# A DTLS Based End-To-End Security Architecture for the Internet of Things with Two-Way Authentication

Thomas Kothmayr*, Corinna Schmitt*, Wen Hu†, Michael Brünig† and Georg Carle*
*Department of Computer Science, Chair for Network Architectures and Services,
Technische Universität München, Germany. Email: kothmayr@in.tum.de, {schmitt, carle}@net.in.tum.de
†CSIRO ICT Centre, Brisbane, Australia. Email: {wen.hu, michael.bruenig}@csiro.au

*Abstract*—**In this paper, we introduce the first fully implemented two way authentication security scheme for the Internet of Things (IoT) based on existing Internet standards, especially the Datagram Transport Layer Security (DTLS) protocol. The proposed security scheme is based on the most widely used public key cryptography (RSA), and works on top of standard low power communication stacks. We believe that by relying on an established standard, existing implementations, engineering techniques and security infrastructure can be reused, which enables easy security uptake. We present an implemented system architecture for the proposed scheme based on a low-power hardware platform suitable for the IoT. We further demonstrate its feasibility (low overheads and high interoperability) through extensive evaluation.**

## I. Introduction

Today, there is a multitude of envisioned and implemented use cases for the IoT and wireless sensor networks (WSNs). It is desireable, in most of these scenarios, to also make the data globally accessible to authorized users and data processing units through the Internet. Naturally, much of the data collected in these scenarios is of a sensitive nature. Even seemingly inconspicuous data, such as the energy consumption measured by a smart meter, can lead to potential infringements on the users' privacy, e.g. by allowing an eavesdropper to conclude whether or not a user is currently at home. This risk is aggravated by the trend towards a separation of sensor network infrastructure and applications [1], [2]. Therefore, a true end-to-end security solution is required to achieve an adequate level of security for the IoT. Protecting the data once it leaves the scope of the local network is not enough.

A similar example in the traditional computing world would be a user browsing the Internet over an unsecured WLAN. Attackers in physical proximity of the user can capture the traffic between the user and a web server. Countermeasures against such attacks include the establishment of a secured connection to the webserver via HTTPS, the use of a VPN tunnel to securely connect to a trusted VPN endpoint and using wireless network security such as WPA.

These solutions are comparable to security approaches in the IoT area. Using WPA is similar to the traditional use of link layer encryption. The VPN solution is equivalent to creating a secure connection between a sensor node and a security endpoint, which may or may not be the final destination of the sensor data. Establishing a HTTPS connection with the server is comparable to our approach: We investigate the use of the DTLS protocol in an end-to-end security architecture for the IoT. DTLS is an adaption of the widespread TLS protocol, used to secure HTTPS, for unreliable datagram transport. By choosing DTLS we have made three high-level design decisions:

**Implementation of a standards based design:** Standardization has helped the widespread uptake of technologies. Radio chips can rely on IEEE 802.15.4 for the physical and the MAC layer. The IPv6 Routing Protocol for Low power and Lossy Networks (RPL) or 6LoWPAN provide routing functionality and CoAP [3] defines the application layer. So far, no such efforts have addressed security in a wider context for the IoT.

**Focus on application-layer end-to-end security:** An end-to-end protocol provides security even if the underlying network infrastructure is only partially under the user's control. As the infrastructure for Machine-to-Machine (M2M) communication is getting increasingly commoditized, this scenario becomes more likely: The European Telecommunications Standards Institute (ETSI) is currently developing a standard that focusses on providing a "horizontal M2M service platform" [2], meaning that it plans to standardize the transport of local device data to a remote data center. For stationary installations security functionality could be provided by the gateway to the higher level network. However, such gateways would present a high-value target for an attacker. If the devices are mobile, for example in an logistics application, there may be no gateway to a provider's network that is under the user's control, similar to how users of smartphones connect directly to their carrier's network. Another example that favours end-to-end security is a multi-tennacy office building that is equipped with a common infrastructure for metering and climate-control purposes. The tennants share the infrastructure but are still able to keep their devices' data private from other members of the network. Using a protocol like DTLS, which is placed between transport and application layer, does not require that the infrastructure provider supports the security mechanism. It is purely in the hands of the two communicating applications to establish security. If the security is provided by a network layer protocol, such as IPsec, the same is true to a lower degree because the network stacks of both devices must support the same security protocol.

**Support for unreliable transport protocols:** Reliable transport protocols like TCP incur an overhead over simpler, unreliable protocols such as UDP. Especially for energy starved, battery powered devices this overhead is often too costly and TCP has been shown to perform poorly in low-bandwidth scenarios [4]. This is reflected in the design of the emerging standard CoAP, which uses UDP transport and even defines a binding to DTLS for security [3]. By using DTLS in conjunction with UDP our approach does not force the application delevoper to use reliable transport - as would be the case if TLS would be used. It is still possible to use DTLS over transport protocols like TCP, since DTLS only assumes unreliable transport. This is a weaker property than the reliability provided by TCP. However, the adaptions of DTLS for unreliable transport introduce additional overhead when compared to TLS. Applications requiring reliable messaging cannot be sent with DTLS over UDP and must choose a reliable transport layer instead.

