Chapter 1
Two-way Authentication for the Internet-of-Things

Corinna Schmitt[1], Thomas Kothmayr[2], Wen Hu[3], and Burkhard Stiller[1]

Abstract This chapter introduces the first fully implemented two-way authentication security scheme for Internet-of-Things (IoT) based on existing Internet standards, specifically the Datagram Transport Layer Security (DTLS) protocol. By relying on an established standard, existing implementations, engineering techniques, and security infrastructure can be reused, which enables an easy security uptake. The proposed security scheme uses two public key cryptography algorithms, RSA (Rivest, Shamir und Adleman) and Elliptic Curve Cryptography (ECC), tailored for the resource heterogeneous nature of IoT devices. The two-way authentication solution presented is designed to work over standard communication stacks that offer UDP/IPv6 networking for Low power Wireless Personal Area Networks (LoWPANs). A prototype implementation of DTLS is presented here in the context of a system architecture, and the scheme's feasibility (low overheads and high interoperability) is demonstrated through extensive evaluations on the DTLS-supporting platform OPAL as clusterhead with children of different IoT hardware platforms.

Key words: Internet-of-Things (IoT), Wireless Sensor Networks (WSNs), Two-way Authentication, Constrained Devices, End-to-End Security, Datagram Transport Layer Security (DTLS), Elliptic Curve Cryptography (ECC)

[1] Communication Systems Group CSG, Department of Informatics IfI, University of Zurich, Binzmühlestrasse 14, 8050 Zurich, Switzerland, e-mail: {schmitt,stiller}@ifi.uzh.ch
[2] Fakultät für Informatik, Technische Universität München, Boltzmannstrasse 3, 85748 Garching, Germany, e-mail: kothmayr@in.tum.de
[3] School of Computer Science and Engineering, The University of New South Wales, Sydney, NSW 2052, Australia, e-mail: wen.hu@unsw.edu.au

1.1 Introduction

Today, a multitude of envisioned as well as implemented use cases for the Internet-of-Things (IoT) and Wireless Sensor Networks (WSNs) exists. It is desirable for certain scenarios to also make data globally accessible: (a) to authorized users only and (b) to data processing units through the Internet. Naturally, much of the data collected, such as locations and personal identifiers, are of sensitive nature. Even seemingly inconspicuous data, such as the energy consumption measured by a smart meter, can lead to potential infringements on the users' privacy, *e.g.*, by allowing an eavesdropper to conclude whether or not a user is currently at home. From an industry perspective a pressing need for security solutions based on standards has risen. The market research firm Gartner states in [1]:

> *"The Internet of Things concept will take more than 10 years to reach the Plateau of Productivity - mainly due to security challenges, privacy policies, data and wireless standards, and the realization that the Internet of Things requires the build-out of a topology of services, applications and a connecting infrastructure."*

Regarding the infrastructure, security risks are aggravated by the trend towards a separation of sensor network infrastructure and applications [2, 3]. Therefore, a true end-to-end security solution is required to achieve an adequate level of security for the IoT. Protecting data once it leaves the local network is not sufficient.

A similar scenario in the traditional computing world comprises a user browsing the Internet on top of an unsecured Wireless LAN (Local Area Network). Attackers in physical proximity of the user can capture the traffic between the user and a Web server. Well known countermeasures against such attacks include the establishment of a secured connection to the Web server via HTTPS (Secured Hyper Text Transfer Protocol), the use of a VPN (Virtual Private Network) tunnel to securely connect to a trusted VPN endpoint, and using wireless network security such as WPA (Wireless Protected Access). These solutions are comparable to security approaches in the IoT area. Using WPA is similar to the traditional use of a link layer encryption. The VPN solution is equivalent to creating a secure connection between a sensor node and a security endpoint, which may or may not be the final destination of the sensor data. Establishing a HTTPS connection with the server is comparable to our approach: We investigate the use of the DTLS protocol in an end-to-end security architecture for the IoT. DTLS is an adaption of the widespread TLS protocol, used to secure HTTPS, for unreliable datagram transport.

However, the Internet is not limited to servers, routers, and computers with manifold resources anymore as more constrained devices are connected to it, forming the Internet-of-Things (IoT). Those devices - sensor nodes/motes - are very limited in memory (approx. 10-50 kByte RAM and 100-256 kByte ROM), computational capacity, and power (a few AAA batteries). Nevertheless, these devices still have to support end-to-end security, as requested by IoT, and secure communication with their limited resources.

For developing a security solution developers have to take design decisions into account. The presented solution in this book chapter considers three essential high-level design decisions:

Implementation of a standards-based design: Standardization has helped the widespread uptake of technologies. Radio chips can rely on IEEE 802.15.4 for the physical and the MAC layer. The IPv6 Routing Protocol for Low power and Lossy Networks (RPL), or called IPv6 over Low Power Wireless Personal Area Networks (6LoWPAN), provides routing functionality and the Constrained Application Protocol (CoAP) [4, 5] defines the application layer. So far, no such efforts have addressed security in a wider context for the IoT.

Focus on application layer end-to-end security: An end-to-end protocol provides security even if the underlying network infrastructure is only partially under the user's control. As the infrastructure for Machine-to-Machine (M2M) communication is getting increasingly commoditized, this scenario becomes more likely: The European Telecommunications Standards Institute (ETSI) is currently developing a standard that focuses on providing a "horizontal M2M service platform" [3], meaning that it plans to standardize the transport of local device data to a remote data center. For stationary installations security functionality could be provided by the gateway to the higher-level network. However, such gateways would present a high-value target for an attacker. If the devices are mobile, for example in a logistics application, there may not be a gateway to a provider's network that is under the user's control, similar to how users of smart phones connect directly to their carrier's network. Another example that favors end-to-end security is a multi-tenancy office building that is equipped with a common infrastructure for metering and climate-control purposes. The tenants share the infrastructure but are still able to keep their devices' data private from other members of the network. Using a protocol like Datagram Transport Layer Security (DTLS), which is placed between transport and application layer, does not require that the infrastructure provider supports the security mechanism. It is purely in the hands of the two communicating applications to establish security. If the security is provided by a network layer protocol, such as IPsec (Internet Protocol Security), the same is true to a lesser degree, because the network stacks of both devices must support the same security protocol.

Support for UDP (User Datagram Protocol): Reliable transport protocols like the Transmission Control Protocol (TCP) incur an overhead over simpler, unreliable protocols such as UDP. Especially for energy starved, battery powered devices this overhead is often extremely costly and TCP has been shown to perform poorly in low-bandwidth scenarios [6]. This is reflected in the design of the emerging standard CoAP, which uses UDP transport and defines a binding to DTLS for security [4]. By using DTLS in conjunction with UDP in the proposed approach (cf. Section 1.4) the application developer are not forced to use reliable transport as would be the case if TLS would be used. It is still possible to use DTLS over transport protocols like TCP, because DTLS only assumes an unreliable transport.

A resource-full device, perhaps including a Trusted Platform Module (TPM), raises the memory capacity to 50 kByte RAM and 256 kByte ROM, which allows the use of the DTLS protocol in an end-to-end security architecture for IoT. DTLS is an adaption of the widespread TLS (Transport Layer Security) protocol, used for Hypertext Transfer Protocol Secure (HTTPS), for unreliable datagram transport. In the proposed solution for resource-full devices a common DTLS handshake is performed, where both communication parties authenticate each other using a X.509 certificate in order to establish a secure communication channel before exchanging data itself. When looking on third design decision it is a weaker property than the reliability provided by TCP. However, the adaptations of DTLS for an unreliable transport introduces additional overhead compared to TLS. It might be beneficial to use TCP during the handshake phase, but the DTLS reliability mechanism for the proposed solution should be adapted to the special requirements of constrained networks as it is the case for wireless sensor networks. A study of TCP's influence on the handshake is, therefore, out of scope of this article. [7]

For devices with fewer resources (*e.g.*, at around 10 kByte RAM and 100 kByte ROM), two-way authentication, as done within DTLS, is not feasible, because computing resources are too limited for extensive computations. But the aforementioned three high-level design decisions should also be supported. Therefore, Noack [8] developed a similar two-way authentication handshake solution applying keyed hash functions for authorization and message authentication as proposed by [9], instead of certificates. For further information it is referred to [8].

The rest of this book chapter is organized as follows: First, background information about the IoT, the specialized area Wireless Sensor Networks, and device class definition is presented. Followed by a brief characterization of existing security solutions for different devices classes. Section 1.3 introduces the standard-based end-to-end security architecture in detail, as well as general security goals and device's roles in a WSN. For resource-full devices a DTLS-based solution is presented in Section 1.4 based on [7]. The solution supports two-way authentication for all involved communication parties. Section 1.5 focuses on evaluation of the proposed solution addressing resource consumption, handshake performance, and comparison to related work. Finally, a brief summary and conclusion is given.

1.2 Background Information

This chapter presents the principles of the IoT with a special focus on WSNs and the used hardware. Furthermore, the vulnerabilities of such networks to different attacks are presents as well as the resulting necessary end-to-end security. Finally, related work for the proposed DTLS-based solution is presented.

1.2.1 Wireless Sensor Networks and Internet-of-Things

WSNs consist of a large number of small, cheap, and smart computing and sensing devices connected over radio. The advances in hardware, software, and networking have made practical WSN deployment technically and economically viable and enabled many applications in the areas such as habitat monitoring, health, structure monitoring, precision agriculture, and military.

While WSNs have revolutionized the way in which the authors of this book chapter understand, monitor, and control complex physical environments, the communication protocols for WSNs were custom-made to minimize resource consumptions for embedded sensing devices. Recent research in the IoT area has been in favor of the unique identification (*e.g.*, by IPv6 addresses) of embedded computing devices within the existing Internet infrastructure over resource consumption. Therefore, the embedded devices in IoT are globally addressable and communicate with optimized Internet communication protocols such as 6LoWPAN. The IoT facilitates machine-to-machine communications (M2M) without human intervention and enables advanced applications such as smart home control and wearable computing. For further information about the IoT it is referred to [10, 11]. Within this book chapter the authors focus on WSNs, especially on light-weighted solutions, due to the limited hardware resources of used devices. Detailed information about requirements, protocols, and architecture requirements are presented in [12]. In the following only a brief overview of device resources are introduced in order to justify the proposed solution in Section 1.4.

