A Survey on Protocols securing the Internet of Things: DTLS, IPSec and IEEE 802.11i

Rene Roepke[1], Timo Thraem[1], Johannes Wagener[1], and Alex Wiesmaier[1,2,3]

[1] TU Darmstadt
[2] AGT International
[3] Hochschule Darmstadt

Abstract. Until 2020 more than 20 billion devices will be connected to the Internet. The communication between these devices must be secured for privacy, security and safety reasons. Constrained environments, low power assumptions and rapidly changing networks introduce new problems for classical communication protocols. One solution to this problem is to adapt successful protocols as lightweight implementations to fit the requirements established in an IoT context.
In this work, we discuss cryptographic protocols and their lightweight variants with focus on the applicability to the Internet of Things. We give recommendations on which protocol to use in which application area and examine the applicability of the discussed protocols in the automotive environment, a concrete use case of the Internet of Things.

Keywords: Internet of Things, IoT, cryptography, protocol, 6LoWPAN, communication, networks, automotive, DTLS, IPSec, IEEE 802.11i

1 Introduction

Over the past years, the Internet of Things (IoT) has become one of the most used buzzwords in the computer industry. It is an umbrella term for the connection of different real world objects with each other via internet.

Consider Car-to-X communication as an example for an IoT environment. Car components like the breaks, airbags, fuel consumption, speedometer and the board computer are connected with each other. The car, for its part, is connected to the vendor and to its environment. This includes other cars and the traffic system.

A large majority of contributions and discussions regarding IoT is focused on the benefits which could or will arise from the upgrade of current infrastructure or the inclusion of unconnected parts to IoT networks.

With more and more different devices communicating, the question is how to secure the data traffic. Traditionally, most components connected to the internet are systems with enough computing power and memory to perform sufficiently secure cryptographic operations within suitable time and effort. This is not the case for most IoT components. These components are usually lacking the required memory or computing power to perform standard protocols offered by

classical network theory, like TLS, to enforce security through authentication, encryption and integrity.

This work addresses security in IoT by analyzing different cryptographic protocols which are already in use and which are thought of as suitable for secure communication between IoT components. The examined protocols are representatives for mechanisms located on different layers of the OSI model to honor the fact that security can be enforced on different layers.

The work is structured as follows. Section 2 summarizes related work on similar topics. In Section 3, the Datagram Transport Layer Security protocol is presented. Section 4 describes IPSec. Section 5 introduces the IEEE 802.11i protocols and in Section 6, a comparison of the presented protocols is given. The work is concluded in Section 7.

2 Related Work

In previous works, different protocols to ensure security have been studied and surveyed.

Clark and Jacob did a survey on cryptographic authentication protocols [1]. The authors discussed ten different types of authentication protocols and described attacks against some of them. They did not perform a comparison or give advise on which protocol is suitable for which application area.

Akyildiz et al. performed a survey on sensor networks [2]. In a first step, the authors performed a requirement analysis for sensor network protocols, followed by a discussion of the different layers and an overview of different sensor network protocols. However, the security aspect is not discussed in this work.

Akkaya and Younis did a survey on network routing protocols in wireless sensor networks with respect to their constrains [3]. Only the path establishment is discussed.

Güngör et al. discussed communication protocols in smart grids with respect to the physical layer and the application area [4]. The focus is on the application not on security.

In contrast to the related work, this work gives an overview of security protocols with their applicability in the context of IoT, including sensor and less constrained networks. By analyzing protocols located on different layers and applying them to a concrete scenario, this survey work is the first one of its kind to the knowledge of the authors.

3 Datagram Transport Layer Security

Datagram Transport Layer Security (DTLS) is a Session Layer protocol for secure datagram-based communication. It was introduced by Nagendra Modadugu and Eric Rescorla in 2004 and is standardized in RFC 4347 [5]. Its current version is version 1.2 and it is specified in RFC 6347 [6].

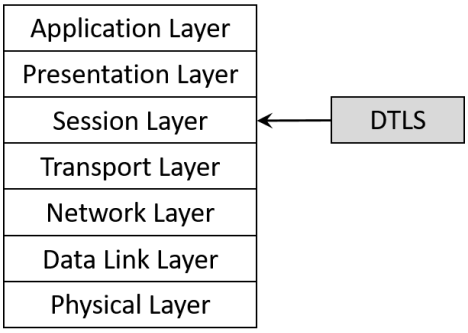

Fig. 1. Position of DTLS in the OSI Reference Model (derived from [7])

It is located in the Session Layer between the Presentation Layer and the Transport Layer. DTLS enables end-to-end security between applications in multi-hop networks.

DTLS is an extension of the Transport Layer Security (TLS), which unfortunately is not capable of handling datagram traffic since it is using communication via TCP. DTLS is still very similar to TLS, hence we focus on the differences. It reuses pre-existing protocol infrastructures of TLS and additional features to support datagram-based communication, e.g. using UDP.

DTLS ensures authentication, confidentiality and integrity. Due to its similarities, DTLS inherits security issues from TLS. It is rated as secure as TLS in a comparable configuration and under the assumption that similar cryptographic primitives are used [6]. Security issues based on TCP are not inherited by DTLS, because it is relying on datagram traffic, i.e. for example communication via UDP.

DTLS provides a simple interface to a generic security, so it is easy to be used in software. It is also referred to as Datagram TLS [8].

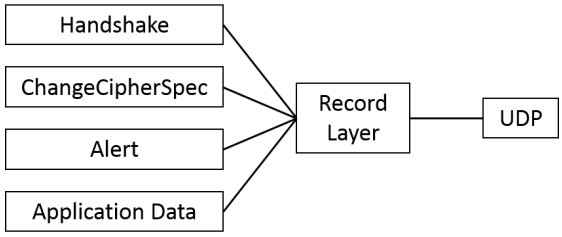

Fig. 2. The Structure of DTLS (derived from Figure 1 in [8])

As shown in Figure 2, DTLS contains a record layer with a connection for UDP-based communication with various types of messages. DTLS supports a hand-

shake protocol to establish a secure communication between a client and a server. Similar to TLS, it also supports alert messages, changes in cipher specifications and sending application data.

3.1 Record Layer

TLS uses a Record Layer to handle actual data and DTLS reuses this concept with some minor changes. It takes data from the Session Layer and serves it to the Transport layer after reformatting and encryption. It uses a record format but unlike TLS, it does not use fragmentation in the record layer. Data is formatted as single datagrams.

```
struct {
    ContentType       type;
    ProtocolVersion   version;
    uint16            epoch;
    uint48            sequence_number;
    uint16            length;
    opaque            payload[length];
} DTLS Record;
```

Fig. 3. Record format for DTLS ([8])

The record format (see Figure 3) contains similar information as the record format of TLS. `Type` is the higher-level protocol used to process the enclosed fragment as specified in TLS 1.1 [9]. `Version` is a struct of two integers, the major version number and the minor version number of the protocol. The record format also contains a field for the payload and the payload-length.

Compared to TLS, two new fields are introduced in the record format: `epoch` and `sequence_number`. Endpoints use `epoch` numbers to distinguish which cipher state has been used to secure the payload. The `epoch` number is incremented in the handshake, when the cipher state changes. It is also used to resolve ambiguity when data loss occurs during session negotiation.

For instance, consider a server receiving two records: one record with sequence number 9 and epoch number 2 and another record with sequence number 11 and epoch number 3. The client sent data records with sequence numbers 9, 10, 11 and 12. Records 10 and 12 are lost. Since `epoch` numbers are incremented when sending `ChangeCipherSpec` (see Section 3.2) the server can resolve possible ambiguity, assuming record 10 was the `ChangeCipherSpec`, because record 9 and 11 would have the same `epoch` number otherwise.

Sequence Numbers are used for reordering and to protect against replay attacks. Due to datagram transport, records can get lost or be delivered in a different order than intended. While TLS uses implicit record sequence numbers

(RSNs), DTLS enforces explicit RSNs to protect against replay attacks and for reordering.

3.2 Handshake Protocol

The `Handshake` protocol of DTLS is a three round-trip key establishment and algorithm negotiation protocol and it is very similar to the handshake protocol used in TLS. Figure 4 depicts the handshake protocol as it is used in DTLS.

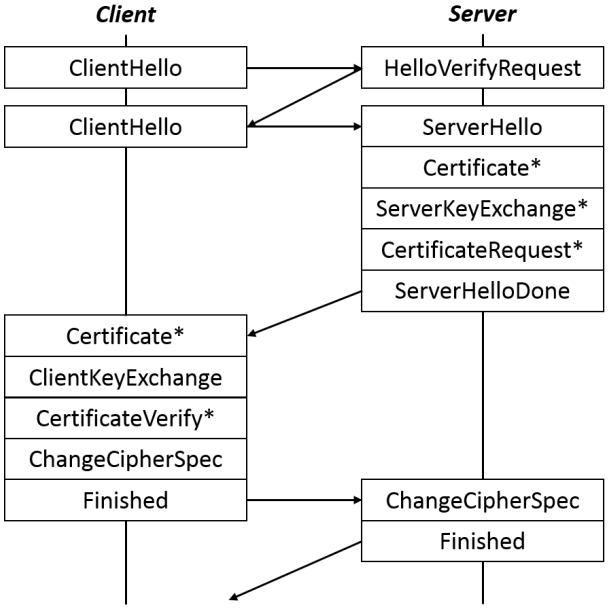

Fig. 4. Handshake Protocol (derived from [6])

Compared to TLS, the handshake protocol contains an additional round-trip, which is needed for cookie exchange. DTLS uses cookies to prevent denial of service attacks and to verify that the client is capable of receiving packets at its claimed address.

Given a client and a server, the client initializes the handshake by sending `ClientHello` message, containing a 32-bit timestamp, a 28-bit random nonce R_c, the protocol version, a list of applicable algorithms and a session id, to the server. This message contains also a cookie field, which is initialized with an empty cookie. The server checks for liveliness of the client by responding with a `HelloVerifyRequest` message containing a cookie.

In the next round of the handshake, the client sends the `ClientHello` with the same parameter values as before, but it adds the received cookie from the `HelloVerifyRequest`. In case that the client already has a cached, stateless

cookie from previous exchanges, the first round can be skipped by sending a `ClientHello` message containing the cookie. The server does not need to check for liveliness anymore and continues. The message formats of `ClientHello` and `HelloVerifyRequest` are displayed in Figure 5.

```
struct {
    ProtocolVersion   client_version;
    Random            random;
    SessionID         session_id;
    Cookie            cookie;
    CipherSuite       cipher_suites<2..2^16-1>;
    CompressionMethod comp_meth<1..2^8-1>;
} ClientHello;

struct {
ProtocolVersion server_version;
Cookie cookie;
} HelloVerifyRequest;
```

Fig. 5. ClientHello and HelloVerifyRequest format (derived from [8])

In the second round, the server responds with multiple messages. It sends a `ServerHello` message including similar information as in the `ClientHello`, as well as a random nonce R_s. The `Certificate` message contains the certificate information of the server for authentication of the server towards the client. This message is optional (as indicated with a star * in Figure 4)

Additionally, a `ServerKeyExchange` message may be sent to hand out a temporal public key for RSA. This is required if the server has no certificate, or if its certificate is for signing only. By sending a `CertificateRequest`, the server can request the clients certificate. The `ServerHelloDone` is the last message and it is a marker that no other message is to be expected after this.

After receiving the `ServerHelloDone` message, the client sends multiple messages depending on the response of the server. If the client received a certificate of the server, it verifies it. When receiving a `CertificateRequest` message, the client responds with its certificate information.

