architectural considerations. Moreover, we highlighted the fact that also seemingly unobtrusive IoT protocol adaptations, e.g., as provided by the 6LoWPAN layer, can have detrimental impact on the overall security properties of an otherwise secure networking solution. Hence, we re-iterate the need to consider network security as a *primary goal* in the design of new as well as in the adaptation of existing networking solutions. To conclude, it is our hope and strong belief that the protocol analyses and the security solutions presented in this thesis make a valuable contribution to the successful deployment of standard end-to-end IP security in the IoT. Finally, we hope that our work inspires future research in the field of IoT network security.

Glossary

6LoWPAN	IPv6 over Low-Power Wireless Personal Area Networks
AAA	Authentication, Authorization, and Accounting
ACE	Authentication and Authorization for Constrained Environments
ACL	Access Control List
AES	Advanced Encryption Standard
AH	Authentication Header
API	Application Programming Interface
Bluetooth LE	Bluetooth Low Energy
CA	Certificate Authority
CCM	Counter with CBC-MAC
CMAC	Cipher-based MAC
CPU	Central Processing Unit
CoAP	Constrained Application Protocol
DDoS	Distributed Denial-of-Service
DH	Diffie-Hellman
DICE	DTLS In Constrained Environments
DM	Davies-Meyer
DSA	Digital Signature Algorithm
DTLS	Datagram Transport Layer Security
DoS	Denial-of-Service
EAP	Extensible Authentication Protocol
ECC	Elliptic Curve Cryptography
ECDH	Elliptic Curve Diffie-Hellman
ECDSA	Elliptic Curve Digital Signature Algorithm
ECN	Explicit Congestion Notification
EID	Extension Header ID
ESP	Encapsulating Security Payload
GHC	Generic Header Compression
HIP	Host Identity Protocol
HIP DEX	Host Identity Protocol Diet EXchange

HIPv2	Host Identity Protocol Version 2
HIT	Host Identity Tag
HMAC	Hash-based Message Authentication Code
HTTP	Hypertext Transfer Protocol
ICMP	Internet Control Message Protocol
IDS	Intrusion Detection System
IETF	Internet Engineering Task Force
IKEv2	Internet Key Exchange Protocol Version 2
IP	Internet Protocol
IPComp	IP Payload Compression Protocol
IPSecME	IP Security Maintenance and Extensions
IPsec	Internet Protocol Security
IPv4	Internet Protocol version 4
IPv6	Internet Protocol version 6
ISP	Internet Service Provider
IoT	Internet of Things
JOSE	JSON Object Signing and Encryption
KDC	Key Distribution Center
LAN	Local Area Network
M2M	machine-to-machine
MAC	Message Authentication Code
MCU	microcontroller
MMO	Matyas-Meyer-Oseas
MTU	Maximum Transmission Unit
Minimal IKEv2	Minimal Internet Key Exchange Protocol Version 2
ND	Neighbor Discovery
NHC	Next Header Compression
NIST	National Institute of Standards and Technology
ORCHID	Overlay Routable Cryptographic Hash IDentifier
PANA	Protocol for Carrying Authentication for Network Access
PDR	Packet Delivery Ratio
PKI	Public-Key Infrastructure
PSK	Pre-Shared Key
ROHC	RObust Header Compression
RPL	IPv6 Routing Protocol for Low-Power and Lossy Networks
RSA	Ron Rivest, Adi Shamir, and Leonard Adleman
RSSI	Received Signal Strength Indication
RTCP	RTP Control Protocol
RTT	Round-Trip Time

SCVP	Server-based Certificate Validation Protocol
SHA-1	Secure Hash Standard version 1
SHA-2	Secure Hash Standard version 2
SoC	System on a Chip
TCP	Transmission Control Protocol
TLS	Transport Layer Security
TLV	Type-Length-Value
TPM	Trusted Platform Module
UDP	User Datagram Protocol
URI	Uniform Resource Identifier
VLAN	Virtual LAN
VPN	Virtual Private Network
WSN	Wireless Sensor Network
XMPP	Extensible Messaging and Presence Protocol

Bibliography

[6lowpan] Internet Engineering Task Force. IPv6 over IEEE 802.15.4 BOF (6lowpan). [Online @ `http://www.ietf.org/meeting/62/session/6lowpan.txt` [Last accessed: 16.06.2015].

[ABV$^+$04] B. Aboba, L. Blunk, J. Vollbrecht, J. Carlson, and H. Levkowetz. Extensible Authentication Protocol (EAP). RFC 3748 (Proposed Standard), Internet Engineering Task Force, June 2004.

[ACE14] Authentication and Authorization for Constrained Environments (ace) WG. Charter charter-ietf-ace-01 (Approved), Internet Engineering Task Force, June 2014.

[AIM10] L. Atzori, A. Iera, and G. Morabito. The Internet of Things: A survey. *Computer Networks*, 54(15):2787–2805, 2010.

[AN97] T. Aura and P. Nikander. Stateless Connections. In *Proceedings of the International Conference on Information and Communications Security*, ICICS '97, volume 1334 of *Lecture Notes in Computer Science*, pages 87–97. Springer, 1997.

[ANL01] T. Aura, P. Nikander, and J. Leiwo. DOS-Resistant Authentication with Client Puzzles. In B. Christianson, J. Malcolm, B. Crispo, and M. Roe, editors, *Security Protocols*, volume 2133 of *Lecture Notes in Computer Science*, pages 170–177. Springer Berlin Heidelberg, 2001.

[APW09] M. Albrecht, K. Paterson, and G. Watson. Plaintext Recovery Attacks against SSH. In *Proceedings of the 30th IEEE Symposium on Security and Privacy*, S&P '09, pages 16–26, May 2009.

[Bar87] P. Barrett. Implementing the Rivest Shamir and Adleman Public Key Encryption Algorithm on a Standard Digital Signal Processor. In A. Odlyzko, editor, *Advances in Cryptology – CRYPTO '86*, volume 263 of *Lecture Notes in Computer Science*, pages 311–323. Springer Berlin Heidelberg, 1987.

[Bar14] R. Barnes. Use Cases and Requirements for JSON Object Signing and Encryption (JOSE). RFC 7165 (Informational), Internet Engineering Task Force, April 2014.

[BBD06] A. Becher, Z. Benenson, and M. Dornseif. Tampering with Motes: Real-World Physical Attacks on Wireless Sensor Networks. In

J. Clark, R. Paige, F. Polack, and P. Brooke, editors, *Security in Pervasive Computing*, volume 3934 of *Lecture Notes in Computer Science*, pages 104–118. Springer Berlin Heidelberg, 2006.

[BBL⁺12] R. Bonetto, N. Bui, V. Lakkundi, A. Olivereau, A. Serbanati, and M. Rossi. Secure Communication for Smart IoT Objects: Protocol Stacks, Use Cases and Practical Examples. In *Proceeding of the IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks*, WoWMoM '12, pages 1–7, June 2012.

[BCK96] M. Bellare, R. Canetti, and H. Krawczyk. Keying hash functions for message authentication. In N. Koblitz, editor, *Advances in Cryptology – CRYPTO '96*, volume 1109 of *Lecture Notes in Computer Science*, pages 1–15. Springer Berlin Heidelberg, 1996.

[BDHS⁺11] A. Bartoli, M. Dohler, J. Hernández-Serrano, A. Kountouris, and D. Barthel. Low-Power Low-Rate Goes Long-Range: The Case for Secure and Cooperative Machine-to-Machine Communications. In V. Casares-Giner, P. Manzoni, and A. Pont, editors, *NETWORKING 2011 Workshops*, volume 6827 of *Lecture Notes in Computer Science*, pages 219–230. Springer Berlin Heidelberg, 2011.

[BEK14] C. Bormann, M. Ersue, and A. Keranen. Terminology for Constrained-Node Networks. RFC 7228 (Informational), Internet Engineering Task Force, May 2014.

[BG06] P. G. Bradford and O. V. Gavrylyako. Hash chains with diminishing ranges for sensors. *International Journal of High Performance Computing and Networking*, 4(1):31–38, 2006.