As mentioned before the used devices are small; thus, the devices have limited resources that gave the devices the name: “constrained devices”. They are limited in memory, computational capacity, and power. In the RFC 7228 [13] constrained devices were specified and organized in classes based on memory resources. These devices can be grouped into the following three classes:

- **Class 0 devices** are sensor-like nodes, usually pre-configured, and have less than 10 kByte RAM respectively 100 kByte Flash. In general class 0 devices are not able to communicate directly and secure with the Internet. In order to participate in Internet communications help of larger devices is required. Looking on the projects scope class 0 devices are unable to secure or managed communication in traditional sense.
- **Class 1 devices** have around 10 kByte RAM respectively 100 kByte Flash available. Compared to class 0 devices they are unable to talk easily to other Internet nodes employing a full protocol stack (*e.g.*, HTTP, TLS, security protocols, Extensible Markup Language (XML) based data representations). Generally, class 1 devices use specifically designed protocol stacks (*e.g.*, CoAP over UDP) and, therefore, do not require gateway nodes for conversation purposes. Class 1 devices are able to provide support for security functions required on large networks, as it is scope of this proposal. Furthermore, those devices can be integrated as fully developed peers into an IP network.

- **Class 2 devices** are more memory richer, usually having around 50 kByte RAM and 250 kByte Flash available. With this range they can support mostly same protocol stacks as used on notebooks or servers. But class 2 devices can still benefit from lightweight and energy-efficient protocols in order to increase lifetime. Compared to the other device classes those devices use a smaller percentage of their resources for networking, leaving more resources available for applications. Furthermore, if class 2 devices also support protocol stacks for lower classes development costs can be reduced and interoperability increased.

Devices with capabilities significantly beyond class 2 must be mentioned, because they are also included in the wide device diversity in the IoT. Those devices can usually use existing protocols in an unchanged manner, but may still be constrained by a limited energy supply. Smartphones are an example for these class 2+ devices.

Throughout this book chapter all devices within class 0 and class 1 build the group of "resource-less devices"; for all devices in class 2 or higher the term "resource-full device" is used.

As mentioned before, another limiting factor is energy. [14] points out that power consumption of wireless sensor network devices could be divided into three domains: sensing, communication, and data processing. The latter two are also valid for devices beyond class 2. All three domains are closely related to the application and supported protocols in the network. If sensing takes place only sporadically, power consumption will be smaller than supporting constant monitoring tasks. Another dimension that requires energy is an event detection that is closely related to sleep modes. Most energy is required for communication purposes, including transmission and reception of data. Depending on the used power and data size/format the cost can differ. But the most consuming part - the start-up of the transceiver and, thus, turning a device on or off - does not only save energy. By looking on energy consumption for data processing it is smaller compared to communications and it is highly influenced by protocols supported. [14] presented a good comparison:

> *"Assuming Rayleigh fading and fourth power distance loss, the energy cost of transmitting 1 kByte a distance of 100 m is approximately the same as that for executing 3 million instructions by a 100 million instructions per second (MIPS)/W processor."*

They also pointed out that it is an advantage to integrated powerful devices in the network and outsource data processing functions to those devices if possible.

[13] pointed out that all devices can also be grouped corresponding to energy and power. In general, it can be said that devices beyond class 2 usually have unlimited power resource. Class 0, class 1, and class 2 devices can be distinguished in event energy-limited devices using event-based harvesting sources, period energy-limited devices including battery recharging or replacing, and lifetime energy-limited devices with fixed batteries without replacing them. The energy support is closely related to the application, its lifetime, and the type of power source used by the device.

1.2.2 Security Solutions for Wireless Sensor Networks

As described before, WSNs within the area of IoT are very prone security-wise, because the information transmitted includes sensitive data, is transmitted via the wireless medium using UDP, and usually only the end points of the communication chain should be able to read the message. Thus, the call for end-to-end security solutions grows. The development of a solution is challenging, because of the heterogeneous characteristics of deployed networks and the used devices with limited resources. Depending on the hardware used and the application itself different solutions were developed addressing resource-full and resource-less devices as pointed out in [7, 8].

Traditionally, security protocols in sensor networks focus on link layer security, protecting data on a hop-by-hop basis. The simplest approach to link layer security consists of using a network-wide encryption key, which often is the case in ZigBee networks [15]. ZigBee also provides support for cluster and individual link keys. MiniSec [16] is another well-known security mechanism for WSNs that provides data confidentiality, authentication and replay protection. Similar to ZigBee, the packet overhead introduced by MiniSec has only a few Byte. The widespread TinySec link layer security mechanism is no longer considered secure [16]. Most security protocols do not include a mechanism on how encryption keys are distributed to the nodes. Keys are either loaded onto the nodes before setup or a separate key establishment protocol is used. Public key cryptography (PKC) is used in traditional computing to facilitate secure key establishment. However, public key cryptography, in particular the widespread RSA algorithm, has been considered too resource consuming for constrained devices. Some security protocols, such as Sizzle [17], advocate the use of the more resource efficient ECC public key cryptosystem. Other research efforts, such as the secFleck [18] mote, provide support for faster RSA operations through hardware.

Approaches without PKC often rely on the pre-distribution of connection keys. Random key pre-distribution schemes, such as the q-composite scheme by [19], establish connections with a nodes neighbors with a certain probability $p < 1$. Intuitively, pre-distributed key schemes such as this require a large amount of keys to be loaded onto the nodes before deployment. Depending on the method used, this approach is scaling in $O(n^2)$ or $O(n)$ where n is the number of nodes in the network. The Peer Intermediaries for Key Establishment (PIKE) protocol achieves sub linear scaling in $O(\sqrt[2]{n})$ by relying on the other nodes as trusted intermediaries. While PIKE provides higher memory efficiency than random schemes, it still leaks additional key information when motes are captured.

Recently, more research into end-to-end security protocols for the IoT and WSNs is being conducted. As outlined in the introduction, such a protocol protects the message payload from the data source until it reaches its target. Because end-to-end protocols are usually implemented in the network or application layer, for-warding nodes do not need to perform any additional cryptographic operations since the routing information is transmitted in the clear. On the flip side, this means end-to-

end security protocols do not provide the same level of protection of a network's availability as a link layer protocol. One example of an end-to-end security protocol is Sizzle by [17]. Sizzle is a compact web server stack providing HTTP services secured by SSL (Secure Sockets Layer). It uses 160-bit ECC keys for key establishment, which provide a similar level of security as 1024-bit RSA keys. In contrast to the presented solution in Section 1.4, it requires a reliable transport layer, which has been shown to incure large performance penalties in low bandwidth situations [6]. Sizzle also omits two-way authentication: Only the Sizzle enabled node is authenticated by a remote, more resource rich, client. This is insufficient for machine-to-machine communication in the IoT. SSNAIL [20] makes similar design choices as Sizzle and performs an ECC handshake over reliable TCP transport. Similar to the implementation of the described DTLS solution in this book chapter, SSNAIL is able to perform a full, two-way authenticated handshake but it still requires a reliable transport protocol.

[21] discussed how the IPsec protocol could be integrated into 6LoWPAN, the compressed IPv6 implementation used in most IP-enabled sensor networks. Their work focuses on how data transfer with IPsec can be made efficient in the context of 6LoWPAN. Regarding the Internet Key Exchange (IKE) protocol, which is used for key establishment in IPsec networks, [22] discussed methods for reducing the headers to make IKE more suitable for con-strained devices, but do not present a performance analysis alongside their proposal.

As mentioned in Section 1.1, CoAP is an application layer standardization effort for the IoT. The current draft specifies a binding of CoAP to DTLS to achieve security [4]. Another proposal by [23] aims to reduce the communication overhead of the DTLS headers through compression. As with the work on IPsec, the authors of this book chapter are currently not aware of any publication evaluating the performance of DTLS over 6LoWPAN. The presented DTLS solution in this book chapter can thus support these efforts by providing a set of real-world measurements from the presented DTLS implementation.

1.3 A Standard-based End-to-End Security Architecture

The assumed system architecture is following the IoT model. It is assumed that IPv6 connects the Internet in the near future and parts of it run 6LoWPAN. The Transport layer in 6LoWPAN is UDP, which can be considered unreliable; the routing layer is RPL [24] or Hydro [6]. The DTLS implementation uses Hydro for routing, because at the time of writing the implementation code there was no available RPL implementation for TinyOS [25]. RPL has since been standardized in RFC 6550 [24] and is distributed with newer versions of TinyOS. Thus, RPL was chosen for the two-way authentication solution with ECC for resource-less devices that is not addressed in this book chapter but can be read in detail in [8]. However, both routing protocols are similar enough so that a change should have negligible impact on the

presented results. IEEE 802.15.4 is used for the physical and Media Access Control (MAC) layer. Based on this protocol stack DTLS and ECC were chosen as security protocol, which places it in the application layer on top of the UDP transport layer. The final stack structure might differ depending on chosen deployment, application, and implementation of functionalities. For example, the certificate authority (CA) or Access Control (AC) server can be included in the gateway complex. Detailed description is available in [7, 26].

The general idea of the architecture (cf. Figure 1.1 [7]) is that a subscriber wants to access data of the WSN. A designated device in the WSN, called publisher, is al-lowed to publish collected data and make it available from outside of the WSN (*e.g.*, for analysis purposes or announcement to other systems like an air conditioning system). The subscriber has to establish a connection to the publisher and, therefore, it is necessary to establish an end-to-end secured connection. Because of sensitive data (*e.g.*, Global Positioning System (GPS) data) included in the measurements of sensor nodes it is essential to authenticate the communication partners (here: subscriber and publisher). Depending on the resources of the devices different solutions are possible. This book chapter focuses on resource-full devices and the solution is described in upcoming Section 1.4 For an appropriate solution for resource-less device it is referred to [8, 27].