Next, the client sends a `ClientKeyExchange` message. It contains a pre-master secret and it is encrypted with the servers public key. If no certificate was sent by the server, the temporal public key of the `ServerKeyExchange` message is used to encrypt the pre-master secret. The shared pre-master secret will be used as key material to compute the master secret on both sides separately. With the master secret, client and server can derive the keys for the chosen algorithm.

Suppose the client only has a signing certificate. To enable the server to verify the previously sent certificate, a `CertificateVerify` containing a signed hash

over all previously transmitted handshake messages is sent. This way, the server can verify the authenticity of the client.

The purpose of the `ChangeCipherSpec` message is to commit to the agreed cipher suite. It is sent in the third round of the handshake protocol. The `Finished` message contains a message authentication code (MAC) of the previous handshake messages encrypted with the agreed cipher suite. When receiving the `Finished` message, the server changes its cipher suite and acknowledges it too. To finalize the handshake, the server sends a `Finished` message like the client did before. Both `Finished` messages, contain a MAC over all sent messages computed with the master secret. If the MAC values are equal, the handshake was successful and both parties can begin transmitting application data.

Because handshake messages can be larger than a single DTLS record, fragmenting is supported. In Figure 6, the format of the handshake messages is displayed.

```
struct {
    HandshakeType     msg_type;
    uint24            length;
    uint16            message_seq;
    uint24            frag_offset;
    uint24            frag_length;
    HandshakeMessage  msg_frag[frag_length];
} Handshake;
```

Fig. 6. Handshake format for DTLS (derived from [8])

The handshake header contains the overall message `length`, a message sequence number (MSN, `message_seq`), as well as the fragment offset (`frag_offset`) and its length (`frag_length`). These information ensure that handshake fragments can be ordered and reassembled correctly.

As previously stated, DTLS does not support fragmentation for the record layer. Fragmentation is only supported for handshake messages.

3.3 Timeout and Retransmission

Due to the fact that handshake messages can be lost, a state machine for retransmission is implemented using a single timer at each endpoint. An endpoint keeps retransmitting its last message until a reply is received or a maximum timer value is reached. Retransmission timer values can vary between one and three seconds, but due time-sensitive applications, one second is recommended [6].

As shown in Figure 7, the state machine is based on four states, i.e. Preparing, Sending, Waiting and Finished.

In the PREPARING state, all computations that are necessary to prepare the next transmission of messages are done. It buffers the messages for transmission and enters the SENDING state.

In the SENDING state, the buffered messages are transmitted. Once the messages have been sent, the FINISHED state is entered if it was the last round in the handshake. Or, if more messages are expected, a retransmission timer is set and the WAITING state is entered.

In the WAITING state, there are multiple ways to continue:

- When the retransmission timer expires, the SENDING state is entered, where it retransmits the messages, resets the retransmission timer, and returns to the WAITING state.
- A retransmitted round of messages is received and the SENDING state is entered, where it retransmits the messages, resets the retransmission timer, and returns to the WAITING state.
- When receiving a round of messages and it is the final round of the handshake, the FINISHED state is entered. If the received round of messages was not the final round, the PREPARING state is entered.

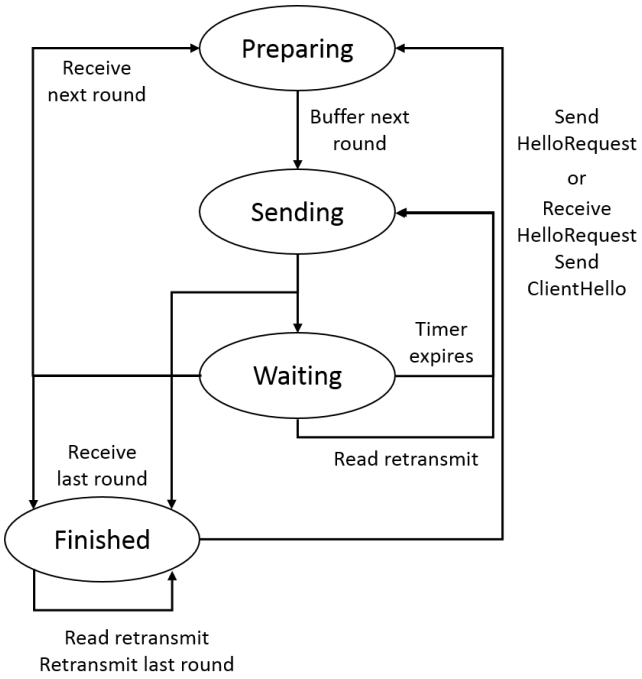

Fig. 7. Handshake format for DTLS (derived from [6])

Because DTLS clients send the first message (`ClientHello`), their state machine starts in the PREPARING state. For the servers, the state machine starts in the WAITING state, but with empty buffers and no retransmission timer.

When the server wants to redo a handshake, the state machine transitions from the FINISHED state to the PREPARING state transmitting the `HelloRequest`. If the client receives a `HelloRequest`, its state machine transitions from FINISHED to PREPARING to transmit the `ClientHello`.

When a retransmission timer expires the timer value is doubled, up to a maximum of 60 seconds [6]. Congestion should not be a concern, since retransmission is only used in the handshake and not in the application data transfer.

3.4 ChangeCipherSpec Protocol

The `ChangeCipherSpec` Protocol contains just a single message. It is one byte large and contains the value 1. By sending it in the handshake, the sender informs the receiver about switching to the negotiated cipher suite. The `ChangeCipherSpec` Protocol can not be used for key renewal later in the session. It is simply used in the handshake protocol.

3.5 Alert Protocol

The `Alert` Protocol of DTLS is used to notify the other party that an error might have occurred, for example if a certificate could not be verified. It is also used for warnings.

If a warning is send out, the connection remains established, but when sending an error, the connection will be shut down.

Another purpose for alert messages is the graceful termination of the connection when one party is done and has nothing to send anymore. Therefore, a *CloseNotify* message is send by both parties.

3.6 Record protection

The cipher modes of TLS 1.0 are all unsuitable for DTLS because of residual states between records. By chaining the records during encryption, TLS requires data records to be processed without any loss and in the intended order. DTLS cannot ensure reliable or in-order delivery, such that if a record gets lost all remaining records would be useless.

For TLS 1.1, a cipher block chaining mode (CBC) was proposed, which has been adopted to DTLS. By using explicit initialization vectors (IVs) instead of using the last ciphertext block from the previous message, CBC is suitable for DTLS. Within each datagram, a random data block is prepended to the datagram encrypted with CBC and all encrypted blocks are transmitted. The receiver discards the first plaintext block to retrieve the actual record data. With an explicit IV each record can be separately decrypted [8]. Therefore, AES and Triple-DES are suitable encryption algorithms for DTLS.

For message integrity, MACs are used to secure the payload. In the current version of DTLS, the same MAC computation as in TLS 1.2 is used, but instead of using implicit sequence numbers, the `epoch` and `sequence_number` (as described in 3.1) from the record itself is used. The offered MAC computations are HMAC-MD5, HMAC-SHA-1 and HMAC-SHA-256.

As shown in Figure 8, securing the payload's confidentiality and integrity is done in two steps. First, the MAC over the concatenation of `SQN`, `HDR` and the payload is computed. `SQN` are 8-bytes for the sequence number and epoch of the current record and `HDR` is a concatenated value containing the version number, the type and the length of the payload. In the second step, the payload, the computed MAC value and a padding are concatenated, such that the resulting plaintext length is a multiple of the block size b of the used encryption scheme ($b = 8$ for Triple-DES and $b = 16$ for AES) [10, 11].

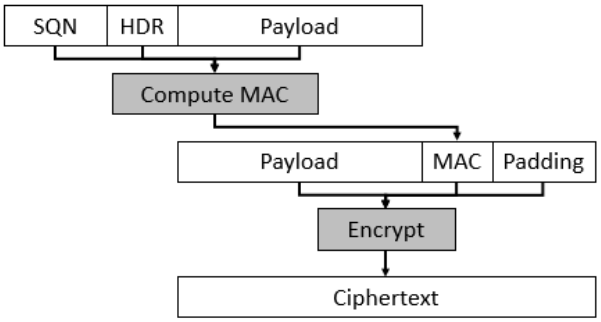

Fig. 8. Securing the payload (derived from [11])

3.7 Known Attacks

Based on its high similarity to TLS, DTLS and TLS have a similar attack surface. Most attacks on TLS can be considered to be an attack on DTLS if the attack does not target TCP. DTLS is implemented on UDP, hence TCP-based attacks are not applicable to DTLS. For more details on known attacks on TLS, [12] summarized known attacks on TLS 1.2 and also stated the applicability to DTLS but without further explanations. Newer versions of TLS are not applicable since DTLS' current version is designed by TLS 1.2.

Throughout the design of DTLS, known weaknesses of TLS and UDP-based communication have been considered. Replay attacks during data transmission are resolved with sequence numbers in the record layer. Also DTLS is susceptible to Denial of Service (DoS) attacks but with no fragmentation in the record layer, hosts do not need to buffer partial records. Memory can be used more efficiently and DoS attacks are less effective.

During the handshake, a stateless cookie exchange prevents DoS attacks. This way, DTLS is secured against resource consumption attacks and amplification attacks. In order to optimize the handshake, servers can skip parts of the cookie exchange. But this should only be done in environments which are safe against DoS attacks.

The authors of [13] describe a plaintext recovery attack against the OpenSSL implementation of DTLS, and a partial plaintext recovery attack against the GnuTLS implementation of DTLS. The attack is possible because of differences between the implementations and the RFC 4346 [9]. It could have been prevented if the implementation had been in accordance with the specification instead of using a prior release version of DTLS. An extension of these attacks was presented by the authors of [11]. Al Fardan and Paterson developed more attacks against DTLS and TLS based on a timing analysis of decryption processing.

As mentioned earlier, DTLS ensures authentication, confidentiality and integrity. Overall, it is rated as secure as TLS in a comparable configuration and known attacks on TLS have also been applied to DTLS [11, 12].

3.8 Differences to TLS

The main goal in the design of DTLS was to follow the specification of TLS as closely as possible. Newly introduced features of DTLS are for dealing with unreliable datagram communication and therefore the differences between TLS and DTLS are rather small [5]

TLS provides secure, transparent communication and is used in environments where no memory or power limitations are given. It requires reliable transport channels, typically with TCP and no datagram traffic is intended. Due to more datagram traffic in today's applications, less connection-oriented but still reliable and secure communication is needed. DTLS holds the requirement of a secure channel, but no reliable communication is possible. Messages can be lost or arrive in a different order. DTLS uses explicit sequence numbers to resolve reordering problems and lost messages are retransmitted based on a timer. Compared to TLS, DTLS has a smaller Path Maximum Transmission Unit (PMTU) and does not support fragmentation. This way, hosts can use memory more efficiently because they do not have to buffer partial records before decryption.

While TLS comes with different cipher suites, DTLS cannot use most of them, due to fragmentation and datagram loss in UDP. DTLS supports CBC mode, like it was introduced for TLS 1.1. Triple-DES and AES are compatible encryption schemes with DTLS is this mode. RC4, which secures TLS 1.0 connections is computationally efficient but insecure and cannot be applied to lossy datagram-based traffic.

The handshake protocol and record layer of TLS and DTLS are quite similar. The additional round for the stateless cookie exchange in the handshake protocol and the epoch and sequence numbers in the record format differ from DTLS to TLS.

As compared in [8], DTLS and TLS are very similar, but DTLS has a small overhead in the handshake protocol. Due to an additional round and larger

message fragments, the handshake protocol is larger than for TLS. Because of single datagram traffic, DTLS records have no fragmentation headers, which means the overhead for data records is lower than for TLS. Latency measures of TLS and DTLS handshakes showed exactly the expected difference of one extra round-trip time (RTT).