[BH05] S. Bellovin and R. Housley. Guidelines for Cryptographic Key Management. RFC 4107 (Best Current Practice), Internet Engineering Task Force, June 2005.

[BHL06] A. Bittau, M. Handley, and J. Lackey. The final nail in WEP's coffin. In *Proceedings of the IEEE Symposium on Security and Privacy*, S&P '06, pages 386–400, 2006.

[BN07] P. Boyle and T. Newe. Security Protocols for Use with Wireless Sensor Networks: A Survey of Security Architectures. In *Proceeding of the 3rd International Conference on Wireless and Mobile Communications*, ICWMC '07, pages 54–54, 2007.

[Boe11] T. Boettcher. Analysis and mitigation of fragmentation attacks against the 6LoWPAN adaptation layer. Diploma thesis, RWTH Aachen University, November 2011.

[Bog13] M. Bogner. Selective Payload Security in the Internet of Things. Bachelor's thesis, RWTH Aachen University, August 2013.

[Bor13] C. Bormann. Guidance for Light-Weight Implementations of the Internet Protocol Suite. Internet-Draft draft-ietf-lwig-guidance-03 (Working Draft), Internet Engineering Task Force, February 2013.

[Bor14] C. Bormann. 6LoWPAN-GHC: Generic Header Compression for IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs). RFC 7400 (Proposed Standard), Internet Engineering Task Force, November 2014.

[BÖS11] J. Bos, O. Özen, and M. Stam. Efficient Hashing Using the AES Instruction Set. In B. Preneel and T. Takagi, editors, *Cryptographic Hardware and Embedded Systems*, CHES '11, volume 6917 of *Lecture Notes in Computer Science*, pages 507–522. Springer Berlin Heidelberg, 2011.

[Bra89] R. Braden. Requirements for Internet Hosts - Communication Layers. RFC 1122 (Internet Standard), Internet Engineering Task Force, October 1989.

[BT13] Bluetooth Specification 4.1, Bluetooth Special Interest Group Core Specification 4.1, December 2013.

[cc2420] Texas Instruments. CC2420 Datasheet. [Online @ `http://www.ti.com/lit/ds/symlink/cc2420.pdf` [Last accessed: 16.06.2015].

[cc2520] Texas Instruments. CC2520 Datasheet. Online @ `http://www.ti.com/lit/ds/symlink/cc2520.pdf` [Last accessed: 16.06.2015].

[CERT96] CERT. Advisory CA-1996-26: Denial-of-Service Attack via ping, 1996.

[CERT97] CERT. Advisory CA-1997-28: IP Denial-of-Service Attacks, 1997.

[CLR$^+$15] A. Castellani, S. Loreto, A. Rahman, T. Fossati, and E. Dijk. Guidelines for HTTP-CoAP Mapping Implementations. Internet-Draft draft-ietf-core-http-mapping-06 (Working Draft), Internet Engineering Task Force, March 2015.

[CPG$^+$15] S. Cirani, M. Picone, P. Gonizzi, L. Veltri, and G. Ferrari. IoT-OAS: An OAuth-Based Authorization Service Architecture for Secure Services in IoT Scenarios. *IEEE Sensors Journal*, 15(2):1224–1234, 2015.

[cpshub] CPS.HUB/NRW. Online @ `http://cps-hub-nrw.de/` [Last accessed: 16.06.2015].

[CSF$^+$08] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, and W. Polk. Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. RFC 5280 (Proposed Standard), Internet Engineering Task Force, May 2008.

[CWK97] S. Chan, E. Wong, and K. Ko. Fair packet discarding for controlling ABR traffic in ATM networks. *IEEE Transactions on Communications*, 45(8):913–916, 1997.

[CY07] S. Camtepe and B. Yener. Combinatorial Design of Key Distribution Mechanisms for Wireless Sensor Networks. *IEEE/ACM Transactions on Networking*, 15(2):346–358, 2007.

[CZ12] M. Campagna and G. Zaverucha. A Cryptographic Suite for Embedded Systems (SuiteE). Internet-Draft draft-campagna-suitee-04 (Working Draft), Internet Engineering Task Force, October 2012.

[Dam90] I. B. Damgård. A Design Principle for Hash Functions. In G. Brassard, editor, *Proceedings of Advances in Cryptology – CRYPTO '89*, volume 435 of *Lecture Notes in Computer Science*, pages 416–427. Springer New York, 1990.

[DC14] M. Doddavenkatappa and M. C. Chan. P3: A Practical Packet Pipeline using synchronous transmissions for wireless sensor networks. In *Proceedings of the 13th International Symposium on Information Processing in Sensor Networks*, IPSN '14, pages 203–214, 2014.

[DEFT11] A. Dunkels, J. Eriksson, N. Finne, and N. Tsiftes. Powertrace: Network-Level Power Profiling for Low-Power Wireless Networks. Technical report, Swedish Institute of Computer Science, 2011.

[Deu96] P. Deutsch. DEFLATE Compressed Data Format Specification version 1.3. RFC 1951 (Informational), Internet Engineering Task Force, May 1996.

[DG96] P. Deutsch and J.-L. Gailly. ZLIB Compressed Data Format Specification version 3.3. RFC 1950 (Informational), Internet Engineering Task Force, May 1996.

[DGV04] A. Dunkels, B. Gronvall, and T. Voigt. Contiki - A Lightweight and Flexible Operating System for Tiny Networked Sensors. In *Proceedings of the 29th Annual IEEE International Conference on Local Computer Networks*, LCN '04, pages 455–462, 2004.

[DH76] W. Diffie and M. E. Hellman. New Directions in Cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.

[DH98] S. Deering and R. Hinden. Internet Protocol, Version 6 (IPv6) Specification. RFC 2460 (Draft Standard), Internet Engineering Task Force, December 1998.

[DHJS00] M. Degermark, H. Hannu, L. Jonsson, and K. Svanbro. Evaluation of CRTP performance over cellular radio links. *IEEE Personal Communications*, 7(4):20–25, 2000.

[DICE13] DTLS In Constrained Environments (dice) WG. Charter charter-ietf-dice-01 (Approved), Internet Engineering Task Force, September 2013.

[DK07] H. Delfs and H. Knebl. *Introduction to Cryptography: Principles and Applications.* Information Security and Cryptography. Springer Berlin Heidelberg, 2007.

[DKS07] E. Davies, S. Krishnan, and P. Savola. IPv6 Transition/Co-existence Security Considerations. RFC 4942 (Informational), Internet Engineering Task Force, September 2007.

[DN93] C. Dwork and M. Naor. Pricing via Processing or Combatting Junk Mail. In E. Brickell, editor, *Advances in Cryptology – CRYPTO '92*, volume 740 of *Lecture Notes in Computer Science*, pages 139–147. Springer Berlin Heidelberg, 1993.

[DOD11] S. Duquennoy, F. Österlind, and A. Dunkels. Lossy links, low power, high throughput. In *Proceedings of the 9th ACM Conference on Embedded Networked Sensor Systems*, SenSys '11, pages 12–25, 2011.

[DR08] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246 (Proposed Standard), Internet Engineering Task Force, August 2008.

[DS01] D. Dean and A. Stubblefield. Using Client Puzzles to Protect TLS. In *Proceedings of the 10th Conference on USENIX Security Symposium*, SSYM '01, 2001.

[Dun03] A. Dunkels. Full TCP/IP for 8-bit Architectures. In *Proceedings of the 1st International Conference on Mobile Systems, Applications and Services*, MobiSys '03, pages 85–98, 2003.

[dVPC+08] G. V. de Velde, C. Popoviciu, T. Chown, O. Bonness, and C. Hahn. IPv6 Unicast Address Assignment Considerations. RFC 5375 (Informational), Internet Engineering Task Force, December 2008.

[EG02] L. Eschenauer and V. D. Gligor. A Key-management Scheme for Distributed Sensor Networks. In *Proceedings of the 9th ACM Conference on Computer and Communications Security*, CCS '02, pages 41–47, 2002.