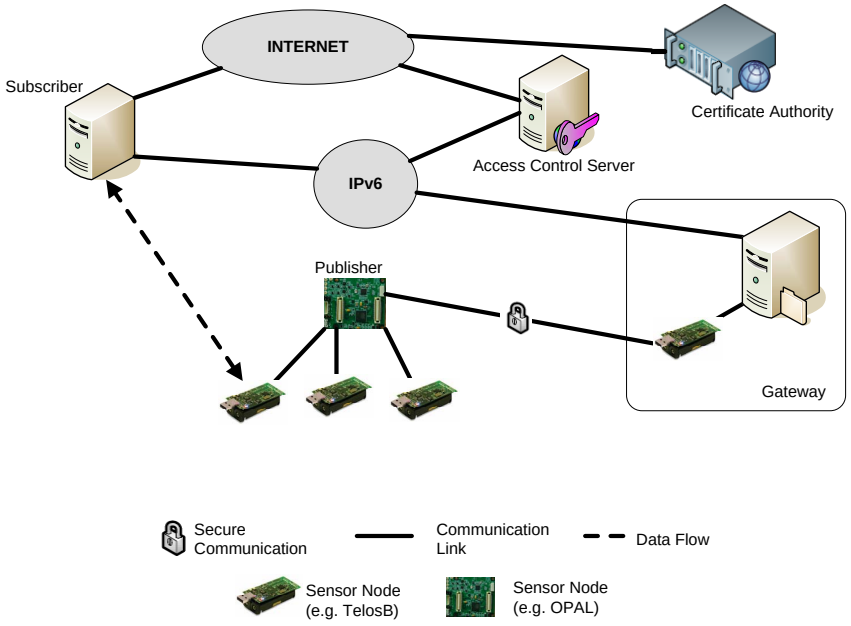

Fig. 1.1 System Architecture.

Due to the assumption that WSNs are a subset of the IoT area, it is assumed that similar security needs as in traditional networks, like IP networks, have to be considered. Thus, the solution proposed will address the following three security goals:

- **Authenticity:** Recipients of a message can identify their communication partners and can detect if the sender information has been forged.
- **Integrity:** Communication partners can detect changes to a message during transmission.
- **Confidentiality:** Attackers cannot gain knowledge about the contents of a secured message.

For the remainder of this book chapter the following roles for devices in a WSN are considered, all besides the roles of publisher and subscriber as introduced in [8]:

- **Collectors** are limited to collecting and transmitting environmental data. They do not execute any preprocessing tasks on collected data; instead they send only raw information. Those could include, for example, humidity, temperature, light, and voltage. Measurements are executed periodically and immediately followed by data transmission to the gateway.
- **Aggregators** are selected devices in the network with more resources than collectors. The aggregator can pre-process data within the network and, thus, reduce the traffic in the network. The performed degree of aggregation (doa) can vary depending on the device resources and per-formed function. Examples are data aggregation and message aggregation as described in [26].
- The **gateway** is usually a complex of a sensor node and a server-like component (e.g., router, server, persona computer (PC)). It connects the WSN components to the IP networks on the outside. It basically brings wireless communication to wired communication and makes the collected data available for other applications. [26]

1.4 DTLS Solution for Resource-full Devices

This section assumes resource-full devices that are able to perform difficult encryption operations and, thus, can perform a DTLS handshake in order to support two-way authentication. In the following the used message structure, the per-formed DTLS handshake, and security considerations are addressed following [7, 26, 27].

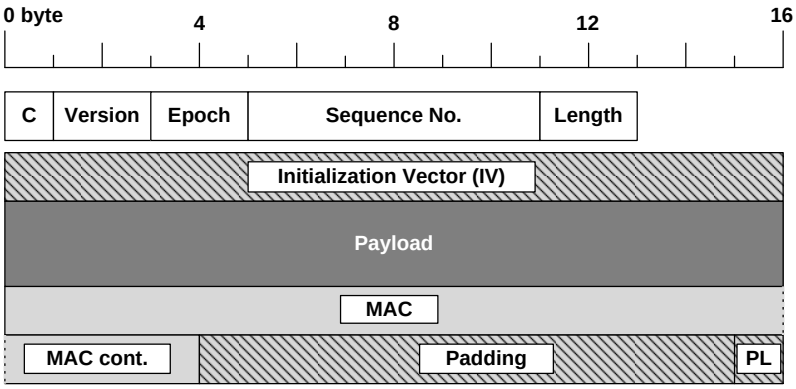

Fig. 1.2 A DTLS record protected with CBC block cipher.

1.4.1 DTLS Protected Message Structure

All messages sent via DTLS are prepended with a 13 Byte long DTLS record header. This header specifies the content of the message (*e.g.*, application data or handshake data), the version of the protocol employed, as well as a 64-bit sequence number and the record length. The top two Byte of the sequence number are used to specify the epoch of the message, which changes once new encryption parameters have been negotiated between client and server. Figure 1.2 shows the DTLS record header in white [7]. The record header is either followed by the plaintext if no security has been negotiated yet, or by the DTLS block cipher. If a block cipher is used, the plaintext is prepended by a random Initialization Vector (IV), which has the size of the cipher block length. This protects against attacks where attackers can adaptively choose plaintext. The plaintext is followed by a Hash-based Message Authentication Code (HMAC), which allows the receiver to detect if the DTLS record has been altered. Finally, the message is padded to a multiple of the cipher block length. The payload itself is encrypted with the block cipher, where IV and padding are not used to calculate the HMAC. Unlike TLS, DTLS does not allow for stream ciphers because they are sensitive to message loss and reordering. Instead, DTLS uses block ciphers in the Cipher-Block Chaining (CBC) mode of operation.

1.4.2 Certificate Structure

As mentioned earlier the proposed solution in this book chapter requires certificates for authentication purposes. Thus, this section briefly introduces the general structure of X.509 certificates based on RFC 6818 [28]. For further information it is referred to the RFC 6818 and common literature about Public Key Infrastrukture

(PKI). Based on RFC 6818 a X.509 certificate should include the following items in general:

1. Serial Number
2. Validity: Not Before: Date and time, Not After: Date and time
3. Subject: commonName = localhost
4. X509v3 extensions including X509v3 Basic Constraints: CA:FALSE, Netscape Comment: OpenSSL Generated Certificate, X509v3 Subject Key Identifier, and X509v3 Authority Key Identifier

Depending on the implementation additional information should be requested that will be incorporated into the certificate request. For the proposed solution the following items where selected:

1. Country Name (2 letter code) [ ]
2. State or Providence Name (full name) [ ]
3. Locality Name (*e.g.,* city) [ ]
4. Organization Name (*e.g.,* company) [ ]
5. Organization Unit name (*e.g.,* section) [ ]
6. Common name (*e.g.,* YOUR Name) [ ]
7. Email address [ ]
8. optional: challenge password [ ] and company name [ ]

1.4.3 DTLS Handshake Assumption

The key material and cipher suite, consisting of a block cipher and a hash algorithm, are negotiated between client and server during the handshake phase, which commences before any application data can be transferred. There are three types of handshake: unauthenticated, server authenticated and fully authenticated handshakes. During an unauthenticated handshake neither party authenticates against the other and during a server-authenticated handshake only the server proves its identity to the client. In a fully authenticated handshake the client has to authenticate itself to the server as well. In the following the unauthenticated handshake is not considered, because it provides no authenticity at all.

There are different algorithms that can be used for authentication in a DTLS handshake. Variants based on ECC have been shown in embedded networks [17]. Since it was argued for standard-based communication architecture for the IoT to promote interoperability, the rest of this section will focus on authentication based on RSA. Because it is todays dominant PKC system [29] a suitable infrastructure for obtaining certificates from commercial certificate authorities is already in place.

Figure 1.3 shows a fully authenticated DTLS handshake [7]. Individual messages are grouped into “message flights” according to their direction and occurrence sequence. Flight 1 and 2 are an optional feature to protect the server against Denial-of-Service (DoS) attacks. The client has to prove that it can receive as well as send

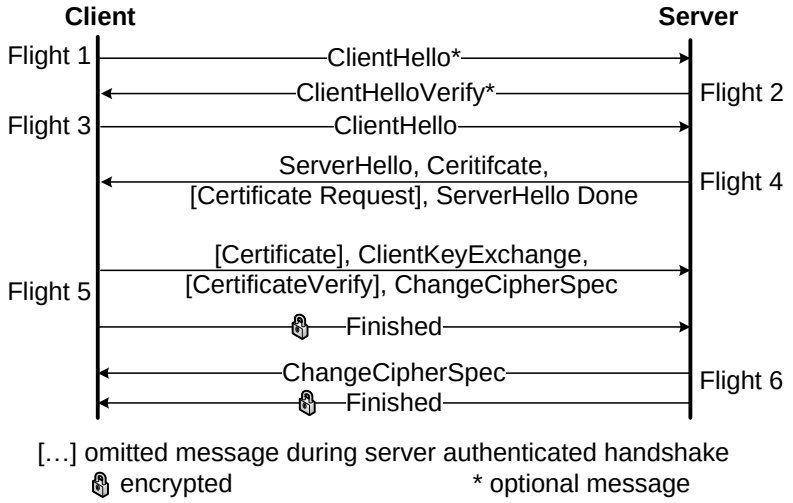

Fig. 1.3 Fully Authenticated DTLS Handshake.

data by resending its *ClientHello* message with the cookie sent in the *ClientHello-Verify* message by the server. The *ClientHello* message contains the protocol version supported by the client as well as the cipher suites that it supports. The server answers with its *ServerHello* message that contains the cipher suite chosen from the list offered by the client. The server also sends a X.509 certificate to authenticate itself followed by a *CertificateRequest* message if the server expects the client to authenticate. The *ServerHelloDone* message only indicates the end of flight 4. If requested and supported, the client sends its own certificate message at the beginning of flight 5. The *ClientKeyExchange* message contains half of the pre-master secret encrypted with the servers public RSA key from the server's certificate. The other half of the pre-master secret was transmitted unprotected in the *ServerHello* message. The keying material is subsequently derived from the pre-master secret. Since half of the pre-master secret is encrypted with the servers public key it can only complete the handshake if it possesses the private key matching the public key in the server certificate. Accordingly, in the *CertificateVerify* message the client authenticates itself by proving that it is in possession of the private key matching the client's public key.

It does this by signing a hashed digest of all previous handshake messages with its private key. The server can verify it through the public key of the client. The *ChangeCipherSpec* message indicates that all following messages by the client will be encrypted with the negotiated cipher suite and keying material. The *Finished* message contains an encrypted message digest of all previous handshake messages to ensure both parties are indeed operating based on the same, unaltered, handshake data. The server answers with its own *ChangeCiperSpec* and *Finished* message to complete the handshake.