3.9 DTLS in IoT

DTLS was not designed for lossy networks and constrained devices at first, but it quickly became a key candidate for security in IoT anyway [14]. Since it is considered to be heavy, lightweight implementations of DTLS were needed. As presented in [14, 15], lightweight DTLS implementations could be based on using Pre-shared Keys (PSK) or raw public keys. `TinyDTLS` is one candidate which was developed by Bergmann [16]. The major advantage of DTLS over other security protocols is the UDP-based communication and the memory efficient properties on the host.

Another approach to adapt DTLS to IoT environments was suggested by Raza et al. [17, 18]. By proposing a DTLS header compression, DTLS can be used with the 6LowPAN standard and the compression does not compromise the end-to-end security properties of DTLS. In IoT scenarios lower power communication stacks are very common and DTLS can be adapted to it.

The IETF DTLS in Constrained Environments (DICE) working group [19] leads the research on supporting DTLS usage in constrained environments with tasks like adapting record layer for secure multicast messages or developing reasonable implementations for IoT purposes. The IETF Constrained RESTful environments (CoRE) [20] working groups specifies the standardization of CoAP and also proposed DTLS usage with CoAP as still the standard [21]. Considering the ongoing standardization activities, there is necessity for modifications in order to adapt DTLS more and more to IoT scenarios.

4 IPSec

The IPSec standard is specified in its current version by the Internet Engineering Task Force in 4301 [22] and 4302 [23]. IPSec is located on the Internet layer (layer 3) of the OSI layer model. It can be used to ensure confidentiality, integrity and authenticity [2]. With the introduction of IPSec, a security model was introduced, which provides security on a low layer in the OSI model. It allows to secure multiple connections between hosts or gateways without the need to change the implementation on the higher layers. IPSec ensures end-to-end security between devices respectively networks on the internet layer.

4.1 Internet Key Exchange Protocol

Before IPSec can be used to protect connections, keys must be shared between the communication parties to establish a so called security association (SA). In

IPSec, the keys can be managed manually, which is called manual keying or with internet key exchange protocol (IKE). With manual keying, the keys between the parties are pre-shared and configured on the endpoints. IKE runs on UDP and is therefore connectionless and not reliable. If IKE is used, the keys are shared automatically. IKE defines how security parameters are agreed upon and shared keys are exchanged. IKE defines two phases.
In phase one, mutual authentication and session keys are established. This is based on pre-shared secrets or public key pairs for authentication. IKE exists in two versions: IKEv1 and IKEv2 [5]. To reduce the complexity of IKEv1, the protocol was fundamentally simplified with IKEv2. In IKEv1 and IKEv2, different crypto suites can be chosen to establish a secure communication between parties. Beside the negotiation of the crypto suites, IKE provides stateless cookies, which prevent Denial-of-service attacks. In IKEv2, the party to which the connection is established decides whether to use cookies or not. If cookies are used, the number of messages increases from 4 to 6 messages in IKEv2. In this work the use of IKEv2 is assumed.
In phase two, Security Associations (SA) are established. If public key authentication is used in phase one, IKE is a hybrid cryptographic protocol. But even if pre-shared secrets are used, the negotiated keys in phase two are independently chosen from those in phase one. This ensures perfect forward secrecy. The SAs are used to secure the data transmission after. One of the most important drawbacks of IKE is the fact that it does not support certificate transmission. In order to authenticate a peer the peer must be known before SAs can be established.

All in all, it can be said, that in phase one the security parameters are negotiated, which are used to share SAs. In phase two, the SAs are negotiated and the information inside the SAs are used to secure the connection between the parties.

4.2 Databases and data structures

To be able to secure an IP connection with IPSec, the device must create at least three databases. These databases are called Security Association Database (SAD), Security Policy Database (SPD) and Peer Authorization Database (PAD).

Security Association and Security Association Database
The Security Association (SA) is a dataset stored in the Security Association Database (SAD). SAs store how to secure the connection between parties. Each SA entry contains at least a Security Parameter Index (SPI), Destination IP and Security protocol identifier. The SPI identifies the security parameters to secure a connection. SAs are unidirectional, which means that for each connection and security protocol identifier one SA must be established. There are two types of SAs. The `IKE_SA`, which has a long time validity, is based on a PKI or pre-shared secret and is used to negotiate the `CHILD_SA`. `CHILD_SA`s, which are used for data transmission, are the second type of SAs. In the `IKE_SA` the parameters used in phase one of the IKE are stored. The parameters negotiated in phase two of the IKE protocol are stored in `CHILD_SA`s.

Security Policy Database
The Security Policy Database (SPD) manages the SAs. The SPD decides if the incoming or outgoing traffic has to be protected via IPSec or not according to the stored policies. An entry in the SPD consists of at least protocol, local IP, local port, remote IP, remote port and action. The action can be 'bypass', 'discard' or 'protect'. If the action is 'protect', the mode and the IPSec security mechanism is specified inside the SPD.

Peer Authorization Database
The Peer Authorization Database (PAD) provides the link between SPD and SA management protocol (IKE). In the PAD, the peers or groups of peers which are authorized to communicate via IPSec with the local entity are identified. The protocol and method to authenticate each peer is specified inside the SAD. Constrains for types and values of IDs that can be asserted by a peer with regard to child SA creation are managed in the PAD. This ensures that no peer can assert identities for lookup in the SPD which it is not authorized to represent, when child SAs are created.

4.3 IPSec Modes and Security Mechanisms

In IPSec two modes and two security mechanisms are standardized. Depending on the scenario of use and protection targets different combinations of mode and security mechanism are used.

Modes
The two modes of IPSec are Tunnel Mode and Transport Mode. Transport Mode sets up a secure end-to-end connection between two hosts. In Transport Mode, the end-nodes must be configured to use IPSec. Tunnel Mode sets up a secure connection between two networks. If Tunnel Mode is used, the IP packet is encapsulated inside an IPSec packet. A private tunnel between gateways is established in Tunnel Mode. This allows two networks to communicate securely through a public network. In Tunnel Mode only the gateways have to be configured for the IPSec connection and not each peer behind the gateways.

Security Mechanisms
The IPSec security mechanisms are Authentication Header (AH)[6] and Encapsulated Security Payload (ESP)[7].
Authentication Header:
If Authentication Header is used as security mechanism, the IPSec header contains an encrypted hash of the whole IP packet. AH offers source authentication and message integrity, but not message confidentiality [4]. It authenticates the IP header, the AH header and the IP payload. For authentication, the field with the authenticated data in the AH and mutable fields are set to zero. All fields which are present in the packet before the HMAC is generated for authentication must

be authenticated. All Fields which are manipulated during respectively after authentication must be set to zero to ensure a valide autehntication check. Figure 9 illustrates how AH is applied. In figure 10 the structure of AH is illustrated. The destination peer uses the Security Parameter Index (SPI) in the header to identify the the correct security association. The sequence number is used to prevent replay attacks. The Integrity Check Value (ICV) is the cryptographic hash value which allows the receiver to check the integrity of the whole IP packet including the AH.

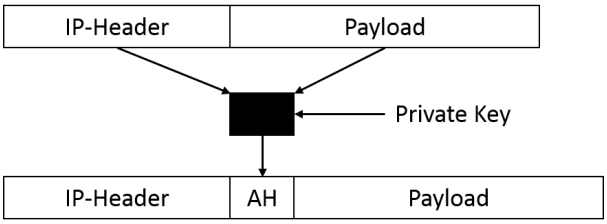

Fig. 9. Authentication Header

Next Header	Header length	Reserved
Security Parameter Index		
Sequence Number		
Integrity Check Value-ICV (variable) = ESP Auth		

Fig. 10. Authentication Header Structure (derived from [23]

Encapsulated Security Payload:
Encapsulation Security Payload (ESP) guarantees the integrity and confidentiality of the original IP payload combining a secure hash and encryption of the IP payload (including the ESP trailer). If used in Tunnel Mode (gateway to gateway), the whole IP packet is protected. Figure 11 illustrates how ESP is applied. In figure 12 the structure of the ESP header is illustrated. The payload is contained in the header because the payload is encapsulated in the header fields. The fields Padding, Padding length and Next header are part of the trailer components. The SPI and the sequence number take over the same task as in AH. The IP payload and the padding are ecrypted. The padding length is needed for the decryption. The ESP-Auth is used to authenticate the ESP header, the IP payload and the ESP trailer as illustrated in figure 11.

Both AH and ESP use HMAC for authentication and ESP uses symmetric encryption to encrypt the IP payload and the ESP trailer. ESP and AH can be

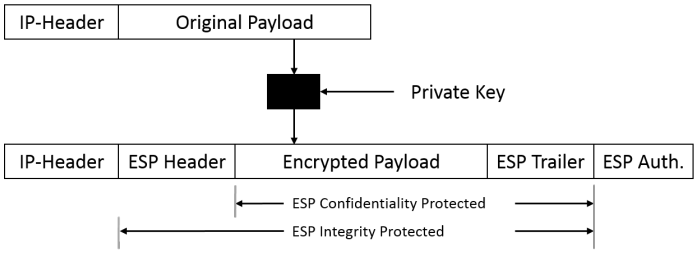

Fig. 11. Encapsulated Security Payload

Security Parameter Index		
Sequence Number		
IP Payload (variable length)		
IP Payload (variable length)	Padding (variable length)	
Padding (variable length)	Padding length	Next Header
Integrity Check Value-ICV (variable) = ESP Auth		

Fig. 12. Encapsulated Security Payload Structure (derived from [24]

combined. AH does not only authenticate the payload but also the non mutable IP Header field. By combining AH and ESP all ESP security mechanisms can be applied with this additional AH mechanism. In IPSec, a replay prevention technique called anti-replay window is used. Since AH and ESP contain sequence numbers, the anti-replay window makes use of these sequence numbers. The sequence numbers increase per packet. The anti-replay window has a specific size (n). If packet 'm' arrives and no packet with a higher sequence number arrived before, the window is shifted so that the oldest packet which the peer will accept is m-n. If packet 'b', a packet with a higher sequence number than 'm', arrived before the packet 'm', the sequence number of 'm' must be in the window a-n to be accepted. If 'm' has a lower sequence number, it will be discarded. As already mentioned, SAs are unidirectional. If the connection between two parties is secured with AH and ESP, each party has to create four SAs. Nowadays, the most common usage of IPSec is Tunnel Mode with ESP.

4.4 IPSec Workflow

After key exchange via manual keying or IKE, respectively after the SAs are established IPSec can be used. In dependence of the data flow (inbound or outbound connection), IPSec uses the databases in different order.

Inbound Connection
Figure 13 illustrates how incoming connections are processed. If an IPSec packet is received, the SPI from the packet (packet_SPI) is looked up in the SAD. If the SAD contains a SA which matches the information provided by the packet, it is picked for further processing. If no SA is defined the message is send without IPSec protection. With the information from the SA the packet is processed, which means authenticated and/or decrypted. Then the SPD is used to look up whether the policies are fulfilled. If the policies are fulfilled, the packet is forwarded to the next instance of the OSI layer. If not, the packet is discarded. Note that not just the packet_SPI is evaluated to match an SA. The lookup in the SAD is a longest match search with the order

- SPI, destination, and source address
- SPI and destination address
- SPI

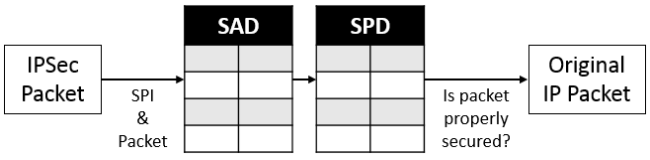

Fig. 13. Inbound Connection

Outbound Connection
The processing of outgoing connections is illustrated in figure 14. Before an IP packet is passed to the link layer, a lookup in the SPD is performed to check whether to secure the packet with IPSec or not. If the SPD has no policy, the packet is send without IPSec protection. If at least one policy in the SPD is found, the SA or SAs in the SAD associated with the policy are used to protect the packet and the SPI is inserted into the IPSec packet so that the receiver can process the IPSec packet.