[ET05] P. Eronen and H. Tschofenig. Pre-Shared Key Ciphersuites for Transport Layer Security (TLS). RFC 4279 (Proposed Standard), Internet Engineering Task Force, December 2005.

[FALZ12] V. Fajardo, J. Arkko, J. Loughney, and G. Zorn. Diameter Base Protocol. RFC 6733 (Proposed Standard), Internet Engineering Task Force, October 2012.

[FHM+07] T. Freeman, R. Housley, A. Malpani, D. Cooper, and W. Polk. Server-Based Certificate Validation Protocol (SCVP). RFC 5055 (Proposed Standard), Internet Engineering Task Force, December 2007.

[FMMA06] S. Fouladgar, B. Mainaud, K. Masmoudi, and H. Afifi. Tiny 3-TLS: A Trust Delegation Protocol for Wireless Sensor Networks. In *Proceedings of the 3rd European Conference on Security and Privacy in Ad-Hoc and Sensor Networks*, ESAS '06, pages 32–42, 2006.

[FOP+08] D. Forsberg, Y. Ohba, B. Patil, H. Tschofenig, and A. Yegin. Protocol for Carrying Authentication for Network Access (PANA). RFC 5191 (Proposed Standard), Internet Engineering Task Force, May 2008.

[GBB14] S. Gerdes, O. Bergmann, and C. Bormann. Delegated Authenticated Authorization for Constrained Environments. In *Proceedings of the 22nd IEEE International Conference on Network Protocols*, ICNP '14, pages 654–659, 2014.

[GBB15] S. Gerdes, O. Bergmann, and C. Bormann. Delegated CoAP Authentication and Authorization Framework (DCAF). Internet-Draft draft-gerdes-ace-dcaf-authorize-02 (Working Draft), Internet Engineering Task Force, March 2015.

[GH11] Y. Gilad and A. Herzberg. Fragmentation Considered Vulnerable: Blindly Intercepting and Discarding Fragments. In *Proceedings of the USENIX Workshop On Offensive Technologies*, WOOT '11, 2011.

[GH13] Y. Gilad and A. Herzberg. Fragmentation Considered Vulnerable. *ACM Transactions on Information and System Security*, 15(4):16:1–16:31, 2013.

[GM01] P. Golle and N. Modadugu. Authenticating Streamed Data in the Presence of Random Packet Loss (Extended Abstract). In *Proceedings of the Network and Distributed System Security Symposium*, NDSS '01, pages 13–22. Internet Society, 2001.

[GMF$^+$05] V. Gupta, M. Millard, S. Fung, Y. Zhu, N. Gura, H. Eberle, and S. Shantz. Sizzle: A Standards-based end-to-end Security Architecture for the Embedded Internet. In *Proceedings of the 3rd IEEE International Conference on Pervasive Computing and Communications*, PerCom '05, pages 247–256, 2005.

[GMKK$^+$13a] O. Garcia-Morchon, S. Kumar, S. Keoh, R. Hummen, and R. Struik. Security Considerations in the IP-based Internet of Things. Internet-Draft draft-garcia-core-security-06 (Working Draft), Internet Engineering Task Force, September 2013.

[GMKK$^+$13b] O. Garcia-Morchon, S. L. Keoh, S. Kumar, P. Moreno-Sanchez, F. Vidal-Meca, and J. H. Ziegeldorf. Securing the IP-based Internet of Things with HIP and DTLS. In *Proceedings of the 6th ACM Conference on Security and Privacy in Wireless and Mobile Networks*, WiSec '13, pages 119–124, 2013.

[GMSS10] J. Granjal, E. Monteiro, and J. Sa Silva. Enabling Network-Layer Security on IPv6 Wireless Sensor Networks. In *Proceedings of the IEEE Global Telecommunications Conference*, GLOBECOM '10, pages 1–6, 2010.

[GMSS13] J. Granjal, E. Monteiro, and J. Sa Silva. End-to-end transport-layer security for Internet-integrated sensing applications with mutual and delegated ECC public-key authentication. In *Proceedings of the IFIP Networking Conference*, pages 1–9, 2013.

[GMW10] O. Garcia-Morchon and K. Wehrle. Modular Context-aware Access Control for Medical Sensor Networks. In *Proceedings of the 15th ACM*

Symposium on Access Control Models and Technologies, SACMAT '10, pages 129–138, 2010.

[Gon11] F. Gont. Security Assessment of the Internet Protocol Version 4. RFC 6274 (Informational), Internet Engineering Task Force, July 2011.

[GR97] R. Gennaro and P. Rohatgi. How to sign digital streams. In J. Kaliski, BurtonS., editor, *Advances in Cryptology – CRYPTO '97*, volume 1294 of *Lecture Notes in Computer Science*, pages 180–197. Springer Berlin Heidelberg, 1997.

[Har12] D. Hardt. The OAuth 2.0 Authorization Framework. RFC 6749 (Proposed Standard), Internet Engineering Task Force, October 2012.

[Har14a] K. Hartke. Practical Issues with Datagram Transport Layer Security in Constrained Environments. Internet-Draft draft-hartke-dice-practical-issues-01 (Working Draft), Internet Engineering Task Force, April 2014.

[Har14b] K. Hartke. Observing Resources in CoAP. Internet-Draft draft-ietf-core-observe-16 (Working Draft), Internet Engineering Task Force, December 2014.

[HBH$^+$05] C. Hartung, J. Balasalle, R. Han, C. Hartung, J. Balasalle, and R. Han. Node Compromise in Sensor Networks: The Need for Secure Systems. Technical report, University of Colorado at Boulder, 2005.

[HC02] J. Hill and D. Culler. Mica: A Wireless Platform for Deeply Embedded Networks. *IEEE Micro*, 22(6):12–24, 2002.

[HC08] J. W. Hui and D. E. Culler. IP is Dead, Long Live IP for Wireless Sensor Networks. In *Proceedings of the 6th ACM Conference on Embedded Network Sensor Systems*, SenSys '08, pages 15–28, 2008.

[HCSO09] W. Hu, P. Corke, W. Shih, and L. Overs. secFleck: A Public Key Technology Platform for Wireless Sensor Networks. In U. Roedig and C. Sreenan, editors, *Wireless Sensor Networks*, volume 5432 of *Lecture Notes in Computer Science*, pages 296–311. Springer Berlin Heidelberg, 2009.

[Hee11] T. M. Heer. *Direct End-to-Middle Authentication in Cooperative Networks.* PhD thesis, RWTH Aachen University, December 2011.

[HGMH$^+$11] T. Heer, O. Garcia-Morchon, R. Hummen, S. L. Keoh, S. S. Kumar, and K. Wehrle. Security Challenges in the IP-based Internet of Things. *Wireless Personal Communications*, 61(3):527–542, 2011.

[HGS13] R. Hummen, J. Gilger, and H. Shafagh. Extended DTLS Session Resumption for Constrained Network Environments. Internet-Draft draft-hummen-dtls-extended-session-resumption-01 (Working Draft), Internet Engineering Task Force, October 2013.

[HHCW12] R. Hummen, M. Henze, D. Catrein, and K. Wehrle. A Cloud Design for User-controlled Storage and Processing of Sensor Data. In *Proceedings of the 4th IEEE International Conference on Cloud Computing Technology and Science*, CloudCom '12, pages 232–240, 2012.

[HHH13] R. Hummen, M. Henze, and J. Hiller. HIP Middlebox Puzzle Offloading and End-host Notification. Internet-Draft draft-hummen-hip-middle-puzzle-01 (Working Draft), Internet Engineering Task Force, January 2013.

[HHHW13] R. Hummen, J. Hiller, M. Henze, and K. Wehrle. Slimfit – A HIP DEX Compression Layer for the IP-based Internet of Things. In *Proceedings of the 9th IEEE International Conference on Wireless and Mobile Computing, Networking and Communications*, WiMob '13, pages 259–266, 2013.