1.4.4 Security Considerations for DTLS Handshake

By choosing DTLS as the security protocol the aforementioned security goals can be achieved. DTLS is a modification of TLS for the unreliable UDP and inherits its security properties [30]. Using an application layer security protocol like DTLS as opposed to link or network layer security protocols such as MiniSec [16] has a number of advantages but also some drawbacks as described in the following.

Lower layer security protocols do not provide end-to-end communication security. Data is decrypted on receipt and re-encrypted for forwarding on each hop in a multi-hop network. An attacker can therefore gain access to all clear text data that passes through a compromised node. Scalability is often also an issue for these protocols because they need to establish a secured connection with each of their neighbors to form a mesh network and cryptographic overhead occurs on each hop. On the other hand, in an end-to-end security protocol, cryptographic overhead occurs on the sender and receiver only. Compromised nodes only provide an attacker with access to the measurement data from local nodes. Routing algorithms are also agnostic of the payload protection, thus even nodes that have not established a secure connection can be used to forward packets to a subscriber/destination. A scenario could be in an office building shared by multiple occupants (parties): each party subscribes to a part of the sensor readings only and wishes to keep the data they subscribed to private from other parties, yet they still may share a common communication network in order to reduce cost.

However, an application layer security protocol does not protect routing information. Adversaries can therefore analyze the traffic patterns of a network in clear text. They may even launch a DoS, worm hole, or resource consumption attack that lowers the availability of the network [31]. In this book chapter, the authors focus on end-to-end communication security and rely on other schemes for securing lower communication layers [31].

Scenarios like the one above raise the need for proper authentication of data publishing devices and access control throughout the network. Therefore, an AC was included into the assumed system architecture. The AC is a trusted entity and a more resource-rich server, on which the access rights for the publishers (= motes) of the secured network are stored. The identity of a default subscriber is usually preconfigured on a publisher before it is deployed. If any additional subscribers want to initialize a connection with the publisher, they first have to obtain an access ticket from the AC. The AC verifies that the sub-scriber has the right to access the information available from the publisher. The publisher then only has to evaluate the identity of the subscriber and verify the ticket it has received from the AC. Details of this scenario are subsequently omitted because they are out of scope of this chapter. More details can be found in [26]. This requires a unique identity for a publisher in the network. In the Internet, identities are usually established via PKC and the identifiers provided through X.509 certificates. A X.509 certificate contains, among other information, the public key of an entity and its common

name (*e.g.*, my-bank.com). An example is presented in [32]. A trusted third party - the CA -, which serves two purposes, signs the certificate: Firstly, the signature allows the receiver to detect modifications to the certificate. Secondly, it also states that the CA has verified the identity of the entity that requested the certificate.

Hu et al. showed that RSA, the most commonly used public key algorithm in the Internet, can be used in sensor networks with the assistance of a Trusted Platform Module (TPM), which costs less than 5% of a common sensor node [18]. A TPM is an embedded chip that provides tamper proof generation and storage of RSA keys as well as hardware support for the RSA algorithm. The certificate of a TPM equipped publisher and the certificate of a trusted CA must be stored on the publisher prior to deployment. For publishers that are not equipped with TPM chips an authentication via the DTLS pre-shared key cipher-suite is proposed, which requires a small number of random Byte, from which the actual key is derived, to be preloaded to the publishers before deployment. This secret must also be made available to the AC server, which will disclose the key to devices with sufficient authorization.

For the sake of completeness, it should be mentioned here that an alternative was also implemented using ECC to achieve two-way authentication for resource-less devices. ECC is used for key generation, key exchange, signatures, and encryption. The applied two-way authentication protocol is based on a modified version of the Bellare, Canetti, Krawczyk (BCK) handshake [33], where the modification consists of the usage of pre-shared keys for defense against a man-in-the-middle attack and for additional authorization of different communication parties. For more details it is referred to [8, 27].

1.5 Evaluation

The proposed DTLS solution was evaluated on test-beds including devices of class 1 (*e.g.*, TelosB [34]) and class 2 (*e.g.*, OPAL [35]). The setup was already described in Section 1.3 and the protocol was implemented in TinyOS 2.x [25] with Berkeley Low-power IP stack (BLIP) version 2.0. As pointed out in Section 1.2, the main challenge for two-way authentication solutions is a resource efficient solution that requires only a small part of the available resources, allowing running a reasonable application in addition to the security solution. Thus, the following sections will focus on resource consumption and handshake performance, as well as on a comparison to the related work mentioned in Section 1.2.2 analog to reference [36, 7, 26].

Previous work has already demonstrated techniques to reduce the protocol header overhead during data transmission [21] and has proven the feasibility of performing software encryption and hashing on the sensor node [16], also called mote. Indeed, Raza et al. recently have made first proposals for a compressed header format [23]. Gupta et al. showed the feasibility of a server authenticated SSL handshake [17].

Therefore, the component of the security architecture that is currently least understood in the context of the IoT is the fully authenticated DTLS handshake, which includes both client and server authentication.

We have implemented a DTLS client that performs the DTLS handshake with an OpenSSL 1.0.0d server. The client is targeted at the OPAL sensor node [35], which features an Atmel SAM3U micro-controller [37] and the Atmel AT97SC3203S TPM [38]. It has 48 kByte RAM and the micro-controller is clocked at 48 MHz in the implementation. In the following sections the implementation will be evaluated with regards to its performance during the handshake and data transmission, as well as its energy and memory consumption. Unless otherwise stated, the DTLS cipher suite performed was TLS-RSA-with-AES-128-CBC-SHA. AES-128 has been shown to be one of the fastest block ciphers on motes [39] and offers sufficient security. Furthermore, the selected cipher suite is the required block cipher suite for DTLS from version 1.2 onwards. Other common cipher suites are either based on RC4, which is a stream cipher and thus not permitted by DTLS, or 3DES, which is very slow and as a result causes a large cryptographic overhead.

1.5.1 Data Transfer Latency

In this section the latency as a measure of the systems cryptographic performance is considered. The round-trip time (RTT) for different sizes of plaintext data through a single hop network and a multi hop network with four hops was evaluated as shown in [7]. The timing for the DTLS packets on the mote was measured. Readings for pure plaintext data without any additional headers were obtained by issuing the ping6 command on the subscriber.

A packet sent with both a SHA-1 HMAC and AES-128 encryption is denoted as 'AES-128'. The denotation 'SHA-1' is used if a packet only contained a SHA-1 HMAC. The reading for 8 Byte plaintext data is missing, because the ICMP-Header and the timestamp sent by ping6 are together at least 16 Byte long.

The measurements show a linear increase of round-trip time with jumps occurring approximately every 100 Byte. These spikes can be attributed to the 128 Byte maximum link layer frame size defined by IEEE 802.15.4, which includes header and trailer. These jumps occur earlier when sending DTLS protected packets due to the additional DTLS packet headers, the HMAC size and the explicit Initialization Vector in each packet. See Section 1.4.1 for more details on the packet structure.

Both the increased packet size and processing overhead lead to increased end-to-end transmission latency for DTLS packets compared to plaintext packets (cf. Figure 1.4 [7]). In the single hop scenario, transmission latency was increased by up to 95 ms for AES-128 and up to 75 ms for SHA-1 encryption, which were an average increase of 62% and 35% respectively over the plaintext case. In the multi hop scenario, round trip times increased by a maximum of 163 ms and were 74% longer on average for AES-128 encrypted packets. Packets with a SHA-1 HMAC took up to 129 ms longer for the round-trip with an average of 40% more time being

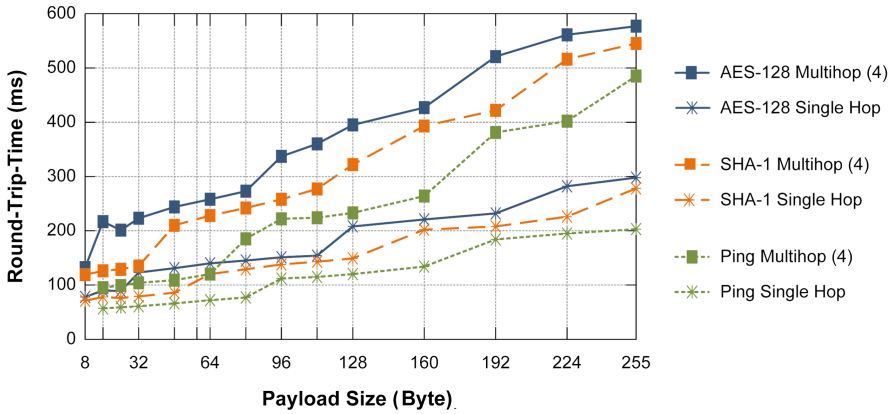

Fig. 1.4 Average (n=100) packet round-trip time for different plaintext sizes.

spent. The decreased performance for transmission latency is mostly due to the large packet overhead of up to 64 Byte, which consists of 13 Byte DTLS record header, 16 Byte Initialization Vector, 20 Byte HMAC, and up to 15 Byte padding. Calculating a SHA-1 hash of a 255 Byte plaintext message only takes 9 ms, encryption with AES-128 takes another 12 ms. Both operations do not contribute significantly to the overall transmission latency. This is consistent with the measurements for 16 Byte plaintext (RTT of 58 ms), which increases to 90 ms with AES-128. Including the overhead of the DTLS record format, 16 Byte plaintext are expanded to a 77 Byte message. Sending 80 Byte via ping requires 78 ms, which indicates a computational overhead of around 12 ms in this case. A more detailed analysis of the transmission overhead from an energy perspective is provided in Section 1.5.4.

1.5.2 Handshake Latency

Another performance indicator to consider is the latency introduced by performing a DTLS handshake. The time from the beginning of the establishment of the handshake was measured until a *Finished* message has been received on the client. In addition to using a 2048-bit key, the results for a 1024-bit key for comparison was included. Figure 1.5 shows the average latency for a fully authenticated and a server-authenticated handshake [7]. For each type of handshake 15 measurements were conducted. The bars show the average over these measurements, and the error bars show the standard deviation.