Fig. 14. Outbound Connection

Known Attacks
In this section known attacks are discussed. The Sans institute[4] published a document with vulnerabilities. The *Cut-And-Past* attack is an attack which is only possible if IPSec is used in tunnel mode and the attacker has access to a second machine in each of the two networks. A second attack with the same setup is *Session-Hijacking*. The prerequisites for this attacks are that the attacker has access to networks connected via IPSec tunnel. Such an attacker is out of the IPSec protection goals scope. Replay attacks in general are prevented by sequence numbers and the replay window. Attacks against the chosen cipher suites are issues rather concerning the cryptographic primitives than the protocol. Attacks against IKE, like DoS attacks are prevented by stateless cookies.

4.5 IPSec in IoT

IPSec can be used to protect traffic between peers at a low level in the OSI layer model. Vasseur & Dunkels [25] argued that 6LoWPAN enabled sensors will be the basis for the IoT. Therefore IPSec should work with 6LoWPAN devices.

Overview of 6LoWPAN
The 6LoWPAN-Standard is introduced to work upon the link layer in the OSI Stack for wireless networks communication between low power devices[26] based on the IEEE 802.15.4 standard. The main component is the 6LowPAN adaption layer which replaces the IP layer. One main feature is the advanced header compression[26]. The maximum transmission unit (MTU) is 102 byte (127 if the link layer is included). Between the link layer and the 6LowPAN layer an additional encryption layer can be added optional. This layer consists of 40 byte. The standard IPv6 header requires 40 byte and the UDP header 8. Therefore only 30 byte are reserved for payload. In 6LoWPAN networks the IPv6 and UDP headers are compressed to 8 byte.

Problems with IPSec in 6LoWPAN
The IPSec specifications from the IETF are not applicable in 6LoWPAN networks. 6LoWPAN networks use header compression [27] which is not usable with the standard implementation of ESP. Further problems with IPSec are header size, computing power and memory. Granjal et al. [28] analyzed the performance of typical IPSec cryptographic algorithms in real sensor networks and the performance of AES, 3DES, SHA1 and SHA2 on a MicaZ [5] embedded computer. Only SHA1 could be performed with 6LoWPAN, because TinyOS and 6LoWPAN implementations do not leave enough SRAM for additional cryptographic mechanisms. According to them, SHA1 is the most energy efficient and most suitable hash algorithm in terms of performance, but insecure.

[4] www.sans.org
[5] http://www.memsic.com/userfiles/files/Datasheets/WSN/micaz_datasheet-t.pdf

Lightweight variants of IPSec

To address the problems with IPSec in IoT scenarios lightweight IPSec implementations for use with 6LowPAN were introduced. In this section, two lightweight solutions are presented. One for the use of a lightweight IKEv2 protocol and one lightweight IPSec implementation which supports 6LoWPAN header compression. With reasonable effort we were just able to find two paper treating IPSec in 6LoWPAN networks. Both from the same main author.

Raza et al. [29] proposed a lightweight variant of the IKEv2 protocol for 6LoWPAN networks. The proposed lightweight IKEv2 can be used for IPSec and for IEEE 802.15.4 link layer security. They proposed that IPSec enabled devices should recognize UDP-IKE packets due to problems with header compression. The NHC encoding for the IKE header contains 4 bits for the NHC ID (1101), one bit for each of SPI, exchange-type (ET), Message ID (ID) and next header (nh). If the SPI header field is 0, the default sensor network SPI is used. If it is set to 1, the SPI is carried inline after the header. If ET is set to 0, two bits specify the four standard exchange-type. If ET is set to 1, all 8 exchange-type bits are carried after the header. ID specifies a sequence number. If ID is set to 0, a 16 bit sequence number is used. If ID is set to 1, a 32 bit sequence number is used. NH indicates the next header. This header format allows header compression and minimizes IPSec overhead. The minimized overhead makes lightweight IKEv2 applicable for 6LoWPAN networks.

Raza et al. [30, 31] proposed a lightweight IPSec header format for AH and ESP. The proposed AH header contains 4 bits for the NHC identifier and one bit for each of payload length (PL), SPI, sequence number (SN) and next header (NH). The NHC identifier is needed to be 6LoWPAN conform. If PL is set to 0, the length field is omitted and can be obtained from the SPI value. If PL is set to 1, the length is carried inline after the header. The SPI and the NH fields are constructed as in the proposed lightweight IKEv2 and SN is constructed as the Message ID in the lightweight IKEv2. The proposed lightweight ESP header contains 4 bits for the NHC Identifier and one bit for each of payload length (PL), SPI, sequence number (SN), a reserved bit and next header (NH). SPI, SN and NH are constructed as the proposed lightweight AH. Raza et al. [31] evaluated the energy consumption, traffic overhead and processing time overhead of IPSec in 6LoWPAN networks with AES-CBC and AES-XBC-MAC-96. The energy consumption, traffic overhead and processing time are "a bit higher" than without, but the evaluation shows that it is feasible to use IPSec for sensor networks.

Discussion

The proposed lightweight variants of IPSec are not standardized. More evaluations of the implementations must be performed to make a good decision whether to standardize it or not. The research of Raza et al. [31] did not consider nodes or peers with less memory which is named as a bottleneck by Granjal et al. [28]. The cryptographic systems evaluated in [31] are not part of the standard IPSec crypto suites, but recommended for IoT devices. In the case of IKEv2

for IoT systems a identity management must be generated, which is faster in terms of identifying peers and authenticating performance than normal public key systems and more scalable than pre-shared keys. Even if a large number of problems must be solved before IPSec for 6LoWPAN can be used, IPSec seems to be a promising candidate to ensure confidentiality, integrity and authenticity on the network layer.
IPSec is a widely used standard in cooperate environments. Even if it is widely used we want to highlight, that IPSec reduces the OSI layer structure to absurdity and that security features are removed with the introduction of IKEv2.

5 IEEE 802.11i

IEEE 802.11i-2004 or shortly 802.11i is a amendment to the IEEE 802.11 standard. It specifies security mechanisms for wireless networks. Published in 2004 [32] by the Institute of Electrical and Electronics Engineers (IEEE), it was later incorporated into the IEEE 802.11-2007 standard [33]. In 802.11i, security association management protocols called the *4-Way Handshake* and the *Group Key Handshake* are introduced as part of the authentication process. Furthermore, the *Temporal Key Integrity Protocol (TKIP)* and the *CTR with CBC-MAC Protocol (CCMP)* for data confidentiality and integrity are defined. All protocols are located on the data link layer.

Sources of information for sections 5.1 and 5.2 are [32] and [34], if not otherwise stated.

5.1 Authentication

When a device, like a smart phone for example, wants to join a wireless network, it has to prove that it is eligible to join the network. This process is called authentication. The device, or Station (STA) in this context, requests network access at a so called Access Point (AP).

The first of two steps of the authentication process described by 802.11i is access control via either IEEE 802.1X EAP [35] or pre-shared key (PSK) authentication. The two cases yield the same result: Both STA and AP hold the Pairwise Master Key (PMK) upon completion of this step. Since the details regarding this are not part of 802.11i, we only give a brief description here on how the PMK is constructed on both sides.

In case of 802.1X EAP authentication, an Authentication Server (AS) is involved, being responsible for the decision whether to grant or deny network access. With this approach, the STA and the AS construct the PMK by using information obtained during their communication. The AS then sends the PMK to the AP. In case of PSK authentication, there is no need for an AS. The PMK is simply constructed by the STA and the AP using the PSK.

Before we go on with the second step of the authentication process, an overview of the involved keys is given at this point, to give the reader a better understanding of the following explanations.

As 'pairwise' indicates, the PMK plays a role in unicast communication between STA and AP. Equivalent to the PMK, there is also a Group Master Key (GMK) for multicast communication from one AP to multiple STAs. PMK and GMK are not used for encryption or message integrity itself, but to derive the Pairwise Transient Key (PTK) and the Group Temporal Key (GTK). A description on how this derivation is done is given later. PTK and GTK are further split up into multiple keys, resulting in key hierarchies. Due to the different way of working of the Temporal Key Integrity Protocol (TKIP) and the CTR with CBC-MAC Protocol (CCMP), the key hierarchies differ for those two protocols. TKIP needs separate keys for data confidentiality and integrity, whereas CCMP achieves these properties with only one key. Figures 15 and 17 illustrate the key hierarchies when using TKIP. The resulting key hierarchies when using CCMP are shown in Figures 16 and 18. The boxes with a grey background are not part of the official terminology, but indicate how parts of the Temporal Key are used.

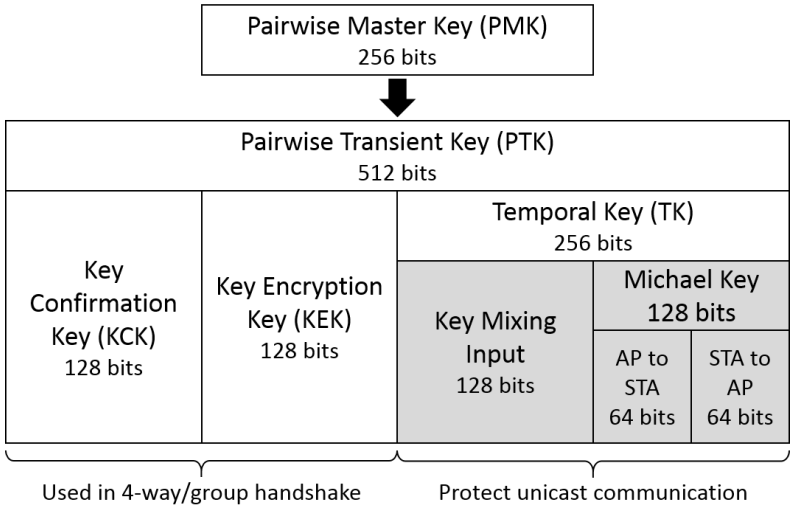

Fig. 15. TKIP Pairwise Key Hierarchy (derived from Figure 10.3 in [34] and Figure 43s in [32])

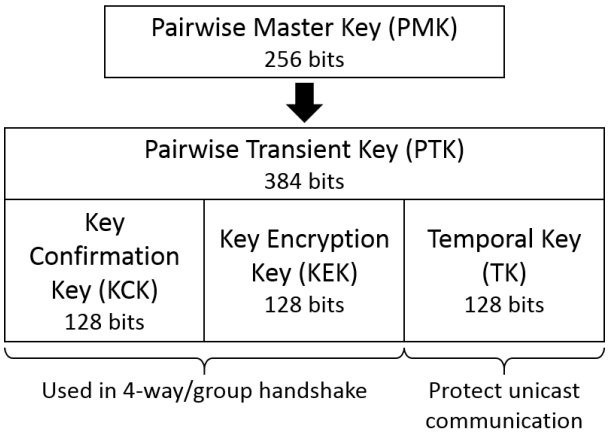

Fig. 16. CCMP Pairwise Key Hierarchy (derived from Figure 10.5 in [34] and Figure 43s in [32])

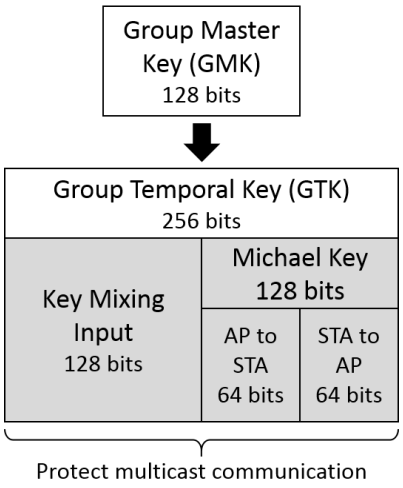

Fig. 17. TKIP Group Key Hierarchy (derived from Figure 10.4 in [34] and Figure 43t in [32])

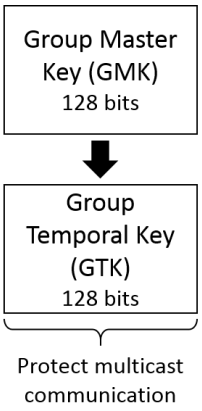

Fig. 18. CCMP Group Key Hierarchy (derived from Figure 10.6 in [34] and Figure 43t in [32])

4-Way Handshake Protocol

Up to this point, STA and AP both hold the PMK, but both sides constructed it on their own behalf. Now they need to prove each other their knowledge of the PMK, so they know they can trust each other. Also, the PTK has to be derived from the PMK and the AP might want to send a GTK to the STA for multicast communication in a secure way. Lastly, STA and AP need to synchronize on when to start encrypted communication. All of this done by executing the 4-Way Handshake Protocol, illustrated in Figure 19. It is the second step and hence the conclusion of the authentication process.