The remainder of this paper is organized as follows: We discuss related work in Section II and present a short introduction to DTLS (c.p. Section III) which is followed by the description of a standards-based security architecture over the whole lifecycle of an IoT deployment in Section IV. We evaluate the proposed DTLS IoT adaptation in a multi-hop testbed in Section V and conclude in Section VI.

## II. Related Work

Traditionaly, encryption and security protocols in sensor networks focus on link layer security, protecting data on a hop-by-hop basis. The simplest approach to link layer security consits of using a single network-wide encryption key, which often is the case in ZigBee networks [5]. ZigBee also provides support for cluster and individual link keys. MiniSec [6] is another well known link layer security mechanism for WSNs that provides data confidentiality, authentication and replay protection in both unicast and broadcast communications. As with ZigBee, the packet overhead introduced by MiniSec is in the order of a few bytes. The widespread TinySec link layer security mechanism is no longer considered secure [6].

Most security protocols do not include a mechanism for how encryption keys are distributed to the nodes. Keys are either loaded onto the nodes before setup or a separate key establishment protocol is used. Public key cryptography (PKC) is used in traditional computing to facilitate secure key establishment. However, public key cryptography, in particular the widespread RSA algorithm, has been considered too resource consuming for constrained devices. Some security protocols, such as Sizzle [7], advocate the use of the more resource efficient Elliptic Curve Cryptography (ECC) public key cryptosystem. Other research efforts, such as the SecFleck by Hu et al. [8], provide support for faster RSA operations through hardware.

Approaches without PKC often rely on the pre-distribution of connection keys. Random key pre-distribution schemes, such as the q-composite random key pre-distribution scheme by Chan et al. [9], establish connections with a node's neighbours with a certain probability $p < 1$. Intuitively, pre-distributed key schemes such as this require a large amount of keys to be loaded onto the nodes before deployment. Depending on the method used, this approach is scaling in $\mathcal{O}(n^2)$ or $\mathcal{O}(n)$ where $n$ is the number of nodes in the network. The Peer Intermediaries for Key Establishment protocol (PIKE) achieves sub linear scaling in $\mathcal{O}(\sqrt{n})$ by relying on the other nodes as trusted intermediaries. While PIKE provides higher memory efficiency than random schemes, it still leaks additional key information when motes are captured.

Recently, more research into end-to-end security protocols for the IoT and WSNs is being conducted. As outlined in the introduction, an end-to-end security protocol protects the message payload from the data source untill it reaches its target. Because this kind of protocol is usually implemented in the network or application layer, forwarding nodes do not need to perform any additional cryptographic operations since the routing information is transmitted in the clear. On the flip side, this means end-to-end security protocols do not provide the same level of protection of a network's avaliability as a link layer protocol could. One example of an end-to-end security protocol is Sizzle by Gupta et al. [7]. Sizzle is a compact web server stack providing HTTP services secured by SSL. It uses 160-bit ECC keys for key establishment which provide a similar level of security as 1024-bit RSA keys. In contrast to our work, it requires a reliable transport layer which has been shown to incur large performance penalties in low bandwidth situations [4]. Sizzle also omitts two-way authentication: Only the Sizzle enabled node is authenticated by a remote, more resource rich, client. This is insufficient for machine to machine communication in the IoT. SSNAIL [10] makes similar design choices as Sizzle and performs an ECC enabled handshake over reliable TCP transport. Similar to our implementation, SSNAIL is able to perform a full, two-way authenticated handshake but it still requires a reliable transport protocol. Additionally, their evaluation is lacking in-depth data about the handshake's performance. Raza et al. [11] discuss how the IPsec protocol can be integrated into 6LoWPAN, the compressed IPv6 implementation used in most IP-enabled sensor networks. Their work focuses on how data transfer with IPsec can be made efficient in the context of 6LoWPAN. Regarding the Internet Key Exchange protocol (IKE), which is used for key establishment in IPsec networks, Raza et al. [12] discuss methods for reducing the headers to make IKE more suitable for constrained devices, but do not present a performance analysis alongside their proposal.

As mentioned in the introduction, CoAP is an application layer standartization effort for the Internet of Things. The current draft specifies a binding of CoAP to DTLS to achieve security [3]. Another proposal by Raza et al. aims to reduce the communication overhead of the DTLS headers through compression [13]. As with the work on IPsec, we are currently not aware of any publication evaluating the performance of DTLS over 6LoWPAN. Our work can thus support these efforts by providing a first set of real-world measurements from our DTLS implementation.