[HHK+09] T. Heer, R. Hummen, M. Komu, S. Götz, and K. Wehrle. End-Host Authentication and Authorization for Middleboxes Based on a Cryptographic Namespace. In *Proceedings of the IEEE International Conference on Communications*, ICC '09, pages 1–6, 2009.

[HHW11] R. Hummen, T. Heer, and K. Wehrle. A Security Protocol Adaptation Layer for the IP-based Internet of Things (Position Paper). IAB Workshop on Interconnecting Smart Objects with the Internet, 2011.

[HHW+13] R. Hummen, J. Hiller, H. Wirtz, M. Henze, H. Shafagh, and K. Wehrle. 6LoWPAN Fragmentation Attacks and Mitigation Mechanisms. In *Proceedings of the 6th ACM Conference on Security and Privacy in Wireless and Mobile Networks*, WiSec '13, pages 55–66, 2013.

[Hil12] J. Hiller. Implementation of the HIP Diet EXchange and Optimizations for Smart Object Networks. Bachelor's thesis, RWTH Aachen University, September 2012.

[HMC07] J. Heffner, M. Mathis, and B. Chandler. IPv4 Reassembly Errors at High Data Rates. RFC 4963 (Informational), Internet Engineering Task Force, July 2007.

[HMV04] D. Hankerson, A. J. Menezes, and S. Vanstone. *Guide to Elliptic Curve Cryptography*. Springer Professional Computing. Springer New York, 2004.

[HNK08] T. Henderson, P. Nikander, and M. Komu. Using the Host Identity Protocol with Legacy Applications. RFC 5338 (Experimental), Internet Engineering Task Force, September 2008.

[homematic] eQ-3 AG. HomeMatic. Online @ `http://www.homematic.com/` [Last accessed: 16.06.2015].

[HRPJ+15] J. Hernandez Ramos, M. Pawlowski, A. Jara, A. Skarmeta Gomez, and L. Ladid. Towards a lightweight authentication and authorization framework for smart objects. *IEEE Journal on Selected Areas in Communications*, PP(99):1–1, 2015.

[HRW12] R. Hummen, C. Röller, and K. Wehrle. Modeling User-defined Trust Overlays for the IP-based Internet of Things. IAB Workshop on Smart Object Security, 2012.

[HS13] A. Herzberg and H. Shulman. Fragmentation Considered Poisonous, or: One-domain-to-rule-them-all.org. In *Proceedings of the IEEE Conference on Communications and Network Security*, CNS '13, pages 224–232, 2013.

[HSR+14] R. Hummen, H. Shafagh, S. Raza, T. Voigt, and K. Wehrle. Delegation-based Authentication and Authorization for the IP-based Internet of Things. In *Proceedings of the 11th IEEE International Conference on Sensor, Communication, and Networking*, SECON '14, 2014.

[HT11] J. Hui and P. Thubert. Compression Format for IPv6 Datagrams over IEEE 802.15.4-Based Networks. RFC 6282 (Proposed Standard), Internet Engineering Task Force, September 2011.

[HVA15] T. Henderson, C. Vogt, and J. Arkko. Host Mobility with the Host Identity Protocol. Internet-Draft draft-ietf-hip-rfc5206-bis-08 (Working Draft), Internet Engineering Task Force, January 2015.

[HWZ+13] R. Hummen, H. Wirtz, J. Ziegeldorf, J. Hiller, and K. Wehrle. Tailoring End-to-End IP Security Protocols to the Internet of Things. In *Proceedings of the 21st IEEE International Conference on Network Protocols*, ICNP '13, pages 1–10, 2013.

[HZH+12] R. Hummen, J. Ziegeldorf, T. Heer, H. Wirtz, and K. Wehrle. SEAMS: A Signaling Layer for End-Host-Assisted Middlebox Services. In *Proceedings of the 11th IEEE International Conference on Trust, Security and Privacy in Computing and Communications*, TrustCom '12, pages 525–532, 2012.

[HZS+13] R. Hummen, J. H. Ziegeldorf, H. Shafagh, S. Raza, and K. Wehrle. Towards Viable Certificate-based Authentication for the Internet of Things. In *Proceedings of the 2nd ACM Workshop on Hot Topics on Wireless Network Security and Privacy*, HotWiSec '13, pages 37–42, 2013.

[IEEE03] IEEE Standard for Information technology – Telecommunications and information exchange between systems – Local and metropolitan area networks – Specific requirements, Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (LR-WPANs), Institute of Electrical and Electronics Engineers Standard 802.15.4, Revision 2003, October 2003.

[IEEE06] IEEE Standard for Information technology – Telecommunications and information exchange between systems – Local and metropolitan area networks – Specific requirements, Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (WPANs), Institute of Electrical and Electronics Engineers Standard 802.15.4, Revision 2006, September 2006.

[IEEE11a] IEEE Standard for Local and Metropolitan Area Networks – Media Access Control (MAC) Bridges and Virtual Bridge Local Area Networks, Institute of Electrical and Electronics Engineers Standard 802.1Q, Revision 2011, August 2011.

[IEEE11b] IEEE Standard for Local and metropolitan area networks, Part 15.4: Low-Rate Wireless Personal Area Networks (LR-WPANs), Institute of Electrical and Electronics Engineers Standard 802.15.4, Revision 2011, September 2011.

[IEEE12] IEEE Standard for Information technology – Telecommunications and information exchange between systems Local and metropolitan area networks – Specific requirements, Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications, Institute of Electrical and Electronics Engineers Standard 802.11, Revision 2012, March 2012.

[Ins15] The McClean Report - A Complete Analysis and Forecast of the Integrated Circuit Industry. Technical report, IC Insights, Inc., 2015.

[ISA11] Wireless Systems for Industrial Automation: Process Control and Related Applications, International Society of Automation ANSI/ISA 100.11a, Revision 2011, 2011.

[Jac88] V. Jacobson. Congestion Avoidance and Control. In *Symposium Proceedings on Communications Architectures and Protocols*, SIGCOMM '88, pages 314–329, 1988.

[JB99] A. Juels and J. G. Brainard. Client Puzzles: A Cryptographic Countermeasure Against Connection Depletion Attacks. In *Proceedings of the Network and Distributed System Security Symposium*, NDSS '99, pages 151–165. Internet Society, 1999.

[JMM15] P. Jokela, R. Moskowitz, and J. Melen. Using the Encapsulating Security Payload (ESP) Transport Format with the Host Identity Protocol (HIP). RFC 7402 (Proposed Standard), April 2015.

[KBC97] H. Krawczyk, M. Bellare, and R. Canetti. HMAC: Keyed-Hashing for Message Authentication. RFC 2104 (Informational), Internet Engineering Task Force, February 1997.

[Kel02] J. Kelsey. Compression and Information Leakage of Plaintext. In *Revised Papers from the 9th International Workshop on Fast Software Encryption*, FSE '02, pages 263–276, 2002.

[KHN$^+$14] C. Kaufman, P. Hoffman, Y. Nir, P. Eronen, and T. Kivinen. Internet Key Exchange Protocol Version 2 (IKEv2). RFC 7296 (Internet Standard), Internet Engineering Task Force, October 2014.

[Kim08] H. Kim. Protection Against Packet Fragmentation Attacks at 6LoWPAN Adaptation Layer. In *Proceedings of the International Conference on Convergence and Hybrid Information Technology*, ICHIT '08, pages 796–801, 2008.

[Kiv15] T. Kivinen. Minimal IKEv2. Internet-Draft draft-ietf-lwig-ikev2-minimal-02 (Working Draft), Internet Engineering Task Force, March 2015.

[KKG10] A. Khurri, D. Kuptsov, and A. Gurtov. On Application of Host Identity Protocol in Wireless Sensor Networks. In *Proceedings of 7th IEEE International Conference on Mobile Adhoc and Sensor Systems*, MASS '10, pages 358–345, 2010.

[KKR$^+$12] J. Ko, K. Klues, C. Richter, W. Hofer, B. Kusy, M. Bruenig, T. Schmid, Q. Wang, P. Dutta, and A. Terzis. Low Power or High Performance? A Tradeoff Whose Time Has Come (and Nearly Gone). In G. Picco and W. Heinzelman, editors, *Wireless Sensor Networks*, volume 7158 of *Lecture Notes in Computer Science*, pages 98–114. Springer Berlin Heidelberg, 2012.