First, both STA and AP generate a nonce. We call it SNonce for the STA and ANonce for the AP.

The AP sends its ANonce to the STA in the first message. The message is not protected in any way. This is simply not necessary, because any modification of the message would cause the handshake to fail. The STA now has all information it needs to derive the PTK.

The second message contains the SNonce and a MIC value computed over the whole message (with the MIC field being zero at the time of calculation) using the KCK (a part of the PTK, see Figures 15 and 16). The term 'MIC' stands for 'Message Integrity Check'. It is basically the same as a message authentication code. The term was introduced, because in 802.11i, the abbreviation 'MAC' is already used for 'Media Access Control'. The MIC function is HMAC-MD5 when TKIP should be used and HMAC-SHA1-128 (output truncated to 128 bits [36]) for CCMP. Including a MIC value in the second message enables the AP to check

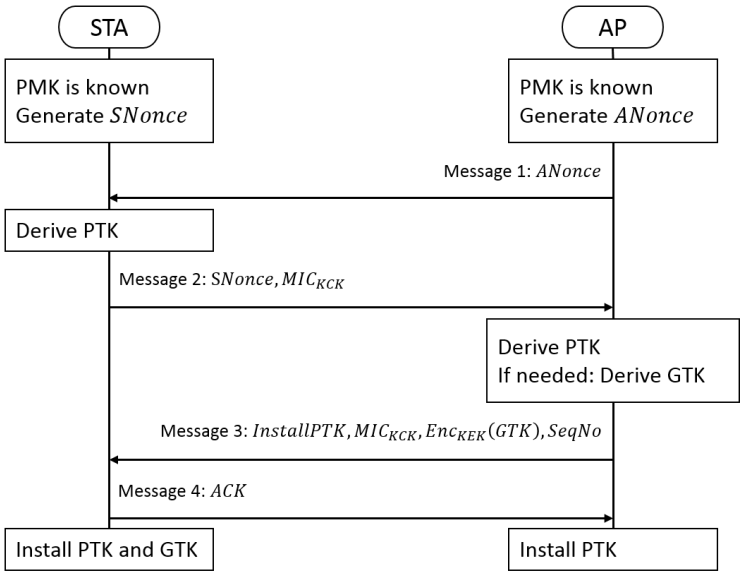

Fig. 19. 4-Way Handshake Protocol (derived from Figure 11c in [32])

whether the message was modified or not. Furthermore, it is a proof to the AP that the STA holds the same PMK.

To see why this is true, consider the following. Since the AP got the SNonce from the STA, it now can also derive the PTK and thereby compute the MIC value of the last received message using its KCK. If the MIC value matches the one it got from the STA, both AP and STA must have used the same PMK to derive the PTK.

If the AP wants to use multicast communication, it generates a GMK, which is a cryptographically-secure random number. It then derives a GTK from the GMK. If there is already a GTK in use, that one is taken at this point. Since the GMK is a random number, the step of deriving a GTK could be omitted, generating the GTK directly. The derivation from a GMK is only done for being consistent with the PMK to PTK case.

The third message contains information telling the STA to install the PTK and a MIC value over the whole message (again with the MIC field being zero at the time of calculation). If needed, the message also includes the GTK, encrypted using the KEK and the sequence number that is used in the next multicast message from the AP (for basic replay protection). As in the previous message, the MIC value ensures the message's integrity and at the same time it is a proof for the STA that the AP holds the same PMK.

The last message is an acknowledgement send from STA to AP to synchronize the start of encrypted communication. Both STA and AP install the derived keys and are then ready to communicate securely.

Both STA and AP have the possibility to cache the PMK. This is useful if a STA exits the network and wants to join again later. If the cached PMK is still valid, only the 4-Way Handshake protocol has to be executed, omitting 802.1X EAP authentication and thus saving time. Using nonces in the PTK derivation ensures that the derived PTK is different each time, even when using the same PMK.

Group Handshake Protocol

When a STA exits the network, the corresponding PTK is deleted. But still, we have to make sure that it can no longer decrypt multicast communication. This means a new GTK is needed and each STA in the network needs to install this new GTK. The Group Handshake Protocol is defined for this purpose. Its steps are shown in Figure 20.

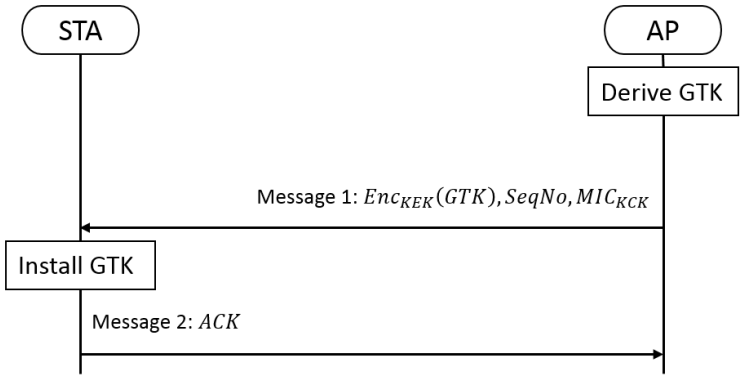

Fig. 20. Group Handshake Protocol (derived from Figure 11d in [32])

First, a new GTK is derived by the AP. Then the AP sends each STA in the network a message containing the new GTK, encrypted using the KEK, a sequence number that is used in the next multicast message from the AP (for basic replay protection) and a MIC value over the whole message, using the KCK. The KEK and KCK are different for each STA as they are part of the PTK and each STA shares a different PTK with the AP.

Each STA installs the new GTK and sends an acknowledgement back to the AP. When the AP has received all acknowledgement messages, it can use the new GTK to protect multicast communication.

Even though Figure 17 indicates that a part of the GTK is used for multicast communication originating from a STA (lower right corner of the figure), this part of the GTK is not used at the moment, because such multicast communication is not yet provided.

Key Derivation

So far, we simply said that the PTK is derived from the PMK and the GTK from the GMK. To do this, a pseudo-random function (PRF) is defined:

```
H-SHA1(K,A,B,X) = HMAC-SHA1(K,A‖Y‖B‖X)

PRF(K,A,B,Len):
   for i=0 to (Len+159)/160 do
      R = R ‖ H-SHA1(K,A,B,i)
   return L(R,0,Len)
```

Where `Y` is a single octet containing 0, `X` is a single octet containing the parameter, ‖ denotes concatenation and `L(R,0,Len)` returns the first `Len` bits of `R`.

With this PRF on hand, the keys are derived the following way.

- TKIP PTK = PRF(PMK, "Pairwise key expansion",
  Min(AP MAC-Adr, STA MAC-Adr) ‖
  Max(AP MAC-Adr, STA MAC-Adr) ‖
  Min(ANonce, SNonce) ‖
  Max(ANonce, SNonce), 512)
- TKIP GTK = PRF(GMK, "Group key expansion",
  AP MAC-Adr ‖ GNonce, 256)
- CCMP PTK = PRF(PMK, "Pairwise key expansion",
  Min(AP MAC-Adr, STA MAC-Adr) ‖
  Max(AP MAC-Adr, STA MAC-Adr) ‖
  Min(ANonce, SNonce) ‖
  Max(ANonce, SNonce), 384)
- CCMP GTK = PRF(GMK, "Group key expansion",
  AP MAC-Adr ‖ GNonce, 128)

The Group nonce (GNonce) is a random or pseudo-random value contributed by the 802.1X authenticator.

5.2 Data Confidentiality and Integrity

After authentication is done and all keys are in place, secure communication can start. To do so, a suitable protocol has to be used. This section describes the the two protocols which were introduced by 802.11i for that purpose.

MSDUs and MPDUs

Before we can take a look at the data confidentiality and integrity protocols, the terms MAC service data unit (MSDU) and MAC protocol data unit (MPDU) have to be briefly explained. MSDU denotes the unit in which data is received by

and handed to the upper layer. Before handing data to the lower level for wireless transmission, MSDU's might be split into multiple smaller MPDU's. When receiving MPDU's, they are re-assembled to retrieve the original MSDU's. Just like MSDU's, each MPDU also has a MAC header containing control and routing information.

Temporal Key Integrity Protocol

In 2001, Fluhrer, Mantin and Shamir published a paper on the weaknesses in the key scheduling algorithm of RC4[37]. Soon after, it was shown that these weaknesses could be used to completely break the Wired Equivalent Privacy protocol (WEP) in practice [38], leading to a need of a secure alternative. The Temporal Key Integrity Protocol (TKIP) was developed as a stop-gap solution, with the need to run on computational weak legacy hardware. It is implemented as part of Wi-Fi Protected Access (WPA). Since TKIP is already deprecated at the time of writing this work [39], we will not go too much into detail here.

Like WEP, TKIP still uses the RC4 cipher, but several features are added to correct the flaws of WEP. For an overview of WEP's design flaws, take a look at [40].

Figure 21 provides an overview of how TKIP encapsulation works.