## III. The DTLS Protocol

All messages sent via DTLS are prepended with a 13 bytes long DTLS record header. This header specifies the content of the message, e.g. application/handshake data, the version of the protocol employed, as well as a 64-bit sequence number and the record length. The top two bytes of the sequence number are used to specify the epoch of the message which changes once new encryption parameters have been negotiated between client and server. Figure 1 shows the DTLS record header in white. The record header is either followed by the plaintext, if no security has been negotiated yet, or by the DTLS block cipher, marked in grey. If a blockcipher is used, the plaintext is prepended by a random Initialization Vector (IV), which has the size of the cipher block length. This protects against an attack on ciphersuites using the ciper-block chaining (CBC) mode of operation for their block ciphers [14]. The plaintext is followed by a Hash-based MAC (HMAC) which allows the receiver to detect if the DTLS record (including the white DTLS header, but not the striped message parts) has been altered. Finally, the message is padded to a multiple of the cipher block length. The area of the message shown in grey in Figure 1 is encrypted with the block cipher. Unlike TLS, DTLS does not allow for stream ciphers because they are sensitive to message loss and reordering. Instead, DTLS uses block ciphers in the CBC mode of operation.

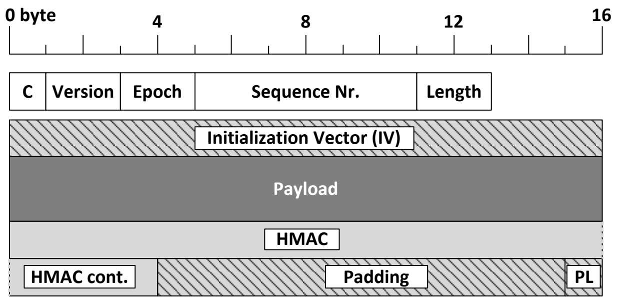

Fig. 1. A DTLS record protected with a CBC block cipher.

The key material and cipher suite, consisting of a block cipher and a hash algorithm, are negotiated between client and server during the handshake phase which commences before any application data can be transferred. There are three types of handshake: First, during an unauthenticated handshake neither party authenticates with the other. Second, during a server authenticated handshake only the server proves its identity to the client. Third, in a fully authenticated handshake the client has to authenticate itself to the server as well. In the following we will not consider the unauthenticated handshake because it provides no authenticity at all. There are different algorithms that can be used for authentication in a DTLS handshake. Variants based on ECC have been shown in embedded networks [7]. Since we argue for a standard-based communication architecture for the IoT to promote *interoperability*, the rest of the paper will focus on authentication based on RSA. Because it is today's dominant PKC system a suitable infrastructure for obtaining certificates from commercial Certificate Authorities (CA) is already in place. Furthermore, hardware support for secure application and storage of RSA keys exists with Trusted Platform Modules (TPMs). The drawback of using RSA is the large key size required for security which is currently 2048-bit [15].

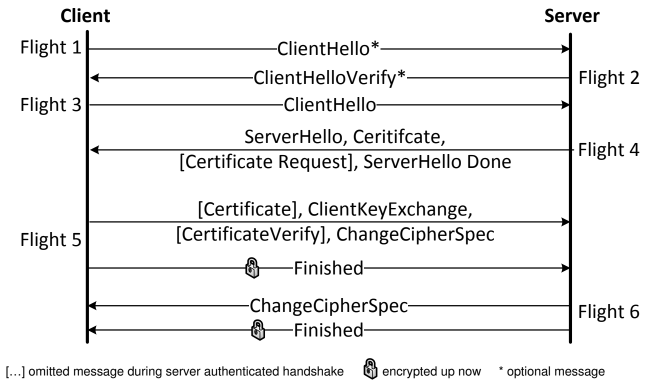
Client
Server
Flight 1
ClientHello*
ClientHelloVerify*
Flight 2
Flight 3
ClientHello
ServerHello, Ceritifcate,
[Certificate Request], ServerHello Done
Flight 4
[Certificate], ClientKeyExchange,
[CertificateVerify], ChangeCipherSpec
Flight 5
Finished
ChangeCipherSpec
Flight 6
Finished
[...] omitted message during server authenticated handshake
encrypted up now
* optional message

Fig. 2. A fully authenticated DTLS Handshake.

Figure 2 shows a fully authenticated DTLS handshake. Individual messages are grouped into "message flights" according to their direction and occurrence sequence. Flights 1 and 2 are an optional feature to protect the server against Denial-of-Service (DoS) attacks. The client has to prove that it can receive data as well as send data by resending its `ClientHello` message with the cookie sent by the server in the `ClientHelloVerify` message. The `ClientHello` message contains the protocol version supported by the client as well as the cipher suites that it supports. The server answers with its `ServerHello` message that contains the cipher suite chosen from the list offered by the client. The server also sends a X.509 certificate to authenticate itself followed by a `CertificateRequest` message if the server expects the client to authenticate. The `ServerHelloDone` message only indicates the end of flight 4. If requested and supported, the client sends its own certificate message at the beginning of flight 5. The `ClientKeyExchange` message contains half of the pre-master secret, which is encrypted with the server's public RSA key from the server's certificate. The other half of the pre-master secret was transmitted unprotected in the `ServerHello` message. The keying material is subsequently derived from this pre-master secret. Since half of the pre-master secret is encrypted with the server's public key it can only complete the handshake if it is in possession of the private key matching the public key in the server certificate. Accordingly, in the `CertificateVerify` message the client authenticates itself by proving that it is in possession of the private key matching the public key in the client's certificate. It does this by signing a hashed digest of all previous handshake messages with its private key. The server can verify this through the public key of the client. The `ChangeCipherSpec` message indicates that all following messages by the client will be encrypted with the negotiated cipher suite and keying material. The `Finished` message contains an encrypted message digest of all previous handshake messages to ensure both parties are indeed operating based on the same, unaltered, handshake data. The server answers with its own `ChangeCiperSpec` and `Finished` message to complete the handshake.