The large standard deviation is caused by the presented implementation behavior when message loss occurs. DTLS states that an implementation should wait for an answer for a set amount of time after sending a flight of messages. If it does not receive an answer during this period, it retransmits the whole flight. This timeout value was set to 5 seconds to avoid unnecessary retransmissions in networks with a

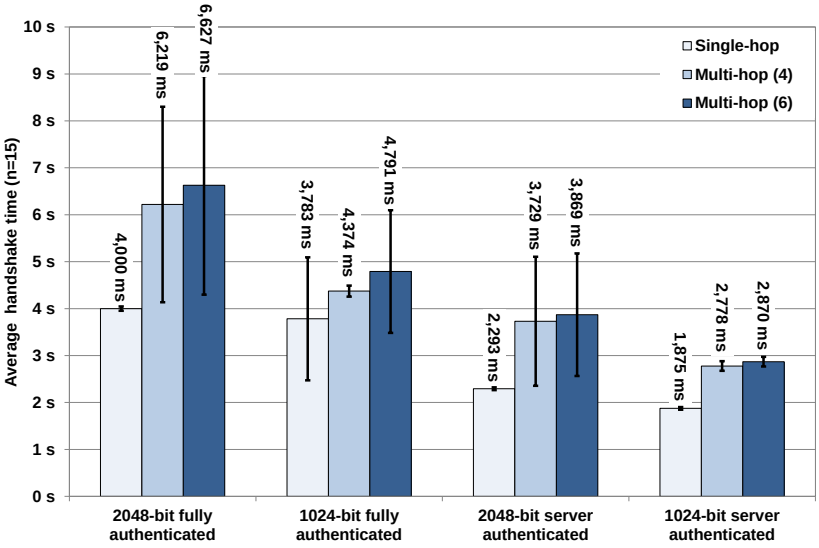

Fig. 1.5 Time to Complete Different Types of DTLS Handshakes.

high end-to-end delay, which is common in a low power lossy network, and/or with energy limited thin clients that are slow to respond. DTLS implementations for the Internet often choose a retransmission timeout of 1 second or less. In general, it can be seen that the time to execute a handshake is shorter for smaller RSA keys and reduced by almost 2 seconds when client authentication is omitted in the handshake. It was observed that packet loss occurred mainly in a multi-hop environment and when larger DTLS messages were being sent. This increases the total handshake time significantly because of the large DTLS retransmission timeout. However, total energy consumption of the client does not increase significantly, because all TPM operations, which are the largest contributor to overall handshake energy costs, are only executed after successful receipt of all relevant server messages. Losing a packet with information obtained from the TPM does not lead to a repeated execution of the TPM operations because the resulting messages are buffered and can be retransmitted. During the experiments no failed handshake attempts where recognized.

DTLS requires successful transmission of all handshake packets over an unreliable transport layer. Since it provides its own reliability mechanism during the handshake, network topology, congestion and link quality have a large impact on the time needed to complete a DTLS handshake. One parameter the programmer can influence to achieve better performance in lossy networks is the maximum transmission unit (MTU) for DTLS handshake packets, which determines the size of individual handshake packet fragments. To study the influence of the MTU on overall handshake establishment time a random, artificial packet drop rate was introduced on the link layer and measured handshake completion times for various MTUs.

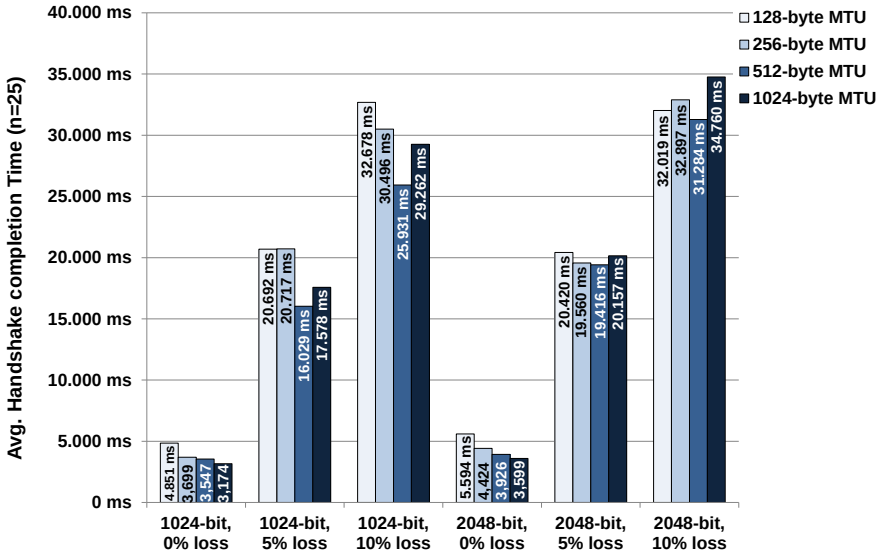

Fig. 1.6 Handshake Completion Times With Various Amounts of Artificial Link Layer Packet Loss and Different MTUs.

Figure 1.6 shows that even a small amount of packet loss has a large impact on overall handshake completion time [7]. It was considered that each link layer packet has an independent chance of being dropped, resulting in the total loss of all packets that follow. The probability for a packet loss is defined by $P(Packetloss) = 1 - PLR^{\lceil \frac{\text{traffic in Byte}}{100\ \text{Byte}} \rceil}$, where PLR is the packet loss rate. If a typical, fully authenticated DTLS handshake was taken, which causes 2,438 Byte of traffic as an example, there is a 72.26% chance of packet loss while transmitting the 2,438 Byte of handshake payload at 5% link layer packet loss. If the link layer packet loss rate is 10%, there is a 92.82% chance of packet loss occurring. In that case, the DTLS reliability mechanism is waiting for a timeout before resending the whole message flight [30]. As before, the retransmission timer was set to 5 seconds during the experiments.

The authors of this book chapter are considering uncorrelated packet loss in this evaluation; even tough packet loss is correlated in reality. The reasoning behind these figures is that it was unknown at which time during the handshake the interference that causes packet loss will start. Therefore, a constant probability of packet loss was used, which will cause all following fragments of the current message flight to be dropped. Additional, correlated packet loss before the next retransmission interval has no adverse impact, because the damage is already done.

The MTU influences the granularity at which handshake the receiver can reassemble messages. A small MTU splits large handshake messages into many different packets, allowing the receiver a fine-grained reassembly if packets are lost. Since every new packet has to bear the DTLS header, the overall amount of traffic

increases, which in turn increases the probability of packet loss. A larger MTU splits messages into fewer packets, which reduces the probability of packet loss because there is less network traffic. However, if packet loss does occur, reassembly cannot be done as fine-grained as with a smaller MTU. Figure 4 shows that a MTU of 512 Byte seems to strike the best balance between reassembly and network traffic in the experiments.

1.5.3 Memory Consumption

In order to determine the static memory allocation of individual components of the presented implementation, the entries in the symbols table of the OPAL binary after compilation were analyzed. Memory has been measured for a fully authenticated handshake with 2048-bit RSA keys. This type of handshake has the largest memory requirements, because it needs more code and buffer space for the client's *Certificate* and *CertificateVerify* messages. The memory consumption was divided into six, respectively seven categories as illustrated in Table 1.1 [7]. Additionally, the maximum stack size was measured by filling the stack with a dummy variable directly after booting and analyzing how much of that continuous memory block had been overwritten after a successful DTLS handshake. Only the first subtotal of Table 1.1 [7]. considers static memory allocation. Because it currently contributes a significant portion of overall stack use, two prototypical methods of initializing the client certificate were implemented. The method represented by Stack Minimum directly sets each individual Byte of the outgoing message buffer to the matching value from the Certificate. The drawback is a increased ROM use, because the code basically contains hundreds of statements in the form $buffer[x] = 0xff$. The "Stack Maximum" method initializes the outgoing message buffer from a temporary array which is filled from a hardcoded, anonymous array, *e.g.*, $uint8_t[CERT_LEN] = \{0xff, 0xff, 0xff, ...\}$. In production the certificate would usually be read from the mote's flash memory, which should fall somewhere in between the figures from these two approaches.

In total, approximately 20 kByte of RAM and 67 kByte of ROM is required for the implementation. The BLIP implementation requires most of the resources, followed by TPM drivers and DTLS networking code. Overall, the implementation is still below the 48 kByte of RAM and 256 kByte of program memory provided by OPAL [7, 26].

Table 1.1 RAM and ROM usage by component.

	RAM (Byte)	ROM (Byte)
Cryptography	541	10,838
DTLS Messages	1,174	2,568
DTLS Network	4,294	5,672
TPM	4,321	4,928
BLIP	6,352	9,298
Application	166	-
System	991	30,075
Total Data + BSS	**17,839**	**63,379**
Stack Minimum	1,098	0
Stack Maximum	2,300	3,936
Total	**18,937 - 20,139**	**63,379 - 67,315**

Table 1.2 Transaction time / energy consumption of DTLS handshake (2048-bit key).

	Current	Fully authenticated handshake	Server-authenticated handshake
Computation	30 mA	35 ms, 4.18 mJ	33 ms, 3.95 mJ
Radio TX	18 mA	242 ms, 17.4 mJ	70 ms, 5.03 mJ
TPM Start	52.2 mA	836 ms, 174.46 mJ	836 ms, 174.5 mJ
TPM TWI	43.6 mA	688 ms, 120.0 mJ	476 ms, 83.0 mJ
TPM Verify	51.8 mA	59 ms, 12.2 mJ	56 ms, 11.6 mJ
TPM Encrypt	51.8 mA	39 ms, 8.07 mJ	40 ms, 8.28 mJ
TPM Sign	52.2 mA	726 ms, 151.5 mJ	-
Total minimum		487.8 mJ	286.4 mJ
CPU idle	11.4 mA	3965 ms, 180.7 mJ	2265 ms, 103.2 mJ
Radio idle	18 mA	3758 ms, 270.4 mJ	2228 ms, 160.3 mJ
Total		939.0 mJ	549.9 mJ

1.5.4 Energy Consumption

The authors measured the energy consumption during the handshake phase across a 10Ω resistor with an oscilloscope. It yielded a value for the electric potential, which can be converted into a value for the current draw by dividing it through the value of the resistance (10Ω).

The energy costs can then be calculated as $U_{probe}/R * t * U_{battery}$. U_{probe} is the measured voltage, $R = 10\Omega$ is the value of the resistor, t is the transaction time, and $U_{battery} = 3.998V$ is the battery voltage. Table 1.2 shows the energy consumption during a typical execution of different handshake types [7]. A 2048-bit RSA key was used, because 1024-bit keys are not recommended for future deployments [40].