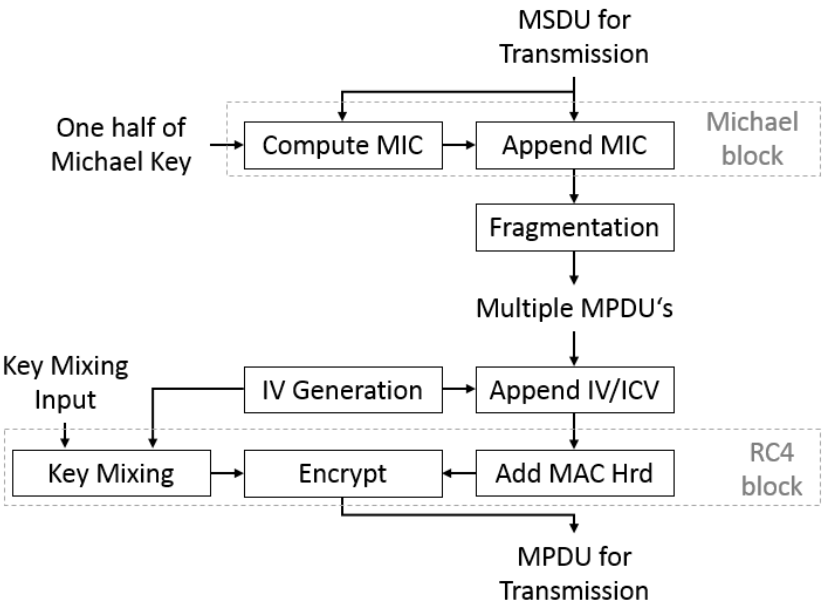

Fig. 21. TKIP encapsulation (derived from Figure 11.4 in [34])

First, a MIC value is computed and appended to the MSDU. The used MIC function is Michael, which was specially designed for TKIP. It only uses substitutions, rotations, and XOR operations for its calculation. If the MSDU is too large for transmission, it is split into multiple MPDU's in the next step. Then the initialization vector (IV) and an integrity check value (ICV) are appended to the MPDU. The IV needs to be included, because the receiver needs it as input for the key mixing function when decrypting the data. The ICV is an adopted feature of WEP. It is a CRC32 check sum of the MPDU. Lastly, a MAC header is prepended and the data, ICV and MIC (only present in the last MPDU, because the MIC value is calculated for the MSDU) parts of the MPDU are encrypted using the output of the key mixing function. Key Mixing produces a new key for every frame that is encrypted. This is a protection against RC4 weak key attacks and used to incorporate the extra bits of the extended IV (48 bits, which is double the size of the WEP IV).

Decapsulation is shown in Figure 22.

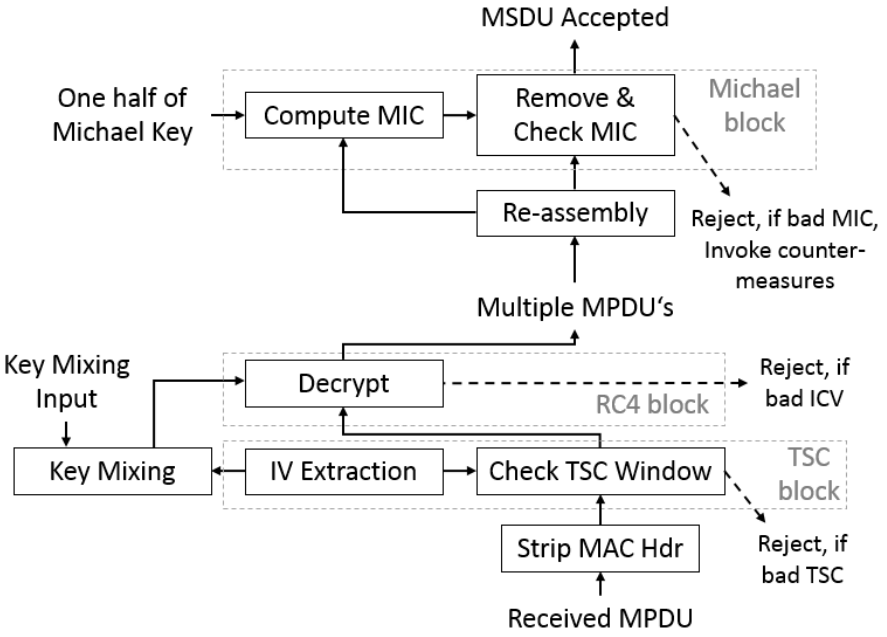

Fig. 22. TKIP decapsulation (derived from Figure 11.5 in [34])

The first thing to check is the TKIP Sequence Counter Window (TSC). This is done for replay protection. Then the MPDU is decrypted using the output of the key mixing function. Now the ICV is calculated and compared to the included one. If everything is fine up to this point, the MPDU's are re-assembled to retrieve a MSDU. The MIC value of the MSDU is computed and compared to

the included one. If the MIC check is positive, the MSDU is accepted. If it fails, countermeasures might be invoked (i.e. hold MPDU processing for 60 seconds, change PTK or GTK).

Counter with CBC-MAC Protocol

Counter with CBC-MAC Protocol (CCMP) is the long-term replacement for WEP. It was designed from scratch, without the need to run on legacy hardware. Therefore, the goal was a protocol that provides the best possible protection. CCMP uses the block cipher mode *Counter with CBC-MAC (CCM)* [41] in conjunction with AES-128. CCM is an authenticated encryption mode, which means confidentiality, integrity and authenticity of data are provided using a single key. Additionally, CCMP also provides replay protection. It is implemented as part of Wi-Fi Protected Access 2 (WPA2).

CCM mode has two parameters M and L. CCMP uses $M = 8$ and $L = 2$ as values for these parameters. The first parameter indicates that the MIC value has a length of eight octets. The second one is the size of the length field. Two octets are enough to hold the length of the largest possible IEEE 802.11 MPDU.

Unlike TKIP, all steps of CCMP happen at MPDU level. The first step in CCMP is to construct a CCMP header. The header size is eight octets and it is composed of a six octet Packet Number (PN), one reserved octet and a one octet KeyId. The PN is a non-negative integer that is incremented for each processed MPDU. That means, during the operating time of a PTK or GTK, each PN value must only be used once. For the PTK and each GTK an own PN is maintained. On start-up or when refreshing a PTK or GTK, the corresponding PN is initialized to 1. The usage of a PN provides replay protection. The reserved octet is for future extensions and the KeyId indicates which GTK was used in multicast communication.

Next, the MIC value is computed using CBC-MAC. This is done by encrypting data in *Cipher Block Chaining (CBC)* mode and using the last block of the chipertext as output. Since CCMP uses AES-128, but an eight octet MIC value, only the first half of the CBC-MAC output is used as MIC. In CCMP, the input for the MIC calculation is not simply the data portion of the MPDU, but rather a construction of `1st block` ‖ `MAC Header` ‖ `CCMP Header` ‖ `Pad` ‖ `PlaintextData` ‖ `Pad`. CCM mode defines the term *Additional Authenticated Data (AAD)*, which is any kind of data that is authenticated, but not encrypted. In the case of CCMP, the AAD is the MAC and the CCMP header. If the length of the AAD or the plaintext data is not a multiple of the block length, they get padded with zeros. The `1st block` is specially constructed as shown in Figure 23.

The `Flag` field has a fixed value of 01011001 for CCMP. The `Priority` field is used to assign a priority to a frame. The `Source Address` is the MAC address of the sender. It is extracted from the MAC header. The `PN` is used like previously described. The `DLen` field indicates the length of the plaintext data.

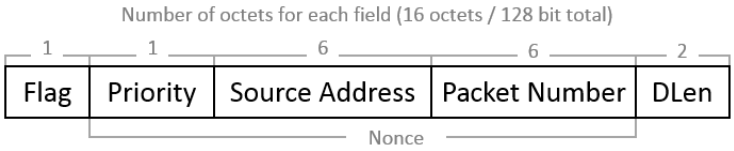

Fig. 23. Structure of the first block for CBC-MAC (derived from Figure 12.10 in [34])

The source address is included in the nonce, because otherwise the same nonce value might be used several times for the same PTK or GTK, as these keys are shared between STA(s) and AP.

After calculating the MIC value, encryption takes place, using AES-128 in Counter (CTR) mode. The input that gets encrypted is `PlaintextData` ‖ `MIC`. No padding is needed at this point, because CTR mode turns a block cipher into a stream cipher. In CTR mode, a counter block is encrypted using a block cipher and a corresponding secret key. The result of this step and a block of the data to encrypt are combined using XOR to produce one block of the ciphertext. For each block processed this way, the value of the counter block is incremented. When the last portion of data does not have block length, only the needed number of bits of the intermediate result is used for the XOR operation. Figure 24 illustrates how the counter block is constructed for CCMP.

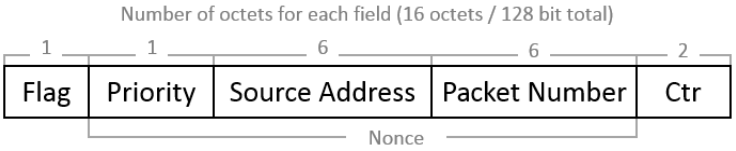

Fig. 24. Structure of the counter block (derived from Figure 12.11 in [34])

The value of the `Ctr` field is initialized with 1. It is incremented for each processed block in CTR mode. The other fields are the same as in the first block for the CBC-MAC. Including a nonce as part of the counter block ensures that encrypting the same data more than once always results in a different ciphertext.

The MPDU composed of `MAC Header` ‖ `CCMP Header` ‖ `Ciphertext` is then ready for transmission. Upon receiving such a MPDU, the receiver first checks the PN in the CCMP header. If it is not greater than the PN of the previously received MPDU, the MPDU is discarded for replay protection. The next step is to decrypt the ciphertext. To do so, the receiver extracts the source address from the MAC header and the PN from the CCMP header. With this information, he can build the same counter block the sender used for encryption and thus decrypt the ciphertext, since he also holds the same key. Then the MIC value of the MPDU is calculated and compared to the included one. If the MIC is valid,

CCMP header and MIC are removed from the MPDU, which is then ready for re-assembly to retrieve a MSDU.

5.3 Known Attacks

WPA and WPA2 both use the 4-Way Handshake protocol in the authentication process. Due to the fact that the first message in the 4-Way Handshake is not protected in any way, the protocol is vulnerable to Denial of Service (DoS) attacks [42]. If an attacker sends a forged first message between the actual first and third message, the handshake fails. This can be used to launch DoS attacks.

The 4-Way Handshake protocol causes another problem when using pre-shared keys for authentication [43]. A PSK is usually derived from a passphrase. When an attacker listens to the network traffic and records the messages of the 4-Way Handshake, he gets to know ANonce and SNonce, because the messages are send unencrypted. With this information, he can launch a dictionary attack. The entries of the dictionary are the possible passphrases, which are used by the attacker to derive possible PSKs. These possible PSKs and the known ANonce and SNonce are then used to derive possible PTKs. For each of these PTKs, the attacker then checks if he is able to calculate the same MIC values as in the recorded handshake messages. If he succeeds, it is highly likely that he found the correct passphrase and thus now holds the correct PTK, which enables him to decrypt the communication between STA and AP. This attack can easily be prevented by using a strong passphrase (suitable length, use of digits, uppercase, lowercase and special characters).

There are a few known attacks against TKIP. In 2008, Tews and Beck were able to inject seven forged packets with custom content into an WPA protected network, passing the Michael MIC check [44]. However, this attack has several limitations. It takes about 12-15 minutes, the size of the forged packets is limited and Quality of Service (QoS) needs to be enabled. Beck improved this attack in 2010, allowing more and longer forged packets to be injected into the network [45]. Ohigashi and Morii also improved the Beck-Tews attack in 2009 [46]. They combined the original attack with a man-in-the-middle approach, reducing its limitations. QoS is not required and executing the attack takes about one minute in the best case.

In [45], Beck also presented a novel attack against Michael, which allows an attacker to concatenate a forged with an unknown valid TKIP packet in such a way that the MIC of the valid TKIP packet is still valid for the new entire packet.

Up to now, there are no known attacks against CCMP.

5.4 802.11i in IoT

The presented protocols are heavily relevant for the Internet of Things. There are a lot of IoT use cases, in which wireless networks are used. Most of the time, such a wireless network is used in the form of a IEEE 802.11 network,

involving at least the 4-Way Handshake and TKIP or CCMP. However, there are exceptions. One of them is described in section 6.1.