[KKV12] E. Kim, D. Kaspar, and J. Vasseur. Design and Application Spaces for IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs). RFC 6568 (Informational), Internet Engineering Task Force, April 2012.

[KLNP07] C. Kuo, M. Luk, R. Negi, and A. Perrig. Message-In-a-Bottle: User-friendly and Secure Key Deployment for Sensor Nodes. In *Proceedings of the 5th International Conference on Embedded Networked Sensor Systems*, SenSys '07, pages 233–246, 2007.

[Kob87] N. Koblitz. Elliptic Curve Cryptosystems. *Mathematics of Computation*, 48(177):203–209, 1987.

[KPS03] C. Kaufman, R. Perlman, and B. Sommerfeld. DoS Protection for UDP-based Protocols. In *Proceedings of the 10th ACM Conference on Computer and Communications Security*, CCS '03, pages 2–7, 2003.

[KR12] J. F. Kurose and K. W. Ross. *Computer Networking: A Top-Down Approach.* Addison-Wesley, 6th edition, 2012.

[Kra05] H. Krawczyk. HMQV: A High-Performance Secure Diffie-Hellman Protocol. In V. Shoup, editor, *Advances in Cryptology – CRYPTO '05*, volume 3621 of *Lecture Notes in Computer Science*, pages 546–566. Springer Berlin Heidelberg, 2005.

[Kri09] S. Krishnan. Handling of Overlapping IPv6 Fragments. RFC 5722 (Proposed Standard), Internet Engineering Task Force, December 2009.

[KS05] J. Kelsey and B. Schneier. Second Preimages on n-Bit Hash Functions for Much Less than 2^n Work. In R. Cramer, editor, *Advances in Cryptology – EUROCRYPT '05*, volume 3494 of *Lecture Notes in Computer Science*, pages 474–490. Springer Berlin Heidelberg, 2005.

[KSH$^+$12] T. Kothmayr, C. Schmitt, W. Hu, M. Brunig, and G. Carle. A DTLS Based End-To-End Security Architecture for the Internet of Things with Two-Way Authentication. In *Proceedings of the 37th IEEE Conference on Local Computer Networks Workshops*, LCN Workshops '12, pages 956–963, 2012.

[KSH$^+$13] T. Kothmayr, C. Schmitt, W. Hu, M. Brünig, and G. Carle. DTLS based Security and Two-Way Authentication for the Internet of Things. *Ad Hoc Networks*, 11(8):2710 – 2723, 2013.

[KWK$^+$12] S. Krishnan, J. Woodyatt, E. Kline, J. Hoagland, and M. Bhatia. A Uniform Format for IPv6 Extension Headers. RFC 6564 (Proposed Standard), Internet Engineering Task Force, April 2012.

[Lam81] L. Lamport. Password Authentication with Insecure Communication. *Communications of the ACM*, 24(11):770–772, 1981.

[Lan10] A. Langley. Transport Layer Security (TLS) Snap Start. Internet-Draft draft-agl-tls-snapstart-00 (Working Draft), Internet Engineering Task Force, June 2010.

[Lan11] R. Langner. Stuxnet: Dissecting a Cyberwarfare Weapon. *IEEE Security & Privacy*, 9(3):49–51, 2011.

[LCC11] A. Ludovici, A. Calveras, and J. Casademont. Forwarding Techniques for IP Fragmented Packets in a Real 6LoWPAN Network. *Sensors*, 11(1):992–1008, 2011.

[LD14] J. Laganier and F. Dupont. An IPv6 Prefix for Overlay Routable Cryptographic Hash Identifiers Version 2 (ORCHIDv2). RFC 7343 (Proposed Standard), Internet Engineering Task Force, September 2014.

[LN08] A. Liu and P. Ning. TinyECC: A Configurable Library for Elliptic Curve Cryptography in Wireless Sensor Networks. In *Proceedings of the International Conference on Information Processing in Sensor Networks*, IPSN '08, pages 245–256, 2008.

[LRA09] J. Lopez, R. Roman, and C. Alcaraz. Analysis of Security Threats, Requirements, Technologies and Standards in Wireless Sensor Networks. In A. Aldini, G. Barthe, and R. Gorrieri, editors, *Foundations of Security Analysis and Design V*, volume 5705 of *Lecture Notes in Computer Science*, pages 289–338. Springer Berlin Heidelberg, 2009.

[MB12] D. McGrew and D. Bailey. AES-CCM Cipher Suites for Transport Layer Security (TLS). RFC 6655 (Proposed Standard), Internet Engineering Task Force, July 2012.

[MBC^{+}01] R. Min, M. Bhardwaj, S. Cho, E. Shih, A. Sinha, A. Wang, and A. Chandrakasan. Low-power Wireless Sensor Networks. In *Proceedings of the 14th International Conference on VLSI Design*, pages 205–210, 2001.

[MBCD14] D. McGrew, D. Bailey, M. Campagna, and R. Dugal. AES-CCM Elliptic Curve Cryptography (ECC) Cipher Suites for TLS. RFC 7251 (Informational), Internet Engineering Task Force, June 2014.

[Mer90] R. Merkle. One Way Hash Functions and DES. In G. Brassard, editor, *Advances in Cryptology – CRYPTO '89*, volume 435 of *Lecture Notes in Computer Science*, pages 428–446. Springer New York, 1990.

[MF10] F. Mattern and C. Floerkemeier. From the Internet of Computers to the Internet of Things. In K. Sachs, I. Petrov, and P. Guerrero, editors, *From Active Data Management to Event-based Systems and More*, pages 242–259. Springer Berlin Heidelberg, 2010.

[MG14] D. Migault and T. Guggemos. Diet-ESP: a flexible and compressed format for IPsec/ESP. Internet-Draft draft-mglt-ipsecme-diet-esp-01 (Working Draft), Internet Engineering Task Force, July 2014.

[MH14] R. Moskowitz and R. Hummen. HIP Diet EXchange (DEX). Internet-Draft draft-moskowitz-hip-dex-02 (Working Draft), Internet Engineering Task Force, December 2014.

[MHJH15] R. Moskowitz, T. Heer, P. Jokela, and T. Henderson. Host Identity Protocol Version 2 (HIPv2). RFC 7401 (Proposed Standard), Internet Engineering Task Force, April 2015.

[Mil86] V. S. Miller. Use of Elliptic Curves in Cryptography. *In Proceedings on Advances in Cryptology – CRYPTO '85, Springer New York*, pages 417–426, 1986.

[Mil01] I. Miller. Protection Against a Variant of the Tiny Fragment Attack (RFC 1858). RFC 3128 (Informational), Internet Engineering Task Force, June 2001.

[Miy10] K. Miyazawa. Design and implementation of Kerberos version 5 for embedded devices. In *Proceedings of the 8th IEEE International Conference on Industrial Informatics*, INDIN '10, pages 449–453, 2010.

[MKHC07] G. Montenegro, N. Kushalnagar, J. Hui, and D. Culler. Transmission of IPv6 Packets over IEEE 802.15.4 Networks. RFC 4944 (Proposed Standard), Internet Engineering Task Force, September 2007.

[Mos12] R. Moskowitz. HIP Diet EXchange (DEX). Internet-Draft draft-moskowitz-hip-dex-00 (Working Draft), Internet Engineering Task Force, August 2012.

[MR04] N. Modadugu and E. Rescorla. The Design and Implementation of Datagram TLS. In *Proceedings of the Network and Distributed System Security Symposium*, NDSS '04. Internet Society, 2004.

[mspgcc] MSPGCC project. GCC toolchain for MSP430. Online @ `http://sourceforge.net/projects/mspgcc/` [Last accessed: 16.06.2015].

[MW14] M. Miller and C. Wallace. End-to-End Object Encryption and Signatures for the Extensible Messaging and Presence Protocol (XMPP). Internet-Draft draft-miller-xmpp-e2e-07 (Working Draft), Internet Engineering Task Force, July 2014.