The contribution of the radio and micro-controller are neglected in further discussion. Both can be considerably reduced by using power saving techniques, *e.g.*, by using the TinyOS Low Power Listening (LPL) Media Access Control layer for the radio (less than 1% radio duty cycles have been reported by the literature repeatedly) and setting the micro-controller into a lower power state where it consumes less than 15μA for SAM3U [37]. However, the transmission costs of messages increase sig-

nificantly if LPL is activated. This tradeoff is subject to the design and configuration of each deployed network. For better comparison the idle energy use as outside of the developers' field of control was viewed and the focus was set on energy costs, which occurred in any case. Sending messages and performing cryptographic operations contribute very little (17.4 mJ and 4.18 mJ, respectively) to the overall energy costs that are directly dependent on the presented DTLS implementation. The total cost is then largely bound by the use of energy of the TPM.

Figure 1.7 shows the measured draw of the TPM chip [7]. "TPM Start" and "TPM Sign" are the longest consecutive operations, which consume 174.46 mJ and 151.5 mJ. The TPM is performing an operation with its RSA private key in "TPM Sign", which is more complex than using a RSA public-key. During the "TPM Start" phase the TPM performs a series of internal self-tests to detect tampering and unauthorized commands. The second large block is "TPM TWI" which describes the amount of time that is spent passing data to the TPM and receiving data from it via the TWI bus clocked at 100 kHz, consuming 120 mJ. It shows as a lower current draw. It can be recognized that directly after the end of the "TPM Start" sequence and before the short spike in "TPM Verify". The spike is the actual verification operation performed by the TPM, which consumes only 12.2 mJ. Similarly, the actual "TPM Encrypt" operation is the spike that follows another section of data transfer on the TWI bus, consuming 8.07 mJ. During "TPM Verify" the TPM uses the stored key of a CA to verify the server certificate presented during the handshake.

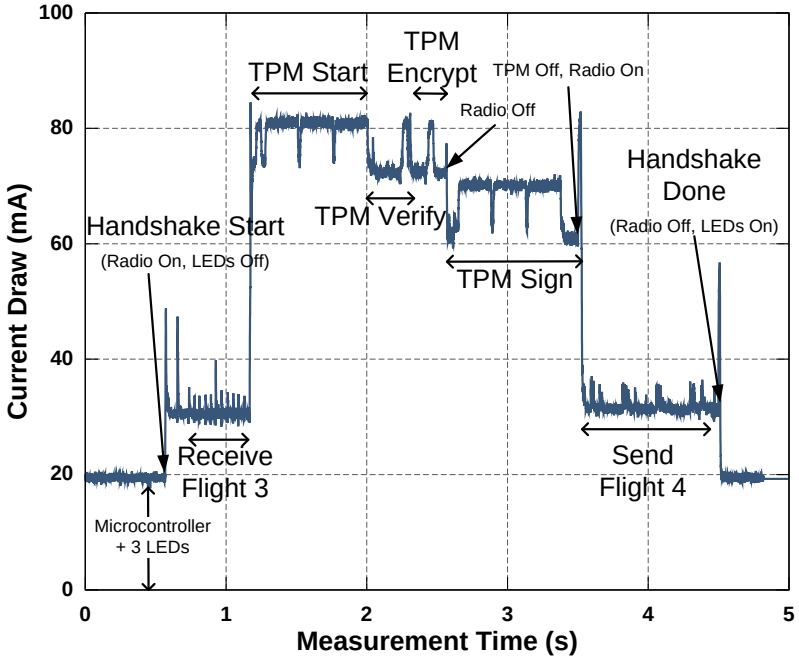

Fig. 1.7 Current draw for a fully authenticated DTLS handshake.

The "TPM Encrypt" operation is used to encrypt a nonce with the server's public key. If the mote is expected to authenticate itself during the handshake, it performs a "TPM Sign" operation to sign a hash over all previous handshake messages with its RSA private key. Since a server-authenticated handshake does not require the expensive "TPM Sign" operation, it uses significantly less energy but also provides weaker overall authentication because an attacker could impersonate a mote toward the server. Communication time is also shorter since the sensor node does not send its certificate. [7, 26]

If two AA 280-mAh batteries power the mote, they contain approximately 30,240 Joule of energy. If 5% of the energy is used for DTLS handshakes for (re)keying purposes, which happen once per day, it could last for more than 8.5 years for a fully authenticated handshake at 487.8 mJ each, or more than 14.5 years for a server-authenticated handshake at 286.4 mJ each. As stated earlier, the calculation of a SHA-1 hash for 255 Byte takes 9 ms and encryption with AES-128 another 12 ms. Given the current draw for computation of 30 mA at 48 MHz clock speed from Table 1.2, this results in the order of 9.9μJ per Byte. [7, 26]

The energy consumption after the completion of the handshake is closely related to the latency values from Figure 1.5, which portraits the influence of the network and processing overhead introduced by DTLS [7]. The increase in latency naturally also leads to an increase in energy consumption, because the radio has to be held in the transmitting state for a longer time in order to prevent it from entering a sleep state. Figure 1.8 shows the overhead in percent that occurs when a plaintext of a given size is encrypted and sent in a secure DTLS record [7]. The baseline for this comparison is the time it would take to send the plaintext without any additional headers or other meta data.

It is assumed that the energy cost to send a message with length x via BLIP follows a discontinuous piecewise linear function: $c(x,a,b) = \lceil \frac{x}{100} \rceil * a + x * b$. Here, a represents the amount of energy needed to access the medium for one IEEE 802.15.4 message and sending the preamble and all other fixed energy costs for one message. The energy required for transmitting one Byte of payload without fixed costs is represented by b. The constant 100 is the maximum link layer message length defined by BLIP. Since the relative overhead was only interesting for the authors, the current draw was ignored and only the relation between message length and time was analyzed. For this purpose the round-trip times were used as measured in Figure 1.5 for a simple ping and divided them by two [7]. Matlab was used to find the minimum of the error function $err(a,b) = \sum_{x \in M} \| \frac{c(x,a,b) - t(x)}{x} \|$ where M is the set of plaintext lengths for which we have obtained measurement times and $t(x)$ returns the measured time for a plaintext length x. This optimization returned $a = 27.368$ and $b = 0.072$. With these results the approximate time could be calculated that was required to send plaintext and larger DTLS records for the same amount of plaintext.

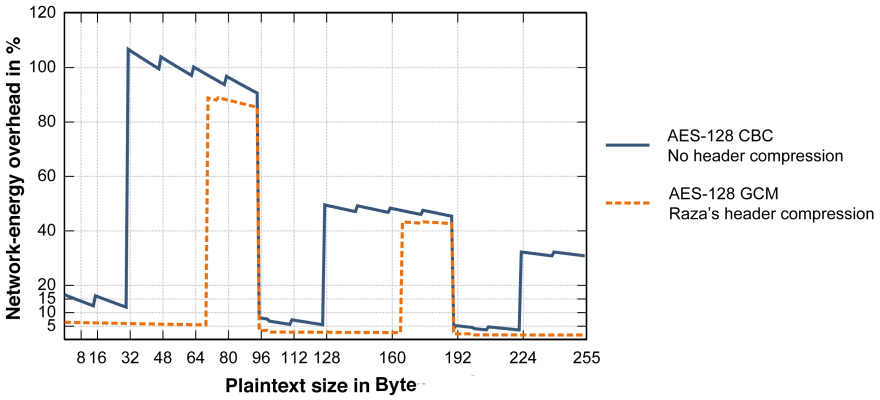

Fig. 1.8 Network Energy Overhead Caused by the DTLS Record Format.

Figure 1.8 shows that the overhead introduced by the DTLS record format is under 17% for small plaintext lengths. It rises to over 100% when the DTLS record will not fit into a single link-layer packet anymore [7]. BLIP then has to fragment the packet and bear the expensive medium access a second time. One way to reduce the network overhead is reducing the size of DTLS records. The proposal is to employ the header compression detailed by [23]. It reduces the size of a DTLS record header from 13 to 5 Byte. Further savings are possible if the block cipher mode of operation is changed from CBC to Galouis/Counter mode of operation (GCM). The plaintext encrypted by GCM will always lead to a cipher text of the same length [41]. Since GCM belongs to the class of block cipher modes called Authenticated Encryption with Associated Data (AEAD) the SHA-1 HMAC is no longer necessary. Instead, GCM can be used directly to authenticate the data and associated headers. The maximum length GCM auth tag, which requires 16 Byte, thus replaces the 20-Byte SHA-1 HMAC. Additionally, the explicit IV is no longer necessary, because GCM is not susceptible to the vulnerability that makes the IV necessary. The maximum DTLS record overhead can thus be reduced from 64 Byte down to 21 Byte: Five Byte for the compressed record header plus the 16 Byte GCM auth tag. Figure 1.8 shows that this more than doubles the area in which a DTLS record only incurs little overhead over sending the plaintext directly [7].

In order to put the TPM energy consumption and processing time in context, measurements of RSA and ECC were also performed. The RSA and ECC TinyOS modules available did not support 2048-bit RSA keys or their respective ECC equivalent. Thus, the authors ported the RSA and ECC implementation of the open source project CyaSSL [42] to TinyOS. This port includes many of the optimization techniques adopted in TinyECC [43], such as Barrett Reduction, Sliding Window multiplication, Shamir's Trick and others. It does not, however, include inline assembly instructions to speed up natural number operations. The presented implementation [44] is made available to the TinyOS community under the GPLv2 license [45].

Table 1.3 Software RSA (2048-bit key) on OPAL. One Private Key and two Public Key operations are required for a handshake.

	Current	Computation time	Energy consumption
RSA - Public Key @ 48 MHz	30 mA	440 ms	52.8 mJ
RSA - Private Key (high memory) @ 48 MHz	30 mA	4,725 ms	566.7 mJ
RSA - Private Key (low memory) @ 48 MHz	30 mA	14,895 ms	1,786 mJ
Handshake RSA total @ 48 MHz	**30 mA**	**5,165 ms**	**619.5 mJ**
RSA - Public Key @ 96 MHz	48 mA	221 ms	42.4 mJ
RSA - Private Key (high memory) @ 96 MHz	48 mA	2,362 ms	453.3 mJ
RSA - Private Key (low memory) @ 96 MHz	48 mA	7,447 ms	1,429 mJ
Handshake RSA total @ 96 MHz	**48 mA**	**2,583 ms**	**495.7 mJ**

Table 1.4 Software ECC over 224-bit prime curve (secp224r1) on OPAL. One of each operation is required for a handshake.