As for lightweight implementations of the presented protocols, researchers have proposed improved variants of both TKIP [47, 48] and CCMP [49, 50]. The problem with using CCM mode in CCMP is the fact that CCM has to perform AES encryption on the data once to get the CBC-MAC and then a second time for the actual encryption. This is inefficient. The proposed improvements try to tackle this problem. Another possibility would be not to use CCM mode in 802.11 networks at all, but rather some other authenticated encryption scheme, since there are several ones with better performance than CCM.

6 Comparison

In this section, the usage of the presented protocols IPSec, DTLS and IEEE 802.11i for IoT is reviewed. Based on an IoT scenario, which involves different areas of application, the utilization is examined and the advantages and limitations of the protocols are discussed.

6.1 Scenario

The considered scenario involves an automotive environment and the communication within a car itself, between different cars and with the infrastructure. The aspect of how the presented protocols are applicable is discussed. As a short introduction to the automotive environment and the major communication channels, a quick overview is given first.

In the future, cars will be involved in the following networks:

- Car-to-Car ad-hoc networks (C2C)
- Car-to-Infrastructure networks (C2I)
- Inter-Car-Communication networks
- Car-to-Facility and Car-to-Company networks

For each of these networks, different security protocols are more or less suitable. Ad-hoc networks, like a C2C network, need to establish a secure communication between parties which do not know each other at first. It is important that these networks have a high performance in connection establishment as well as data transfer.

Inter-Car-Communication networks are networks with a fixed number of parties, but the parties themselves have to deal with limitations in computing power and memory.

Car-to-Facility and Car-to-Company networks connect the cars to manufacturers or other companies. The connection between cars and companies respectively manufacturers are not time critical, but still needs to be highly secure.

DTLS in the scenario DTLS is suitable for Inter-Car-Communication, Car-to-Facility and Car-to-Company networks but less applicable in C2C ad-hoc networks and C2I networks due to its quickly changing parties.

As described in section 3.9, lightweight versions of DTLS can be implemented using pre-shared keys. This enables DTLS to be used in Inter-Car-Communication networks. The setup of DTLS in an Inter-Car-Communication network could be done once, such that algorithms and keys are already negotiated and the handshake can be optimized. Also the memory efficiency on the devices in inter car networks makes DTLS even more applicable. Small low power sensors can simply transmit data to other devices inside the car.

When connecting a car to companies or facilities, DTLS is fully applicable to secure the communication. Stateless cookies between companies and cars can be used in the handshake. Also, there are less restricted power assumptions in networks like these, since the endpoints are the main control unit in a car and and the hosts of companies or facilities.

For C2C ad-hoc networks and C2I networks, the applicability is controversial. Due to its quickly changing parties, an ad-hoc network between cars needs to be established fast. Also, authentication needs to be established with unknown partners. Hence, pre-shared keys and cookies are not applicable in this type of networks. The four rounds in the handshake to negotiate keys and algorithms are considered to be expensive, but the advantages of datagram communication are very important. Datagram-based communication is faster than TCP-based communications via TLS, since no acknowledgments have to transmitted back and forth. After all, performance reasons and the lossy communication with UDP disqualify DTLS for C2C ad-hoc networks. In case of message loss during the handshake, retransmission in the handshake protocol uses timer values about one second or longer. In C2C communication, message exchange should be performed faster, since one scenario could be two cars driving in opposing directions.

Connections between a car and its infrastructure vary a lot, since a car is mobile by its design. Therefore, the applicability of DTLS is similar as for the C2C ad-hoc networks. Quickly changing parties make it impossible to use pre-shared keys. Raw public keys could be a solution to reduce message complexity, but the three round-trip handshake is still very expensive. It would take respectively long to establish an authenticated, secure communication between two parties and hence, DTLS is not recommended for C2I networks either.

IPSec in the scenario IPSec can be used in IoT context for Inter-Car-Communication, Car-to-Facility and Car-to-Company networks. SAs cannot be established with unknown peers without involving a trusted third party. Hence, IPSec does not seem to be suitable for ad-hoc networks.

The lightweight variants of IPSec can be used in networks with power and memory restrictions, i.e. in sensor networks. In in-vehicle networks, the identity of all parties can be pre-shared and the SAs can be pre-setup. It is important that the algorithms can be implemented in hardware in order to perform the de- and encryption more efficiently. This allows fast communication without the need for

IKE handshakes. The keys can be re-negotiated in defined intervals to ensure confidentiality and perfect forward secrecy. IPSec should be used in transport mode with AH in most cases to prevent face data injection. ESP should only be used if the information has to be kept confidential.

For Car-to-Company and Car-to-Facility networks, IPSec can be used in the standard variant. Public-Key cryptography can be used to identify the communication party as defined in IKE standards. Only the main control unit needs to communicate with facilities and companies. This allows the usage of standard IPSec. If sensor data is requested by facilities, the main control unit should act as a information aggregator to prevent external parties from injecting content into the car's internal network. This means not that the connection should be protected with IPSec in Tunnel mode. It means that the external identity should not be able to address sensors, actuators or other entities in the vehicle at all. The communication should be protected with AH and ESP to achieve maximum security.

IEEE 802.11i in the scenario The protocols defined by IEEE 802.11i can be utilized in Inter-Car-Communication. Although such communication happens mostly over a wired connection nowadays, it is imaginable to establish a Wireless LAN network inside a car, which then could be used to connect the individual parts.

For Car-to-Facility and Car-to-Company communication, the applicability of the 802.11i protocols depends on the actual setup. Most likely this is a combination of C2I communication and a wired connection from the Roadside Unit (RSU) to the facility or company. If this is the case, the 802.11i protocols are not relevant. The other possibility is that the RSU does not have a wired access to the internet itself, but is connected to some AP via an 802.11 network. In this case, the protocols defined by 802.11i are used to secure the data traffic.

C2C ad-hoc networks and C2I networks both use Wireless LAN connections for communication, so one might think that the 802.11i protocols apply here. However, this is not the case. The mobility aspect in these networks requires extremely low latencies in communication. The protocols defined by 802.11i are not suitable for such requirements. Instead, C2C and C2I communication makes use of IEEE 802.11p [51] and the IEEE 1609 family [52]. The 802.11p amendment introduces a new functionality, allowing a STA to transmit data frames outside the context of a Basic Service Set (BSS). This mode does not utilize the IEEE 802.11 authentication, association or data confidentiality services. These services are moved to the network, transport layer or session layer, as described in IEEE 1609.2 [53].

6.2 Discussion

As described above, different problems occur when applying the protocols to the Car-to-X networks. Each protocol has its own strengths and limitations. Lightweight variants help to apply the protocols to more constrained networks,

but in some cases, basic design decisions of the protocols make them non-applicable.

Table 1 shows which protocol is applicable to which type of network (marked by an ×).

Table 1. Comparison of the applicability of the protocols

Area of application	**DTLS**	**IPSec**	**IEEE 802.11i**
Car-to-Car ad-hoc networks	-	-	-
Car-to-Infrastructure networks	-	-	-
Inter-Car-Communication networks	×	×	×
Car-to-Company networks	×	×	(×)
Car-to-Facility networks	×	×	(×)

Due to quickly changing parties and the need for fast connection establishment, the IEEE 802.11i protocols and IPSec are not applicable in C2C ad-hoc networks, as well as in C2I networks. While IPSec uses SAs, which cannot be established without a third party, the 802.11i protocols can not guarantee low latency communication. For C2C or C2I, IEEE 802.11p and the IEEE 1609 family are recommended instead. This way, security is handled on an upper layer, for example on the Network or Transport Layer. The connection establishment in DTLS is done in a four round-trip handshake and therefore is considered expensive. Its datagram-based communication on the other hand is very efficient and therefore, very suitable for C2C and C2I networks. Overall, lossy datagram-based communication and long retransmission timers make DTLS not applicable to C2C and C2I networks.

Inter-Car-Communication networks contain sensors, actuators and other entities in the vehicle itself. Often small sensors and control units work on low power assumptions and have limited memory. Lightweight variants of all protocols are applicable to those kinds of networks. Compression approaches enable protocols to be used with 6LowPAN. Also, the fixed number of participants makes it even easier to apply lightweight versions, since they often use pre-shared keys and identities.

In Car-to-Company and Car-to-Facility networks, communication needs to be secured, but performance and power restrictions are less critical. While IPSec and DTLS are applicable without any changes, the 802.11i protocols can be used if the connection is required to be wireless. If a wired connection is given, the IEEE 802.11i protocols are not relevant. For both types of networks, lightweight variants can be considered, but it is not necessary since the communications are between the main control unit of the car and the host of the facility or company, which are both considered to be computational powerful.

In Table 2 the properties of the proposed protocols are compared. In the case of IPSec the security goals confidentiality, integrity, and authenticity can be achieved. If IPSec is configured properly it can be used for authorization as well. Since IKEv2 deprecated the DoS prevention cookie IPSec can not ensure

availability. Even with the DoS prevention cookies it seams impossible to ensure availability in 6LoWPAN networks based on the power, memory and computing power constraints and the network architecture. The additionally needed power, computing power and memory for IPSec in 6LoWPAN is mainly determined by the used algorithms. IPSec in 6LowPAN has an overhead of 1 byte for each packet during communication by only concerning the headers without the integrity checksums. That seams to be fair by concerning the security benefits. IPSec ensures Device-to-Device security.

For DTLS, the security goals of confidentiality, integrity and authenticity are supported. It has never been suggested to use DTLS for authorization. Since DTLS the availability is dependent on a successful handshake and DTLS retransmission timers are usually configured with a large delay, DTLS cannot achieve full availability in constrained networks. Due to mentioned IoT implementations of DTLS, power and memory consumption as well as computation time are depended on the implementation chosen for the network. While TLS uses fragmentation and hence has a fragementation overhead, DTLS does not have any fragmentation overhead during communication (except in the handshake, but this is not of major concern). Therefore, DTLS is considered to have no overhead during communication. Finally, DTLS is suitable for application-to-application security since it is located in the Session Layer.

IEEE 802.11i provides confidentiality, integrity and authenticity for wireless communication. Authorization is not a supported feature. Due to the nature of wireless communication, availability can not be guaranteed. Radio signals can be maliciously disrupted, which is called jamming, or densely crowded radio bands might cause interference. Additionally, the authentication process of IEEE 802.11i is prone to Denial of Service attacks as described in section 5.3. Power and memory consumption as well as computation time depends on whether TKIP or CCMP is used. In general AES used by CCMP is more expensive in terms of computation time than RC4 used by TKIP. However, TKIP was designed to run on legacy hardware, which is optimized for WEP. On modern routers equipped with AES optimized hardware, CCMP is the 'faster' protocol. Regarding message overhead, TKIP adds 8 Bytes per MSDU for the MIC value and 4 Bytes per MPDU for the extension of the WEP IV. CCMP extends each MPDU by 16 Bytes, 8 Bytes for the CCMP header field and 8 Bytes for the MIC value. IEEE 802.11i ensures Hop-to-Hop security in multi-hop networks.

Overall, the applicability of the three examined protocols strongly depends on the given restrictions and the interest in performance. When establishing connections with unknown but quickly changing parties, i.e. in C2C or C2I, performance is most important. For Inter-Car-Communication, performance is still important, but low power assumptions determine the network specifications and therefore, lightweight variants are required. In networks between cars and facilities or companies, less restricted networks are considered and therefore all protocols are suitable.