[MWS04] D. Malan, M. Welsh, and M. Smith. A Public-Key Infrastructure for Key Distribution in TinyOS Based on Elliptic Curve Cryptography. In *Proceedings of the 1st Annual IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks*, SECON '04, pages 71–80, 2004.

[NB09] E. Nordmark and M. Bagnulo. Shim6: Level 3 Multihoming Shim Protocol for IPv6. RFC 5533 (Proposed Standard), Internet Engineering Task Force, June 2009.

[netfilter] netfilter/iptables project homepage - The netfilter.org project. Online @ `http://www.netfilter.org/` [Last accessed: 16.06.2015].

[NHR11] T. Narten, G. Huston, and L. Roberts. IPv6 Address Assignment to End Sites. RFC 6177 (Best Current Practice), Internet Engineering Task Force, March 2011.

[Nir14] Y. Nir. Using Client Puzzles to Protect TLS Servers From Denial of Service Attacks. Internet-Draft draft-nir-tls-puzzles-00 (Working Draft), Internet Engineering Task Force, April 2014.

[NIST01] Advanced Encryption Standard (AES), National Institute of Standards and Technology Federal Information Processing Standard Publication 197, November 2001.

[NIST05] M. Dworkin, Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication, National Institute of Standards and Technology Special Publication 800-38B, May 2005.

[NIST12a] E. Barker, W. Barker, W. Burr, W. Polk, and M. Smid, Recommendation for Key Management – Part 1: General, National Institute of Standards and Technology Special Publication 800-57, Revision 3, July 2012.

[NIST12b] Secure Hash Standard (SHS), National Institute of Standards and Technology Federal Information Processing Standard Publication 180-4, March 2012.

[NIST13a] E. Barker, L. Chen, A. Roginsky, and M. Smid, Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography, National Institute of Standards and Technology Special Publication 800-56A, Revision 2, May 2013.

[NIST13b] Digital Signature Standard (DSS), National Institute of Standards and Technology Federal Information Processing Standard Publications 186-4, July 2013.

[NS15] Y. Nir and V. Smyslov. Protecting Internet Key Exchange (IKE) Implementations from Distributed Denial of Service Attacks. Internet-Draft draft-ietf-ipsecme-ddos-protection-01 (Working Draft), Internet Engineering Task Force, March 2015.

[NVHAG11] P. Nie, J. Vähä-Herttua, T. Aura, and A. Gurtov. Performance Analysis of HIP Diet Exchange for WSN Security Establishment. In *Proceedings of the 7th ACM Symposium on QoS and Security for Wireless and Mobile Networks*, Q2SWinet '11, pages 51–56, 2011.

[NYHR05] C. Neuman, T. Yu, S. Hartman, and K. Raeburn. The Kerberos Network Authentication Service (V5). RFC 4120 (Proposed Standard), Internet Engineering Task Force, July 2005.

[OASIS13] E. Rissanen, eXtensible Access Control Markup Language (XACML), OASIS Standard Version 3.0, January 2013.

[ODE$^+$06] F. Osterlind, A. Dunkels, J. Eriksson, N. Finne, and T. Voigt. Cross-Level Sensor Network Simulation with COOJA. In *Proceedings of the 31st IEEE Conference on Local Computer Networks*, LCN '06, pages 641–648, 2006.

[PACS11] V. Paxson, M. Allman, J. Chu, and M. Sargent. Computing TCP's Retransmission Timer. RFC 6298 (Proposed Standard), Internet Engineering Task Force, June 2011.

[Pav13] I. Pavlov. LZMA specification (DRAFT version). Online @ `http://dl.7-zip.org/lzma-specification.zip` [Last accessed: 16.06.2015], July 2013.

[PCS02] J. Park, E. K. P. Chong, and H. Siegel. Efficient Multicast Packet Authentication Using Signature Amortization. In *Proceedings of the IEEE Symposium on Security and Privacy*, S&P '02, pages 227–240, 2002.

[Per01] A. Perrig. The BiBa One-time Signature and Broadcast Authentication Protocol. In *Proceedings of the 8th ACM Conference on Computer and Communications Security*, CCS '01, pages 28–37, 2001.

[PK00] G. J. Pottie and W. J. Kaiser. Wireless Integrated Network Sensors. *Communications of the ACM*, 43(5):51–58, 2000.

[PK14] J. Park and N. Kang. Lightweight Secure Communication for CoAP-enabled Internet of Things using Delegated DTLS Handshake. In *Proceedings of the International Conference on Information and Communication Technology Convergence*, ICTC '14, pages 28–33, 2014.

[PKG+13] P. Porambage, P. Kumar, A. Gurtov, M. Ylianttila, and E. Harjula. Certificate based keying scheme for DTLS secured IoT. Internet-Draft draft-pporamba-dtls-certkey-01 (Working Draft), Internet Engineering Task Force, December 2013.

[PM03] A. Pannetrat and R. Molva. Efficient Multicast Packet Authentication. In *Proceedings of the Network and Distributed System Security Symposium*, NDSS '03. Internet Society, 2003.

[PN98] T. Ptacek and T. Newsham. Insertion, evasion, and denial of service: Eluding network intrusion detection. Technical report, Secure Networks, Inc., 1998.

[Pos81] J. Postel. Internet Protocol. RFC 791 (Internet Standard), Internet Engineering Task Force, September 1981.

[Pro14a] Proofpoint, Inc. Your Fridge is Full of SPAM: Proof of An IoT-driven Attack. Online @ `https://www.proofpoint.com/es/threat-insight/post/Your-Fridge-is-Full-of-SPAM` [Last accessed: 16.06.2015], January 2014.

[Pro14b] Proofpoint, Inc. Your Fridge is Full of SPAM, Part II: Details. Online @ `https://www.proofpoint.com/us/threat-insight/post/Your-Fridge-is-Full-of-SPAM-Part-2` [Last accessed: 16.06.2015], January 2014.

[PS08] G. Pelletier and K. Sandlund. RObust Header Compression Version 2 (ROHCv2): Profiles for RTP, UDP, IP, ESP and UDP-Lite. RFC 5225 (Proposed Standard), Internet Engineering Task Force, April 2008.

[PS13] J. Pope and R. Simon. The Impact of Packet Fragmentation and Reassembly in Resource Constrained Wireless Networks. *Journal of Computing and Information Technology*, 21(2):97–107, 2013.

[PSC05] J. Polastre, R. Szewczyk, and D. Culler. Telos: Enabling Ultra-low Power Wireless Research. In *Proceedings of the 4th International Symposium on Information Processing in Sensor Networks*, IPSN '05, pages 364–369, 2005.

[PST+02] A. Perrig, R. Szewczyk, J. Tygar, V. Wen, and D. Culler. SPINS: Security Protocols for Sensor Networks. *Wireless Networks*, 8(5):521–534, 2002.

[PSW04] A. Perrig, J. Stankovic, and D. Wagner. Security in Wireless Sensor Networks. *Communications of the ACM*, 47(6):53–57, 2004.

[PTSC00] A. Perrig, J. D. Tygar, D. Song, and R. Canetti. Efficient authentication and signing of multicast streams over lossy channels. In *Proceedings of the IEEE Symposium on Security and Privacy*, S&P '00, pages 56–73, 2000.

[RCBG10] B. Raman, K. Chebrolu, S. Bijwe, and V. Gabale. PIP: A Connection-oriented, Multi-hop, Multi-channel TDMA-based MAC for High Throughput Bulk Transfer. In *Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems*, SenSys '10, pages 15–28, 2010.

[RDC$^+$11] S. Raza, S. Duquennoy, T. Chung, D. Yazar, T. Voigt, and U. Roedig. Securing communication in 6LoWPAN with compressed IPsec. In *Proceedings of the International Conference on Distributed Computing in Sensor Systems and Workshops*, DCOSS '11, pages 1–8, 2011.