## IV. A Standard Based End-to-End Security Architecture

Our system architecture is following the IoT model. We assume that the Internet is connected by IPv6 in the near future, and parts of it run 6LoWPAN. The transport layer in 6LoWPAN is UDP which can be considered unreliable, the routing layer is RPL [16] or Hydro [4]. We currently use Hydro for routing, because it is fairly similar to RPL and it is readily available as part of the TinyOS 2.x distribution. IEEE 802.15.4 is used for the physical and MAC layer. Based on this protocol stack we chose DTLS as our security protocol. This places it in the application layer on top of the UDP transport layer (c.p. Figure 3).

Similar to security needs in traditional networks, such as the Internet, we consider three security goals: Authenticity, Integrity, and Confidentiality.

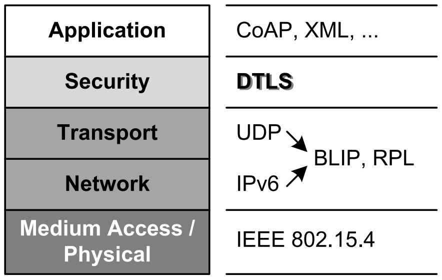

Fig. 3. Protocol stack used in our security architecture.

By choosing DTLS as the security protocol we can achieve these goals. DTLS is a modification of TLS for the unreliable UDP and inherits its security properties [17]. Using an application layer security protocol like DTLS, as opposed to link or network layer security protocols such as MiniSec [6], has a number of advantages but also some drawbacks. Lower layer security protocols do not provide end-to-end communication security. On each hop in a multi-hop network, data is decrypted on receipt and re-encrypted for forwarding. An attacker can thus gain access to all clear text data that passes through a compromised node. Scalability is often also an issue for these protocols, because they need to establish a secured connection with each of their neighbours to form a mesh network, and cryptographic overhead occurs on each hop.

However, an application layer security protocol does not protect routing information. Adversaries can therefore analyze the traffic patterns of a network in clear text. They may even launch a DoS, worm hole, or resource consumption attack that lowers the availability of the network [5]. In this paper, we focus on end-to-end communication security, and rely on other schemes for securing lower communication layers [5].

Scenarios dealing with subscriptions to sensitive data raise the need for proper authentication of data publishing devices and access control throughout the network. We therefore introduce an Access Control server (AC) into our architecture (c.p. Figure 4). The AC is a trusted entity and a more resource-rich server, on which the access rights for the publishers of the secured network are stored. This requires a unique identity for a publisher in the network. In the Internet, identities are usually established via PKC and the identifiers provided through X.509 certificates. A X.509 certificate contains, among other information, the public key of an entity and its common name (e.g. my-bank.com). The certificate is signed by a trusted third party, called the Certificate Authority (CA), which serves two purposes: Firstly, the signature allows the receiver to detect modifications to the certificate. Secondly, it also states that the CA has verified the identity of the entity that requested the certificate. A CA may either be run by the administrator of the network or one of the established internet certificate authorities could be used. Hu et al. showed that RSA, the most commonly used public key algorithm in the Internet, can be used in sensor networks with the assistance of a Trusted Platform Module (TPM) [8]. A TPM is an embedded chip that provides tamper proof generation and storage of RSA keys as well as hardware support for the RSA algorithm. The certificate of a TPM equipped publisher and the certificate of a trusted CA must be stored on the publisher prior to deployment. For publishers that are not equipped with TPM chips we propose authentication via the DTLS pre-shared key cipher-suite, which requires a small number of random bytes, from which the actual key is derived, to be preloaded to the publishers before deployment. This secret must also be made available to the AC server which will disclose the key to devices with sufficient authorization. Figure 4 provides an overview of the proposed architecture.