Table 2. Comparison of the protocol properties (with abbreviations 'Impl.' for Implementation and 'App.' for Application)

Property	**DTLS**	**IPSec**	**IEEE 802.11i**
Confidentiality	×	×	×
Integrity	×	×	×
Authenticity	×	×	×
Authorization	-	×	-
Availability	-	-	-
Power Consumption	Impl. depended	Algorithm depended	Algorithm depended
Computation Time	Impl. depended	Algorithm depended	Algorithm depended
Memory Consumption	Impl. depended	Algorithm depended	Algorithm depended
Overhead	-	1 Byte	Algorithm depended
End-to-End Type	App-to-App	Device-to-Device	Hop-to-Hop

7 Conclusion

The Internet of Things is rapidly evolving. More and more devices are connected and networks are growing constantly. Securing data traffic is a crucial point for the success and acceptance of IoT. To do so, suitable protocols are needed.

This work presented the protocols DTLS, IPSec and the ones defined by IEEE 802.11i, namely 4-Way Handshake and Group Handshake Protocol, Temporal Key Integrity Protocol and Counter with CBC-MAC Protocol. For each of these protocols an explanation was given on how they work and what they are used for. Additionally, lightweight variants were presented and the general applicability in an IoT context was discussed. DTLS, IPSec and the IEEE 802.11i protocols are located on different layers and have been selected to emphasize that security can be established on different layers with different properties.

The work is concluded by the description of a concrete IoT scenario, for which we examined how each protocol can be applied in this context.

DTLS, IPSec and the IEEE 802.11i protocols are not the only protocols that are used to secure communication. Therefore, future work should take a look at more protocols, examine their lightweight variants and analyze the applicability in an IoT context as well and compare them to the work at hand.

References

1. John Clark and Jeremy Jacob. A survey of authentication protocol literature: Version 1.0, 1997.
2. Ian F Akyildiz, Weilian Su, Yogesh Sankarasubramaniam, and Erdal Cayirci. A survey on sensor networks. *Communications magazine, IEEE*, 40(8):102–114, 2002.
3. Kemal Akkaya and Mohamed Younis. A survey on routing protocols for wireless sensor networks. *Ad hoc networks*, 3(3):325–349, 2005.
4. Vehbi C. Güngör, Dilan Sahin, Taskin Kocak, Salih Ergüt, Concettina Buccella, Carlo Cecati, and Gerhard P Hancke. Smart Grid Technologies: Communication Technologies and Standards. *Industrial informatics, IEEE transactions on*, 7(4):529–539, 2011.

5. RFC 4347. https://www.ietf.org/rfc/rfc4347.txt. Accessed: 2016-01-06.
6. RFC 6347. https://www.ietf.org/rfc/rfc6347.txt. Accessed: 2016-01-06.
7. Hubert Zimmermann. Osi reference model-the iso model of architecture for open systems interconnection. *IEEE Transactions on communications*, 28(4):425–432, 1980.
8. Nagendra Modadugu and Eric Rescorla. The Design and Implementation of Datagram TLS. In *NDSS*, 2004. Accessed: 2016-01-06.
9. RFC 4346. https://www.ietf.org/rfc/rfc4346.txt. Accessed: 2016-01-06.
10. RFC 5246. https://www.ietf.org/rfc/rfc5246.txt. Accessed: 2016-01-06.
11. Nadhem J Al Fardan and Kenneth G Paterson. Lucky thirteen: Breaking the tls and dtls record protocols. In *Security and Privacy (SP), 2013 IEEE Symposium on*, pages 526–540. IEEE, 2013.
12. RFC 7457. https://www.ietf.org/rfc/rfc7457.txt. Accessed: 2016-01-06.
13. Nadhem J AlFardan and Kenneth G Paterson. Plaintext-recovery attacks against datagram tls. In *Network and Distributed System Security Symposium (NDSS 2012)*, 2012.
14. Vishwas Lakkundi and Keval Singh. Lightweight dtls implementation in coap-based internet of things. In *Advanced Computing and Communications (ADCOM), 2014 20th Annual International Conference on*, pages 7–11. IEEE, 2014. Accessed: 2016-01-23.
15. Sye Loong Keoh, Sahoo Subhendu Kumar, and Hannes Tschofenig. Securing the internet of things: A standardization perspective. *Internet of Things Journal, IEEE*, 1(3):265–275, 2014. Accessed: 2016-01-23.
16. O Bergmann. Tinydtls software library implementation. Accessed: 2016-01-23.
17. Shahid Raza, Hossein Shafagh, Kasun Hewage, René Hummen, and Thiemo Voigt. Lithe: Lightweight secure coap for the internet of things. *Sensors Journal, IEEE*, 13(10):3711–3720, 2013. Accessed: 2016-01-23.
18. Shahid Raza, Daniele Trabalza, and Thiemo Voigt. 6lowpan compressed dtls for coap. In *Distributed Computing in Sensor Systems (DCOSS), 2012 IEEE 8th International Conference on*, pages 287–289. IEEE, 2012. Accessed: 2016-01-23.
19. IETF DICE Working Group. https://datatracker.ietf.org/wg/dice/charter/. Accessed: 2016-01-23.
20. IETF CoRE Working Group. https://datatracker.ietf.org/wg/core/documents/. Accessed: 2016-01-23.
21. Jiye Park and Namhi Kang. Lightweight secure communication for coap-enabled internet of things using delegated dtls handshake. In *Information and Communication Technology Convergence (ICTC), 2014 International Conference on*, pages 28–33. IEEE, 2014. Accessed: 2016-01-23.
22. RFC 4301. https://tools.ietf.org/html/rfc4301. Accessed: 2016-01-06.
23. RFC 4302. https://tools.ietf.org/html/rfc4302. Accessed: 2016-01-06.
24. RFC 4303. https://tools.ietf.org/html/rfc4303. Accessed: 2016-01-06.
25. Jean-Philippe Vasseur and Adam Dunkels. *Interconnecting Smart Objects with IP: The Next Internet*. Morgan Kaufmann, 2010. Accessed: 2016-01-06.
26. 6LoWPAN: The wireless embedded Internet - Part 1: Why 6LoWPAN? http://www.eetimes.com/document.asp?doc_id=1278794. Accessed: 2016-02-07.
27. RFC 6282. https://tools.ietf.org/html/rfc6282. Accessed: 2016-01-06.
28. Jorge Granjal, Ricardo Silva, Edmundo Monteiro, Jorge Sa Silva, and Fernando Boavida. Why is IPSec a viable option for Wireless Sensor Networks. In *Mobile Ad Hoc and Sensor Systems, 2008. MASS 2008. 5th IEEE International Conference on*, pages 802–807. IEEE, 2008. Accessed: 2016-01-06.

29. Shahid Raza, Thiemo Voigt, and Vilhelm Jutvik. Lightweight IKEv2: A Key Management Solution for both the Compressed IPsec and the IEEE 802.15.4 Security. In *Proceedings of the IETF workshop on smart object security*, 2012. Accessed: 2016-01-06.
30. Shahid Raza, Tony Chung, Simon Duquennoy, Thiemo Voigt, Utz Roedig, et al. Securing Internet of Things with Lightweight IPsec. 2010. Accessed: 2016-01-06.
31. Shahid Raza, Simon Duquennoy, Tony Chung, Thiemo Voigt, Utz Roedig, et al. Securing Communication in 6LoWPAN with Compressed IPsec. In *Distributed Computing in Sensor Systems and Workshops (DCOSS), 2011 International Conference on*, pages 1–8. IEEE, 2011. Accessed: 2016-01-06.
32. IEEE 802.11i-2004. `https://standards.ieee.org/findstds/standard/802.11i-2004.html`. Accessed: 2016-01-31.
33. IEEE 802.11-2007. `https://standards.ieee.org/findstds/standard/802.11-2007.html`. Accessed: 2016-01-31.
34. Edney and William A. Arbaugh. *Real 802.11 Security: Wi-Fi Protected Access and 802.11i*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2003.
35. IEEE 802.1X. `https://standards.ieee.org/findstds/standard/802.1X-2010.html`. Accessed: 2016-02-23.
36. RFC 2104. `https://tools.ietf.org/html/rfc2104`. Accessed: 2016-02-25.
37. Scott R. Fluhrer, Itsik Mantin, and Adi Shamir. Weaknesses in the Key Scheduling Algorithm of RC4. In *Revised Papers from the 8th Annual International Workshop on Selected Areas in Cryptography*, SAC '01, pages 1–24, London, UK, 2001. Springer-Verlag.
38. Adam Stubblefield, John Ioannidis, and Aviel D. Rubin. Using the Fluhrer, Mantin, and Shamir Attack to Break WEP. In *Proceedings of the Network and Distributed System Security Symposium, NDSS 2002, San Diego, California, USA*, 2002.
39. Technical Note - Removal of TKIP from Wi-Fi Devices. `http://www.wi-fi.org/download.php?file=/sites/default/files/private/Wi-Fi_Alliance_Technical_Note_TKIP_v1.0.pdf`. Accessed: 2016-03-02.
40. Nikita Borisov, Ian Goldberg, and David Wagner. Intercepting Mobile Communications: The Insecurity of 802.11. In *Proceedings of the 7th Annual International Conference on Mobile Computing and Networking*, MobiCom '01, pages 180–189, New York, NY, USA, 2001. ACM.
41. RFC 3610. `https://tools.ietf.org/html/rfc3610`. Accessed: 2016-02-27.
42. X. Zha and M. Ma. Security improvements of IEEE 802.11i 4-way handshake scheme. In *Communication Systems (ICCS), 2010 IEEE International Conference on*, pages 667–671, Nov 2010.
43. Weakness in Passphrase Choice in WPA Interface. `http://wifinetnews.com/archives/2003/11/weakness_in_passphrase_choice_in_wpa_interface.html`. Accessed: 2016-03-02.
44. Erik Tews and Martin Beck. Practical Attacks Against WEP and WPA. In *Proceedings of the Second ACM Conference on Wireless Network Security*, WiSec '09, pages 79–86, New York, NY, USA, 2009. ACM.
45. Martin Beck. Enhanced TKIP michael attacks. *CoRR*, abs/1410.6295, 2014.
46. A Practical Message Falsification Attack on WPA. `http://jwis2009.nsysu.edu.tw/location/paper/A%20Practical%20Message%20Falsification%20Attack%20on%20WPA.pdf`. Accessed: 2016-03-02.
47. M. Razvi Doomun and K. M. Sunjiv Soyjaudah. Modified Temporal Key Integrity Protocol for Efficient Wireless Network Security. In *SECRYPT 2007, Proceedings of the International Conference on Security and Cryptography, Barcelona, Spain,*

July 28-13, 2007, SECRYPT is part of ICETE - The International Joint Conference on e-Business and Telecommunications, pages 151–156, 2007.
48. M. Razvi Doomun and K. M. Sunjiv Soyjaudah. LOTKIP: Low Overhead TKIP Optimization for Ad Hoc Wireless Networks. *International Journal of Network Security*, 10:225–237, May 2010.
49. Zadia Codabux-Rossan and M. Razvi Doomun. AES CCMP Algorithm with N-Way Interleaved Cipher Block Chaining. *CoRR*, abs/1208.3576, 2012.
50. M. Razvi Doomun and K. M. Sunjiv Soyjaudah. Resource Saving AES-CCMP Design with Hybrid Counter Mode Block Chaining - MAC. *International Journal of Computer Science and Network Security*, 8(10):1–13, Oct 2008.
51. IEEE 802.11p-2010. `http://standards.ieee.org/findstds/standard/802.11p-2010.html`. Accessed: 2016-03-05.
52. IEEE 1609.0-2013. `http://standards.ieee.org/findstds/standard/1609.0-2013.html`. Accessed: 2016-03-05.
53. IEEE 1609.2-2013. `http://standards.ieee.org/findstds/standard/1609.2-2013.html`. Accessed: 2016-03-05.