[RDS14] S. Raza, S. Duquennoy, and G. Selander. Compression of IPsec AH and ESP Headers for Constrained Environments. Internet-Draft draft-raza-6lo-ipsec-00 (Working Draft), Internet Engineering Task Force, March 2014.

[relic] D. F. Aranha and C. P. L. Gouvêa. RELIC is an Efficient LIbrary for Cryptography. Online @ `http://code.google.com/p/relic-toolkit/` [Last accessed: 16.06.2015].

[RFB01] K. Ramakrishnan, S. Floyd, and D. Black. The Addition of Explicit Congestion Notification (ECN) to IP. RFC 3168 (Proposed Standard), Internet Engineering Task Force, September 2001.

[RK03] E. Rescorla and B. Korver. Guidelines for Writing RFC Text on Security Considerations. RFC 3552 (Best Current Practice), Internet Engineering Task Force, July 2003.

[RKA09] R. Riaz, K.-H. Kim, and H. Ahmed. Security analysis survey and framework design for IP connected LoWPANs. In *Proceedings of the International Symposium on Autonomous Decentralized Systems*, ISADS '09, pages 1–6, 2009.

[RL09] R. Roman and J. Lopez. Integrating Wireless Sensor Networks and the Internet: A Security Analysis. *Internet Research*, 19(2):246–259, 2009.

[RM12] E. Rescorla and N. Modadugu. Datagram Transport Layer Security Version 1.2. RFC 6347 (Proposed Standard), Internet Engineering Task Force, January 2012.

[RNL11] R. Roman, P. Najera, and J. Lopez. Securing the Internet of Things. *Computer*, 44(9):51–58, 2011.

[Röl12] C. Röller. User-Defined Trust Overlay Networks for Smart Objects. Diploma thesis, RWTH Aachen University, 2012.

[RP12] D. Rotondi and S. Piccione. Managing Access Control for Things: A Capability Based Approach. In *Proceedings of the 7th International Conference on Body Area Networks*, BodyNets '12, pages 263–268, 2012.

[RSA78] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21:120–126, 1978.

[RSD14] S. Raza, H. Shafagh, and O. Dupont. Compression of record and handshake headers for constrained environments. Internet-Draft draft-raza-dice-compressed-dtls-00 (Working Draft), Internet Engineering Task Force, March 2014.

[RSH$^+$13] S. Raza, H. Shafagh, K. Hewage, R. Hummen, and T. Voigt. Lithe: Lightweight Secure CoAP for the Internet of Things. *IEEE Sensors Journal*, 13(10):3711–3720, 2013.

[RTV12] S. Raza, D. Trabalza, and T. Voigt. 6LoWPAN Compressed DTLS for CoAP. In *Proceedings of the 8th IEEE International Conference on Distributed Computing in Sensor Systems*, DCOSS '12, pages 287–289, 2012.

[RVJ12] S. Raza, T. Voigt, and V. Jutvik. Lightweight IKEv2: A Key Management Solution for both Compressed IPsec and IEEE 802.15.4 Security. IAB Workshop on Smart Object Security, 2012.

[RVR11] S. Raza, T. Voigt, and U. Roedig. 6LoWPAN Extension for IPsec. IAB Workshop on Interconnecting Smart Objects with the Internet, 2011.

[SA00] F. Stajano and R. Anderson. The Resurrecting Duckling: Security Issues for Ad-hoc Wireless Networks. In B. Christianson, B. Crispo, J. Malcolm, and M. Roe, editors, *Security Protocols*, volume 1796 of *Lecture Notes in Computer Science*, pages 172–182. Springer Berlin Heidelberg, 2000.

[SA11] P. Saint-Andre. Extensible Messaging and Presence Protocol (XMPP): Core. RFC 6120 (Proposed Standard), Internet Engineering Task Force, March 2011.

[SAKR12] M. Sethi, J. Arkko, A. Keranen, and H. Rissanen. Practical Considerations and Implementation Experiences in Securing Smart Object Networks. Internet-Draft draft-aks-crypto-sensors-02 (Working Draft), Internet Engineering Task Force, March 2012.

[SB01] H. Shacham and D. Boneh. TLS Fast-Track Session Establishment. Internet-Draft draft-shacham-tls-fast-track-00 (Working Draft), Internet Engineering Task Force, September 2001.

[SB02] H. Shacham and D. Boneh. Fast-Track Session Establishment for TLS. In M. Tripunitara, editor, *Proceedings of the Network and Distributed System Security Symposium*, NDSS '02, pages 195–202. Internet Society, 2002.

[SB10] Z. Shelby and C. Bormann. *6LoWPAN: The Wireless Embedded Internet.* Wiley Publishing, 2010.

[SB14] S. Sahraoui and A. Bilami. Compressed and Distributed Host Identity Protocol for End-to-End Security in the IoT. In *Proceedings of the 5th International Conference on Next Generation Networks and Services*, NGNS '14, pages 295–301, 2014.

[SBR04] H. Shacham, D. Boneh, and E. Rescorla. Client-side Caching for TLS. *ACM Transactions on Information and System Security*, 7(4):553–575, 2004.

[SCFJ03] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. RTP: A Transport Protocol for Real-Time Applications. RFC 3550 (Internet Standard), Internet Engineering Task Force, July 2003.

[SCNB12] Z. Shelby, S. Chakrabarti, E. Nordmark, and C. Bormann. Neighbor Discovery Optimization for IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs). RFC 6775 (Proposed Standard), Internet Engineering Task Force, November 2012.

[SDTL10] K. Srinivasan, P. Dutta, A. Tavakoli, and P. Levis. An Empirical Study of Low-power Wireless. *ACM Transactions on Sensor Networks*, 6(2):16:1–16:49, 2010.

[selinux] SELinux Project. Security Enhanced Linux (SELinux). Online @ `http://selinuxproject.org/` [Last accessed: 16.06.2015].

[SGG08] A. Silberschatz, P. B. Galvin, and G. Gagne. *Operating System Concepts*. Wiley Publishing, 8th edition, 2008.

[SGS$^+$15] L. Seitz, S. Gerdes, G. Selander, M. Mani, and S. Kumar. ACE use cases. Internet-Draft draft-ietf-ace-usecases-04 (Working Draft), Internet Engineering Task Force, June 2015.

[Sha13] H. Shafagh. Leveraging Public-key-based Authentication for the Internet of Things. Master's thesis, RWTH Aachen University, July 2013.

[SHB14] Z. Shelby, K. Hartke, and C. Bormann. The Constrained Application Protocol (CoAP). RFC 7252 (Proposed Standard), Internet Engineering Task Force, June 2014.

[She10] Z. Shelby. Embedded Web Services. *IEEE Wireless Communications*, 17(6):52–57, 2010.

[Shi07] R. Shirey. Internet Security Glossary, Version 2. RFC 4949 (Informational), Internet Engineering Task Force, August 2007.

[SHSA15] Y. Sheffer, R. Holz, and P. Saint-Andre. Recommendations for Secure Use of Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS). RFC 7525 (Best Current Practice), May 2015.

[SMPT01] A. Shacham, B. Monsour, R. Pereira, and M. Thomas. IP Payload Compression Protocol (IPComp). RFC 3173 (Proposed Standard), Internet Engineering Task Force, September 2001.

[SO12a] Y. B. Saied and A. Olivereau. HIP Tiny Exchange (TEX): A distributed key exchange scheme for HIP-based Internet of Things. In *Proceedings of the 3rd International Conference on Communications and Networking*, ComNet '12, pages 1–8, 2012.

[SO12b] Y. Saied and A. Olivereau. D-HIP: A distributed key exchange scheme for HIP-based Internet of Things. In *Proceedings of the IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks*, WoWMoM '12, pages 1–7, 2012.

[SO12c] Y. Saied and A. Olivereau. (K, N) Threshold Distributed Key Exchange for HIP Based Internet of Things. In *Proceedings of the 10th ACM International Symposium on Mobility Management and Wireless Access*, MobiWac '12, pages 79–86, 2012.

[SPJ10] K. Sandlund, G. Pelletier, and L.-E. Jonsson. The RObust Header Compression (ROHC) Framework. RFC 5795 (Proposed Standard), Internet Engineering Task Force, March 2010.