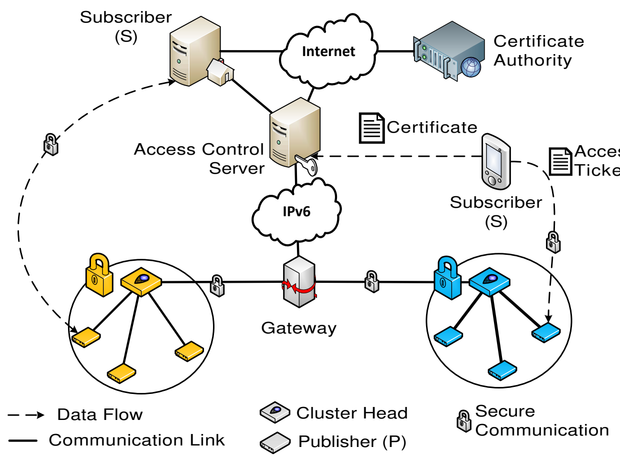

Fig. 4. The overview of our proposed system architecture.

### A. Establishing communication

Data publishers that wish to join the network first securely connect to a pre-configured subscriber. Depending on the capabilities of the node it may chose the appropriate DTLS cipher suite:

**TPM enabled publishers** can perform a fully authenticated handshake with the subscriber, which acts as the DTLS server in this handshake. Both the subscriber and the publisher transmit their RSA certificates in X.509 format. These certificates have been signed by a trusted CA with its private key, guaranteeing that the certificate has not been altered. As long as both parties keep their RSA private key secret, they can be sure of each other's identity. We assume traditional security best practices are applied for the subscriber to keep its private key secret. Publishers (i.e. sensor nodes) are regarded as vulnerable to physical tampering where it is assumed that an attacker can gain access to all information stored on the sensor nodes. It would be disastrous in the case of the RSA private key because it would allow the attacker to impersonate any compromised node. Therefore the RSA private key needs to be stored inside the tamper-proof TPM chip and never be passed outside. Consequently, all private key cryptographic operations must be performed within the TPM. Since TPMs are considered resistant to physical tampering the aforementioned scenario is difficult for an attacker and strong authentication during the handshake is guaranteed. The publisher checks the identity information of the subscriber's certificate against the preconfigured identity. The subscriber may optionally check with the AC server if the publisher is allowed to establish a connection, which is based on the identity information in the publisher's certificate. The top half of Figure 5 shows the establishment of a connection between the publisher and the subscriber.

**Constrained publishers** perform a variation of the TLS Pre-Shared Key (PSK) cipher suite. As discussed earlier, the publisher has a number of random bytes pre-installed before deployment. These are called protokeys and are used to derive the PSK for a session. The publisher sends its identity (e.g., IPv6 address) during the `ClientKeyExchange` message of the handshake, as described in Section III, instead of in a certificate. It also appends a few randomly generated bytes to its PSK identity to form a session identity. The publisher then derives the PSK by applying an HMAC function to the session identity with the protokey as key. The subscriber authenticates with the AC server and requests the PSK for the publisher's session identity. The AC server is the only fully trusted entity in the architecture, so the protokey can safely be stored there. The AC server generates the PSK for the subscriber from the session identity and the protokey and sends it to the subscriber via a secure connection. This scenario employs a chain of trust. The PSK still forms the basis for authentication as with a traditional TLS PSK cipher suite, but by using the protokey and session identity the publisher places less trust in the subscriber. Instead it forces the subscriber to authenticate itself with the AC server which can generate the PSK. Thus, the AC server validates the identity of the subscriber for the publisher. It also checks the identity of the publisher for the subscriber by generating a new PSK based on the protokey and corresponding (session) identity of the publisher. However, authenticity in this scenario is weaker than for a TPM enabled device because it relies on a third party, the AC server, to verify the identity and it is also possible for an attacker to impersonate a publisher if physical access can be obtained.

**Peer-to-Peer (P2P)** communication can only commence after a subscriber (S) is authenticated with the AC server. When S wishes to initiate communication with a publisher (P) in the network it must first obtain an access ticket from the AC server. S specifies to which peer it intends to connect and what type of connection (read, write, read/write) it wants to establish. The AC server checks the subscribers's access rights to the publisher and, if they are sufficient, sends a connection request to P. Now, P can decide whether or not it has sufficient resources to accept the new connection. If it has, P then initiates a DTLS handshake with S to establish a secured connection. This process is shown in Figure 5.

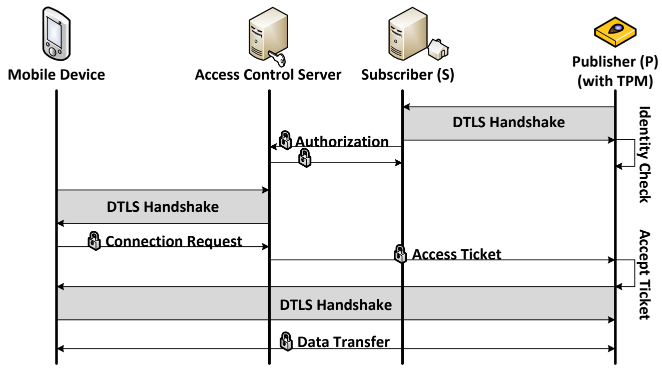

Fig. 5. Establishing a Peer-to-Peer connection.