	Current	Computation time	Energy consumption
EC-DH @ 48 MHz	30 mA	387 ms	46.4 mJ
ECDSA sign @ 48 MHz	30 mA	432 ms	51.8 mJ
ECDSA verify @ 48 MHz	30 mA	795 ms	95.4 mJ
Handshake ECC total @ 48 MHz	**30 mA**	**1,614 ms**	**193.6 mJ**
EC-DH @ 96 MHz	48 mA	187 ms	35.8 mJ
ECDSA sign @ 96 MHz	48 mA	205 ms	39.3 mJ
ECDSA verify @ 96 MHz	48 mA	380 ms	72.9 mJ
Handshake ECC total @ 96 MHz	**48 mA**	**772 ms**	**92.6 mJ**

Table 1.3 shows the results for individual RSA operations with a 2048-bit RSA key performed in software [7]. The figures for the handshake only pertain to the DTLS client, as was the case in the previous evaluations.

With a clock speed of 48 MHz, the software implementation requires more than twice as much time as the TPM and almost 1.5 times the amount of energy. The respective values for the TPM where 2348 ms and 466.2 mJ. This advantage is demised when the TPM is compared to software RSA being performed at 96 MHz, where both require roughly the same amount of time and energy. The RSA implementation still has room for improvement through embedded Assembler code and could thus be made more time and energy efficient than the TPM on the chosen platform. However, the TPM still provides secure storage of the RSA key, which cannot be achieved by software means, and the implementation complexity and RAM requirements of the TPM drivers are far less than those of software RSA implementation. Additionally, newer versions of the available TPM chip have more than halved the computation time for 2048-bit RSA keys.

If secure storage of a motes private key is not a design goal, a software implementation of ECC is recommended instead. As Table 1.4 shows, it requires far less time and energy than either solution for RSA [7]. The figures given were computed over the NIST named curve secp224r1, also known as [46]. It provides equivalent security to a 2048-bit RSA key.

The operations performed during the DTLS handshake are Elliptic Curve Diffie-Hellman (ECDH) for key-agreement followed by a two-way authentication via the Elliptic Curve Digital Signature Algorithm (ECDSA) to avoid Man-in-the-Middle attacks.

1.5.5 Comparison to Related Work

The existing implementation in [36] shows that a DTLS client can be implemented in TinyOS for constrained devices. The use of RSA in the handshake, while being the most widespread public key crypto system in the Internet, basically constitutes the worst-case scenario: RSA keys are very large in size and have high computational demands. Nevertheless, the authors of this book chapter demonstrated that a DTLS handshake with a RSA cipher suite is feasible in WSNs. It was proven that devices are successful in completing such a hand-shake even if packet loss occurs while only requiring a moderate amount of energy for each handshake (under 500 mJ). Handshakes using other public key crypto systems, such as ECC, can therefore be seen as an easy replacement of the basic RSA handshake, because they offer equivalent security at considerably shorter key lengths [8]. However, the authors are not aware of equivalent devices to TPMs that offer secure storage and hardware accelerated computations with ECC keys. Therefore, the implementation for the OPAL mote is superior to ECC based implementations, in that regard, because the authors make use of the Opals TPM for RSA computations and key storage. Although they think that DTLS is a feasible choice for an end-to-end security protocol, there is still room for improvement, which will be described in the next Section. Additionally, DTLS can-not be used to protect routing information and to guard against attacks on a network's availability. Only link layer protocols are able to achieve these goals to a certain degree (cf. Section 1.2.2). Therefore, the authors acknowledge a need for link layer security in WSNs. In use cases like the Internet of Things, link layer and application layer security may complement each other: An application layer security protocol can protect the integrity and confidentiality of application layer messages sent in the local WSN while an additional link layer security protocol can protect the authenticity of the routing information in the local network. In this scenario, only authenticity is needed on the link layer because confidentiality of messages is achieved on the application layer. If messages encrypted with the application layer security protocol leave the local network and enter the Internet they are still protected by encryption, which a link layer protocol cannot achieve alone. Therefore, the presented solution is only compared to other work on application layer and network layer security protocols in WSNs:

Sizzle [17] and SSNAIL [20] both provide SSL or TLS application level security over a reliable transport protocol with 80 bits of security during the handshake through 160-bit ECC keys or 1024-bit RSA keys. The presented DTLS implementation does not require reliable transport, such as TCP, but supports unreliable trans-

port via UDP. Datagram transport is often preferable in WSNs, because it introduces fewer overheads and is more efficient. Additionally, the authors offer 112-bits of security through 2048-bit RSA keys, which are recommended until the year 2030. 80-bits of security were only recommended until 2010 [40]. Neither of these implementations supports secure storage of private keys in a TPM or similar device. If ECC is used, both, Sizzle and SSNAIL, need about one second to perform a handshake and they need between 6 and 7.5 seconds when using RSA. It is comparable to the handshakes performance (about 4 seconds) without the influence of packet loss. However, both implementations require a lot less memory than the presented implementation in this book chapter: Between 2.8 kByte of RAM for Sizzle and 6.3 kByte of RAM for SSNAIL versus the presented solution requirement of 17.2 kByte RAM, which is largely caused by the additional buffer space required for the much larger RSA certificates.

Tiny 3-TLS [47] is different from the presented implementation, because it uses a trusted gateway for trust delegation. In the DTLS implementation, the sensor nodes are independent from a central gateway. The authors of this book chapter cannot directly compare it against performance figures of Tiny 3-TLS. However, if the gateway is fully trusted, the nodes need to perform only symmetric operations, which are much faster than asymmetric operations. If the gateway is only partially trusted, they perform a Diffie-Hellman operation that requires modular exponentiation and may therefore also be resource intensive.

IPsec [48] is independent of the transport layer, because it is located on the network layer. It therefore supports reliable as well as unreliable transport. Raza et al. did not implement the Internet Key Exchange (IKE) protocol, but in-stead deployed the key material prior to their experiments. Their RAM requirements are therefore not comparable to the authors because the handshake, which is a large contributor to memory usage in the DTLS implementation, is not present in theirs. The network overhead of compressed IPsec for payload packets is only 24 Byte, whereas the network overhead of DTLS is currently still up to 64 Byte.

1.6 Summary

This book chapter introduced a standards-based security architecture with two-way authentication for IoT. The DTLS-based solution presented was developed for resource-full devices. Furthermore, a list of important security issues must be kept in mind, when developing an appropriate solution for two-way authentication. The solution for a two-way authentication was described in detail and evaluated in order to justify its applicability for resource-full constrained devices.

In this solution for resource-full devices the authentication is performed during a fully authenticated DTLS handshake and based on an exchange of X.509 certificates

containing RSA keys. The extensive evaluation, based on real IoT systems, shows that the proposed architecture (cf. Figure 1.1) provides message integrity, confidentiality, and authenticity with affordable energy, end-to-end latency, and memory overhead. Thus, a DTLS is a feasible security solution for emerging IoT. A fully authenticated handshake with strong security through 2048-bit RSA keys is considered as feasible for sensor nodes equipped with a TPM chip, because a fully authenticated, RSA-based handshake consumes as little as 488 mJ. The memory requirement of fewer than 20 kByte RAM are well below the 48 kByte of memory offered by the used sensor node. Sensor nodes without a TPM chip forego protection against physical tampering, but can still perform a DTLS handshake based on ECC, which could be performed on the chosen platform with little more than 100 mJ of energy consumption.

A similar solution for resource-less constrained devices was implemented supporting also two-way authentication but not working with DTLS and certificates instead using ECC to save resources. This solution achieves the same security goals as the presented DTLS-based solution but is more resource-efficient. For more information it is referred to [8, 27]. Other work has demonstrated techniques to minimize packet headers for similar protocols [21]. For the future, it is planned that these techniques apply to DTLS together with an Authenticated Encryption with the Associated Data (AEAD) mode of operation to achieve a reduction of network overhead.

Acknowledgements The DTLS solution presented was supported partially by the German Federal Ministry of Education and Research: the SODA Project under Grant Agreement No. 01IS09040A and the AutHoNe Project under Grant Agreement No. 01BN070[25]. The standardization activity within IETF was supported partially by FLAMINGO and SmartenIT, funded by the EU FP7 Program under Contract No. FP7-2012-ICT-318488 and No. FP7-2012- ICT317846, respectively.

References

1. H. LeHong, “Hype Cycle for the Internet of Things,” Gartner Inc., Stanford, U.S.A., Tech. Rep., 2012.
2. I. Leontiadis, C. Efstratiou, C. Mascolo, and J. Crowcroft, “SenShare: Transforming Sensor Networks Into Multi-application Sensing Infrastructures,” in *Wireless Sensor Networks*, ser. LNCS, G. Picco and W. Heinzelman, Eds., vol. 7158. Berlin, Heidelberg, Germany: Springer, 2012, pp. 65–81.
3. European Telecommunications Standards Institute (ETSI), “Machine-to-Machine Communications (M2M), Smart Metering Use Cases,” ETSI, Sophia Antipolis Cedex, France, Tech. Rep. ETSI TR 102 691 V1.1.1 (2010-05), 2010.
4. Z. Shelby, K. Hartke, and C. Bormann, “The Constrained Application Protocol (CoAP),” Internet Engineering Task Force (IETF), Fremont, California 94538 Fremont, California, U.S.A., Tech. Rep. RFC 7252, June 2014.