[SS82] J. A. Storer and T. G. Szymanski. Data Compression via Textual Substitution. *Journal of the ACM*, 29(4):928–951, 1982.

[SSG13] L. Seitz, G. Selander, and C. Gehrmann. Authorization Framework for the Internet-of-Things. In *Proceedings of the 14th IEEE International Symposium and Workshops on a World of Wireless, Mobile and Multimedia Networks*, WoWMoM '13, pages 1–6, 2013.

[ST10] Y. Sheffer and H. Tschofenig. Internet Key Exchange Protocol Version 2 (IKEv2) Session Resumption. RFC 5723 (Proposed Standard), Internet Engineering Task Force, January 2010.

[ST15] S. Santesson and H. Tschofenig. Transport Layer Security (TLS) Cached Information Extension. Internet-Draft draft-ietf-tls-cached-info-19 (Working Draft), Internet Engineering Task Force, March 2015.

[SZET08] J. Salowey, H. Zhou, P. Eronen, and H. Tschofenig. Transport Layer Security (TLS) Session Resumption without Server-Side State. RFC 5077 (Proposed Standard), Internet Engineering Task Force, January 2008.

[TASS13] K. H. Teo, A. Abdullah, S. K. Subramaniam, and G. R. Sinniah. New Reassembly Buffer Management System in 6LoWPAN. *Proceedings of the Asia-Pacific Advanced Network*, 36:57–64, 2013.

[TdOV10] J. Tripathi, J. de Oliveira, and J. P. Vasseur. A performance evaluation study of RPL: Routing Protocol for Low power and Lossy Networks. In *Proceedings of the 44th Annual Conference on Information Sciences and Systems*, CISS '10, pages 1–6, 2010.

[TF15] H. Tschofenig and T. Fossati. A DTLS 1.2 Profile for the Internet of Things. Internet-Draft draft-ietf-dice-profile-13 (Working Draft), Internet Engineering Task Force, June 2015.

[TH14] P. Thubert and J. Hui. LLN Fragment Forwarding and Recovery. Internet-Draft draft-thubert-6lo-forwarding-fragments-02 (Working Draft), Internet Engineering Task Force, November 2014.

[tinydtls] O. Bergmann. tinyDTLS. Online @ `http://tinydtls.sourceforge.net/` [Last accessed: 16.06.2015].

[TK05] H. Tschofenig and D. Kroeselberg. Security Threats for Next Steps in Signaling (NSIS). RFC 4081 (Informational), Internet Engineering Task Force, June 2005.

[tmote] Moteiv Corporation. Tmote Sky Datasheet. Online @ `http://www.eecs.harvard.edu/~konrad/projects/shimmer/references/tmote-sky-datasheet.pdf` [Last accessed: 16.06.2015].

[VMZS+13] F. Vidal Meca, J. Ziegeldorf, P. Sanchez, O. Morchon, S. Kumar, and S. Keoh. HIP Security Architecture for the IP-Based Internet of Things. In *Proceedings of the 27th International Conference on Advanced Information Networking and Applications Workshops*, WAINA' 13, pages 1331–1336, 2013.

[WAR06] Y. Wang, G. Attebury, and B. Ramamurthy. A Survey of Security Issues in Wireless Sensor Networks. *IEEE Communications Surveys Tutorials*, 8(2):2–23, 2006.

[WH13] HART Communication Protocol Specification, HART Communication Foundation Standard HCF_SPEC-13, Revision 7.5, May 2013.

[WHF03] D. Whiting, R. Housley, and N. Ferguson. Counter with CBC-MAC (CCM). RFC 3610 (Informational), Internet Engineering Task Force, September 2003.

[wismote] Arago Systems. WiSMote Datasheet. Online @ `http://www.aragosystems.com/images/stories/WiSMote/Doc/wismote_en.pdf` [Last accessed: 16.06.2015].

[WKC+04] R. Watro, D. Kong, S.-f. Cuti, C. Gardiner, C. Lynn, and P. Kruus. TinyPK: Securing Sensor Networks with Public Key Technology. In *Proceedings of the 2nd ACM Workshop on Security of Ad Hoc and Sensor Networks*, SASN '04, pages 59–64, 2004.

[WL98] C. K. Wong and S. Lam. Digital Signatures for Flows and Multicasts. In *Proceedings of the 6th International Conference on Network Protocols*, ICNP '98, pages 198–209, 1998.

[WLSC07] J. P. Walters, Z. Liang, W. Shi, and V. Chaudhary. Wireless Sensor Network Security: A Survey. *Security in Distributed, Grid, Mobile, and Pervasive Computing*, 1:367, 2007.

[WM95] G.-L. Wu and J. W. Mark. A Buffer Allocation Scheme for ATM Networks: Complete Sharing Based on Virtual Partition. *IEEE/ACM Transactions on Networking*, 3(6):660–670, 1995.

[Woo11] J. Woodyatt. Recommended Simple Security Capabilities in Customer Premises Equipment (CPE) for Providing Residential IPv6 Internet Service. RFC 6092 (Informational), Internet Engineering Task Force, January 2011.

[WRTTG13] A. Weigel, M. Ringwelski, V. Turau, and A. Timm-Giel. Route-Over Forwarding Techniques in a 6LoWPAN. In *Mobile Networks and Management*, volume 125 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, pages 122–135. Springer International Publishing, 2013.

[WTB+12] T. Winter, P. Thubert, A. Brandt, J. Hui, R. Kelsey, P. Levis, K. Pister, R. Struik, J. Vasseur, and R. Alexander. RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks. RFC 6550 (Proposed Standard), Internet Engineering Task Force, March 2012.

[WTJ+11] G. Wu, S. Talwar, K. Johnsson, N. Himayat, and K. Johnson. M2M: From Mobile to Embedded Internet. *IEEE Communications Magazine*, 49(4):36–43, 2011.

[YMG08] J. Yick, B. Mukherjee, and D. Ghosal. Wireless Sensor Network Survey. *Computer Networks*, 52(12):2292 – 2330, 2008.

[Zav11] G. Zaverucha. Standards for Efficient Cryptography 4: Elliptic Curve Qu-Vanstone Implicit Certificate Scheme (ECQV). Working Draft, Certicom Research, March 2011.

[ZB12] ZigBee 2012 Specification, ZigBee Alliance Standard, 2012.

[ZBIP13] ZigBee IP Specification, ZigBee Alliance Standard, February 2013.

[ZL77] J. Ziv and A. Lempel. A Universal Algorithm for Sequential Data Compression. *IEEE Transactions on Information Theory*, 23(3):337–343, 1977.

[zlib] J.-L. Gailly and M. Adler. zlib - A Massively Spiffy Yet Delicately Unobtrusive Compression Library. Online @ `http://www.zlib.net` [Last accessed: 16.06.2015].

[zolertia] Advancare, SL. Zolertia Z1 Datasheet. Online @ `http://zolertia.com/sites/default/files/Zolertia-Z1-Datasheet.pdf` [Last accessed: 16.06.2015].

[ZRT95] G. Ziemba, D. Reed, and P. Traina. Security Considerations for IP Fragment Filtering. RFC 1858 (Informational), Internet Engineering Task Force, 1995.

[ZV10] J. Zhang and V. Varadharajan. Wireless Sensor Network Key Management Survey and Taxonomy. *Journal of Network and Computer Applications*, 33(2):63 – 75, 2010.

Curriculum Vitae

Personal Details

Last Name:	Hummen
First Name:	René
Date of Birth:	October 20, 1982
Place of Birth:	Aachen, NRW, Germany
Nationality:	German

Education

High School 1993 – 2002	Städtisches Gymnasium Herzogenrath Abitur: June 2002
University 2002 – 2009	RWTH Aachen University Major: Computer Science, Minor: Business Economics Degree: Dipl.-Inform.
PhD Student 2009 – 2015	RWTH Aachen University Chair of Communication and Distributed Systems Adviser: Prof. Dr.-Ing. Klaus Wehrle