## B. *Data Transmission*

Once a secure connection with a symmetric key has been established, the security layer is transparent to any applications on the publisher. However, any application first has to wait for the handshake to be completed before data may be exchanged, which does not happen frequently. It happens when a publisher node initially joins the network and afterwards during the re-keying process. Re-keying typically happens periodically (e.g., once per day), and may be performed either with an entirely new handshake or an abbreviated handshake [17] that reuses the previously exchanged secrets. The abbreviated version is significantly faster [7].

## V. Evaluation

Previous work has already demonstrated techniques to reduce the protocol header overhead during data transmission [11] and has proven the feasibility of performing software encryption and hashing on the sensor node [6], also called mote. Indeed, even for DTLS, first proposals for a compressed header format have been made by Raza et al. recently [13]. Gupta et al. showed the feasibility of a server authenticated SSL handshake [7]. Therefore, the component of our security architecture that is currently least understood in the context of the IoT is the fully authenticated DTLS handshake, which includes both client and server authentication. We have implemented a DTLS client that performs the DTLS handshake with an OpenSSL 1.0.0d server.

The client is targeted at the Opal sensor node [18] which features an Atmel SAM3U micro-controller and the Atmel AT97SC3203S TPM. It has 48 kB RAM and the micro-controller is clocked at 48 MHz in our implementation. In the following sections we will evaluate our implementation with regards to its performance during the handshake and data transmission, as well as its energy and memory consumption. Unless otherwise stated, the DTLS cipher suite performed was TLS-RSA-with-AES-128-CBC-SHA. AES-128 has been shown to be one of the fastest block ciphers on motes [19] and offers sufficent security. Furthermore, the cipher suite we chose is the required block cipher suite for DTLS from version 1.2 onwards. Other common cipher suites are either based on RC4, which is a stream cipher and thus not permitted by DTLS, or 3DES which is very slow and thus causes a large cryptographic overhead.

## A. *Latency measurements*

In this section we will consider latency as a measure of the system's performance. Figure 6 shows the round-trip time for different sizes of plaintext data through a single hop network and a multi hop network with four hops. We measured the timing for the DTLS protected packets on the mote. A packet sent with both a SHA-1 HMAC and AES-128 encryption is denoted as "AES-128". The denotation "SHA-1" is used if a packet only contained a SHA-1 HMAC. Readings for pure plaintext data without any additional headers were obtained by issuing the `ping6` command on the subscriber. The reading for 8 byte plaintext data is missing because the ICMP-Header and the timestamp sent by `ping6` are together at least 16 byte long. The chart shows a linear increase of round-trip time with jumps occurring approximately every 100 bytes. These spikes can be attributed to the 128 byte maximum layer 2 frame size defined by IEEE 802.15.4 which includes header and trailer. These jumps occur earlier when sending DTLS protected packets due to the additional DTLS packet headers, the HMAC size and the explicit Initialization Vector in each packet. See Section III for more details on the packet structure.

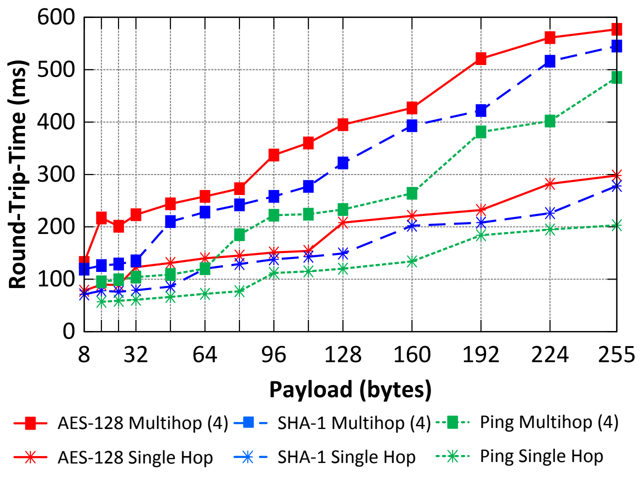

Fig. 6. Packet Round-trip time for different encryptions.

Both the increased packet size and processing overhead lead to an increased end-to-end transmission latency for DTLS packets compared to plaintext packets. In the single hop scenario, transmission latency was increased by up to 95 ms for AES-128 and up to 75 ms for SHA-1 encryption which were an average increase of 62% and 35% respectively over the plaintext case. In the multi hop scenario, round trip times increased by a maximum of 163 ms and were 74% longer on average for AES-128 encrypted packets. Packets with a SHA-1 HMAC took up to 129 ms longer for the round-trip with an average of 40% more time being spent. The decreased performance for transmission latency is mostly due to the large packet overhead of up to 64 bytes which consists of 13 byte DTLS record header, 16 byte Initialization Vector, 20 byte HMAC, and up to 15 byte padding. Calculating a SHA-1 hash of a 255 byte plaintext message only takes 9 ms, encryption with AES-128 takes another 12 ms. Both operations do not contribute significantly to the overall transmission latency.