5. Z. Shelby and B. C., *6LoWPAN: The Wireless Embedded Internet*, ser. Wiley Series in Communications Technology, D. Hutchison, S. Fdida, and J. Sventek, Eds. Chichester, West Sussex, U.K.: Wiley, 2009.
6. S. Dawson-Haggerty, A. Tavakoli, and D. Culler, "Hydro: A Hybrid Routing Protocol for Low-power and Lossy Networks," in *1st IEEE International Conference on Smart Grid Communications*, ser. SmartGrid-Comm. New York, New York, U.S.A.: IEEE, 2010, pp. 268–273.
7. T. Kothmayr, C. Schmitt, M. Hu, W. Brünig, and G. Carle, "DTLS Based Security and Two-Way Authentication for the Internet of Things," in *Ad Hoc Networks*, vol. 11, no. 8. Philadelphia, Pennsylvania, U.S.A.: ELSEVIER, 2013, pp. 2710–2723.
8. M. Noack, "Optimization of Two-way Authentication Protocol in Internet of Things," Master's thesis, University of Zurich, Zurich, Switzerland, August 2014.
9. M. Bellare, R. Canetti, and H. Krawczyk, "Keyed Hash Functions and Message Authentication," in *Advances in Cryptology - CRYPTO 96,*, ser. LNCS, N. Knoblitz, Ed., vol. 1109. Berlin, Germany: Springer, 1996, pp. 1–15.
10. L. Atzori, A. Iera, and G. Morabito, "The Internet-of-Things: A Survey," *ELSEVIER Journal of Computer Networks*, vol. 54, no. 15, pp. 393–422, 2010.
11. D. Miorande, S. Siciari, F. Pellegrini, and I. Chlamtac, "Internet-of-Things: Vision, Applications, and Research Challenges," *Ad Hoc Networks*, vol. 10, no. 7, pp. 1497–1516, 2012.
12. H. Karl and A. Willig, *Protocols and Architectures for Wireless Sensor Networks*. New York, New York, U.S.A.: John Wiley and Sons, 2005.
13. C. Bormann, M. Ersue, and A. Keranen, "Terminology for Constrained-Node Networks," Internet Engineering Task Force (IETF), Fremont, California, U.S.A., Tech. Rep. RFC 7228, May 2013.
14. I. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "Wireless Sensor Networks: A Survey," *Elsevier Journal of Computer Networks*, vol. 38, no. 4, pp. 393–422, 2002.
15. D. Raymond and S. Midkiff, "Denial-of-service in Wireless Sensor Networks: Attacks and Defenses," *Journal Pervasive Computing*, vol. 7, no. 1, pp. 74–81, 2008.
16. M. Luk, G. Mezzour, A. Perrig, and V. Gligor, "MiniSec: A Secure Sensor Network Communication Architecture," in *6th International Conference on Information Processing in Sensor Networks*, ser. IPSN. New York, New York, U.S.A.: ACM, 2007, pp. 470–488.
17. V. Gupta, M. Wurm, Y. Zhu, M. Millard, S. Fung, N. Gura, H. Eberle, and S. Shantz, "Sizzle: A Standards-based End-to-end Security Architecture For The Embedded Internet," *ELSEVIER Pervasive and Mobile Computing*, vol. 1, no. 4, pp. 425–445, 2005.
18. W. Hu, H. Tan, P. Corke, W. Shih, and S. Jha, "Toward Trusted Wireless Sensor Networks," *ACM Transactions on Sensor Networks*, vol. 7, no. 1, pp. 1–25, 2010.
19. H. Chan, A. Perrig, and D. Song, "Random Key Predistribution Schemes For Sensor Networks," in *Symposium on Security and Privacy*. Washington, U.S.A.: IEEE Computer Society, 2003, pp. 197–213.
20. W. Jung, S. Hong, M. Ha, Y.-J. Kim, and D. Kim, "SSL-based Lightweight Security of IP-based Wireless Sensor Networks," in *International Conference on Advanced Information Networking and Applications Workshops*. New York, New York, U.S.A.: IEEE, 2009, pp. 1112–1117.
21. S. Raza, S. Voigt, and U. Rödig, "6LoWPAN Extension for IPsec," in *IETF-IAB International Workshop on Interconnecting Smart Objects with the Internet*. Fremont, California, U.S.A.: IETF-IAB, 2011.
22. S. Raza, T. Voigt, and V. Jutvik, "Lightweight IKEv2: A Key Management Solution For Both The Compressed IPsec and The IEEE 802.15.4 Security," in *IETF Workshop on Smart Object Security*. Fremont, California, U.S.A.: Internet Engineering Task Force (IETF), 2012.
23. S. Raza, D. Trabalza, and T. Voigt, "6LoWPAN Compressed DTLS for CoAP," in *8th IEEE International Conference on Distributed Computing in Sensor Systems*. New York, New York, U.S.A.: IEEE, 2012, pp. 287–289.
24. T. Winter, P. Thubert, A. Brandt, J. Hui, P. Kelsey, K. Levis, P. K., R. Struik, J. Vasseur, and R. Alexander, "RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks," Internet

Engineering Task Force (IETF), Fremont, California, U.S.A., Tech. Rep. RFC 6550, March 2012.
25. "TinyOS Homepage," Online, October 21, 2015. [Online]. Available: http://www.tinyos.net
26. C. Schmitt, "Secure Data Transmission in Wireless Sensor Networks," Ph.D. dissertation, Technische Universität München, Institut für Informatik, Munich, Germany, July 2013.
27. C. Schmitt, B. Stiller, and M. Noack, "Two-way authentication for iot," Internet Engineering Task Force (IETF), Fremont, California, U.S.A., Internet Draft draft-schmitt-ace-twowayauth-for-iot-02, June 2015.
28. D. Cooper, S. Santesson, S. Farrell, S. Boeyed, R. Housley, and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile," Internet Engineering Task Force (IETF), Fremont, California, U.S.A., Tech. Rep. RFC 5280, May 2008.
29. R. Watro, D. Kong, S. Cuti, C. Gardiner, C. Lynn, and P. Kruus, "TinyPK: Securing Sensor Networks With Public Key Technology," in *2nd ACM Workshop on Security of Ad Hoc and Sensor Networks*, ser. SASN. New York, New York, U.S.A.: ACM, 2004, pp. 59–64.
30. N. Modadugu and E. Rescorla, "The design and implementation of datagram tls," in *Network and Distributed System Security Symposium*, ser. NDSSS. Reston, U.S.A.: The Internet Society, 2004.
31. P. Ning, A. Liu, and W. Du, "Mitigating DoS Attacks Against Broadcast Authentication in Wireless Sensor Networks," *ACM Transactions on Sensor Networks*, vol. 4, no. 1, pp. 1–35, January 2008.
32. C. Schmitt, T. Kothmayr, B. Ertl, W. Hu, B. L., and G. Carle, "TinyIPFIX: An Efficient Application Protocol For Data Exchange in Cyber Physical Systems," *Computer Communications*, vol. 74 (2016), pp. 63–76, January 2016.
33. S. Blake-Wilson and M. A., "Authenticated Diffie-Hellman Key Agreement Protocols," in *Selected Areas in Cryptograph*, E. Stafford and M. H., Eds. London, U.K.: Springer, 1999, pp. 339–361.
34. ADVANTIC SISTEMAS Y SERVICIOS S.L.,, "Datasheet TelosB - CM5000-SMA," http://www.advanticsys.com/shop/mtmcm5000sma-p-23.html, October 27, 2015.
35. R. Jurdak, K. Klues, B. Kusy, C. Richter, K. Langendoen, and M. Brünig, "OPAL: A Multiradio Platform For High Throughput Wireless Sensor Networks," *IEEE Embedded Systems Letters*, vol. 3, no. 4, pp. 121–124, 2011.
36. T. Kothmayr, "A Security Architecture for Wireless Sensor Networks based on DTLS," Master's thesis, Technische Universität München, Institut für Informatik, Munich, Germany, 2011.
37. ATMEL, "ATMEL - Datasheet SAM3U Series," http://www.atmel.com/Images/doc6430.pdf, October 27, 2015.
38. A. Cooperation, "AT97SC3203 The ATMEL Trusted Platform Module," http://www.atmel.com/images/doc5128.pdf, 2007.
39. J. Grossschaedl, S. Tillich, C. Rechberger, M. Hofmann, and M. Medwed, "Energy Evaluation of Software Implementations of Block Ciphers Under Memory Constraints," in *Conference on Design, Automation and Test in Europe*. San Jose, U.S.A.: EDA Consortium, 2007, pp. 1110–1115.
40. E. Barker, W. Barker, W. Burr, W. Polk, and M. Smid, "NIST SP800-57: Recommendation for Key Management – Part 1: General (Revised)," National Institute of Standards and Technology, Gaithersburg, MD, U.S.A., Tech. Rep., 2007.
41. D. McGrew and J. Viega, "The Galois/Counter Mode of Operation (GCM), NIST Modes Operation Symmetric Key Block Ciphers," National Institute of Standards and Technology, Gaithersburg, MD, U.S.A., Tech. Rep., 2005.
42. wolfSSL Inc., "wolfSSL Embedded SSL Library (formerly CyaSSL)," http://www.yassl.com/yaSSL/Products-cyassl.html, October 27, 2015.
43. A. Liu and P. Ning, "TinyECC: A Configurable Library For Elliptic Curve Cryptography in Wireless Sensor Networks," in *5th International Conference on Information Processing in Sensor Networks*, ser. ISPN. Washington, U.S.A.: IEEE Computer Society, 2008, pp. 245–256.

44. T. Kothmayr and C. Schmitt, "TinyPKC Implementation," https://corinna-schmitt.de/tinypkc.html, October 27, 2015.
45. GNU, "Gplv2 license," http://www.gnu.org/licenses/gpl-2.0.html, October 27, 2015.
46. NIST, "NIST P-224: Recommended Elliptic Curves For Federal Government Use," National Institute of Standards and Technology (NIST), Washington, U.S.A., Tech. Rep., July 1999.
47. S. Fouladgar, B. Mainaud, K. Masmoudi, and H. Afifi, "Tiny 3-TLS: A Trust Delegation Protocol for Wireless Sensor Networks," in *Proceedings of the Third European Conference on Security and Privacy in Ad-Hoc and Sensor Networks (ESAS)*, ser. LNCS, L. Buttyan, V. D. Gligor, and D. Westhoff, Eds., vol. 4357. Berlin, Heidelberg, Germany: Springer, 2006, pp. 32–42.
48. S. Raza, T. Chung, S. Duquennoy, T. Voigt, and U. Rödig, "Securing Internet of Things with lightweight IPsec," SICS Swedish ICT AB, Kista, Sweden, SICS Report, 2010.