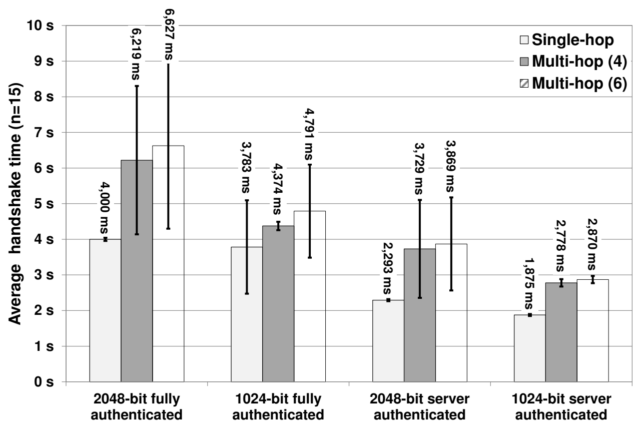

Fig. 7. Time to complete different types of DTLS handshakes.

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

The large standard deviation is caused by our implementation behavior when message loss occurs. DTLS states that an implementation should wait for an answer for a set amount of time after sending a flight of messages. If it does not receive an answer during this period it retransmits the whole flight. We set this timeout value to 5 seconds to avoid unnecessary retransmissions in networks with a high end-to-end delay, which is common in a low power lossy network, and/or with energy limited thin clients that are slow to respond. DTLS implementations for the Internet often choose a retransmission timeout of one second or less. In general, we see that the time to execute a handshake is shorter for smaller RSA-keys and reduced by almost two seconds when client authentication is omitted in the handshake. We observed packet loss mainly in a multi-hop environment and when larger DTLS messages were being sent. This increases the total handshake time significantly because of the large DTLS retransmission timeout. However, total energy consumption of the client does not increase significantly because all TPM operations, which are the largest contributor to overall handshake energy costs, are only executed after successful receipt of all relevant server messages. Losing a packet with information obtained from the TPM does not lead to a repeated execution of the TPM operations because the resulting messages are buffered and can be retransmitted. During our experiments we did not see any failed handshake attempts. In earlier stages of development a lost `Finished` message from the server would cause the handshake to fail. The client did not receive the expected Finished message and kept retransmitting its last message flight. The server, however, already considered the handshake to be complete and was waiting for bulk data transfer from the client, disregarding its repeated retransmissions of the handshake messages. DTLS 1.2 addresses this issue by always issuing a retransmission of the server's last message flight when it receives a finished message from the client. We ported this behavior to our version of OpenSSL to address this problem.

## B. Energy consumption

We measured the energy consumption during the handshake phase across a $10\Omega$ resistor with an oscilloscope. This yielded a value for the electric potential which can be converted into a value for the current draw by dividing it through the value of the resistance ($10\Omega$). The energy costs can then be calculated as: $\frac{U_{probe}}{R} \times t \times U_{battery}$. Where $U_{probe}$ is the measured voltage, $R = 10\Omega$ is the value of the resistor, $t$ is the transaction time, and $U_{battery} = 3.998V$ is the battery voltage. Table I shows the energy consumption during a typical execution of different handshake types. We use a 2048-bit RSA key because 1024-bit key is not recommended for future deployments [8]. Values for current draw in Table I specify the amount that each component contributes to the total current draw. Figure 8 shows a capture from the oscilloscope for a 2048-bit RSA fully authenticated handshake.

| | Current | fully authenticated handshake | server authenticated handshake |
|---|---|---|---|
| Computation | 11.4 mA | 35 ms, 1.59 mJ | 33 ms, 1.50 mJ |
| Radio TX | 18 mA | 242 ms, 17.4 mJ | 70 ms, 5.03 mJ |
| TPM Start | 52.2 mA | 836 ms, 174.46 mJ | 836 ms, 174.5 mJ |
| TPM TWI | 43.6 mA | 688 ms, 120.0 mJ | 476 ms, 83.0 mJ |
| TPM Verify | 51.8 mA | 59 ms, 12.2 mJ | 56 ms, 11.6 mJ |
| TPM Encrypt | 51.8 mA | 39 ms, 8.07 mJ | 40 ms, 8.28 mJ |
| TPM Sign | 52.2 mA | 726 ms, 151.5 mJ | - |
| Total minimum | | 485.2 mJ | 283.9 mJ |
| CPU idle | 11.4 mA | 3965 ms, 180.7 mJ | 2265 ms, 103.2 mJ |
| Radio idle | 18 mA | 3758 ms, 270.4 mJ | 2228 ms, 160.3 mJ |
| Total | | 936.4 mJ | 547.4 mJ |

TABLE I
CURRENT CONSUMPTION, TRANSACTION TIME, AND ENERGY CONSUMPTION OF DTLS HANDSHAKE (2048-BIT KEY).

We chose to neglect the contribution of the radio and micro-controller in further discussion, which have been marked as "CPU idle" and "Radio idle" in Table I. Both can be considerably reduced by using power saving techniques, e.g. by using the TinyOS Low Power Listening (LPL) Media Access Control layer for the radio (less than 1% radio duty cycles have been reported by the literature repeatedly), and setting the micro-controller into a lower power state where it consumes less than 30 $\mu$W for SAM3U when it is idle [20]. Sending messages ("Radio TX") and performing cryptographic operations ("Computation") contribute very little to the overall energy cost. It is largely bound by the energy usage of the TPM.

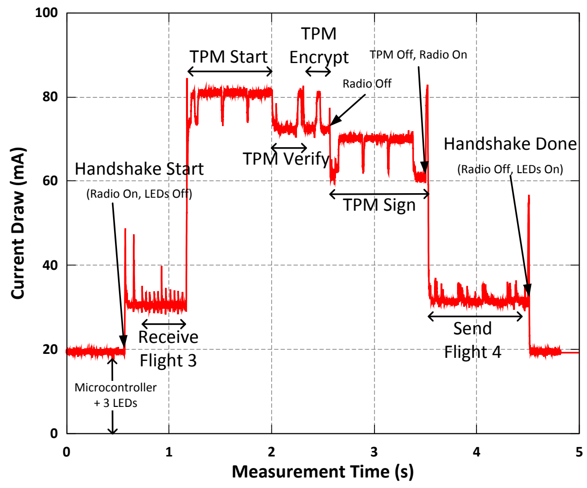

Fig. 8. Current draw for a fully authenticated DTLS handshake.

As can be seen in Figure 8, "TPM Start" and "TPM Sign" are the longest consecutive operations. The TPM is performing an operation with its RSA private key in "TPM Sign" which is more complex than that with a RSA public-key. During the "TPM Start" phase the TPM performs a series of internal self tests to detect tampering and unauthorized commands. The second large block is "TPM TWI" which describes the amount of time that is spent passing data to the TPM and receiving data from it via the TWI bus clocked at 100 KHz. It shows as a lower current draw in Figure 8. It can be seen directly after the end of the "TPM Start" sequence and before the short spike in "TPM Verify". The spike is the actual verification operation performed by the TPM. Similarly, the actual "TPM Encrypt" operation is the spike that follows another section of data transfer on the TWI bus. During "TPM Verify" the TPM uses the stored key of a CA to verify the server certificate presented during the handshake. The "TPM Encrypt" operation is used to encrypt a nonce with the server's public key. If the mote is expected to authenticate itself during the handshake, it performs a "TPM Sign" operation to sign a hash over all previous handshake messages with it's RSA private key. Since a server authenticated handshake does not require the expensive "TPM Sign" operation it uses significantly less energy but also provides weaker overall authentication since an attacker could impersonate a mote towards the server. Communication time is also shorter since the sensor node does not send its certificate.

If the mote is powered by two AA 2,800-mAh batteries, they have an energy of approximately 30,240 Joule. If 5% of the energy is used for DTLS handshakes for (re)keying purposes, which happen once per day, it could last for more than 8.5 years for a fully authenticated handshake at 485.2 mJ each, or more than 14.5 years for a server authenticated handshake at 283.9 mJ each.

The energy consumption after the completion of the handshake is closely related to the latency values from Figure 7 which portrait the influence of the network and processing overhead introduced by DTLS. As stated earlier, the calculation of a SHA-1 Hash for 255 Bytes takes 9 ms and encryption with AES-128 another 12 ms. Given the current draw for computation of 11.4 $mA$ from Table I this results in the order of 3.8 $\mu J$ per Byte.

### C. Memory

In order to determine the memory allocation to individual components of our implementation we analyzed the entries in the symbols table of the Opal binary after compilation. Memory has been measured for a fully authenticated handshake with 2048-bit RSA keys. This type of handshakes has the largest memory requirements since it needs more code and buffer space for the client's `Certificate` and `CertificateVerify` messages. We divide the memory consumption into six respectively seven categories as illustrated in Table II. In total approximately 18 kB of RAM and 63 kB of ROM is required for the implementation. Whereas the BLIP implemention requires most of the resources, followed by TPM drivers and DTLS networking code. But the implementation is still below the 48 kB of RAM / 256 kB of program memory provided by OPAL.

| | RAM (bytes) | ROM (bytes) |
|---|---|---|
| Cryptography | 541 | 10,838 |
| DTLS Messages | 1,174 | 2,568 |
| DTLS Network | 4,294 | 5,672 |
| TPM | 4,321 | 4,928 |
| BLIP | 6,352 | 9,298 |
| Application | 166 | - |
| System | 991 | 30,075 |
| **Total** | **17,839** | **63,379** |

TABLE II
RAM AND ROM USAGE OF DIFFERENT COMPONENTS

## VI. Conclusion

We have introduced a standard based security architecture with two way authentication for the IoT. The authentication is performed during a fully authenticated DTLS handshake and based on an exchange of X.509 certificates containing RSA keys which we have implemented. Our extensive evaluation based on real IoT systems show that our proposed architecture provides message integrity, confidentiality and authenticity with affordable energy, end-to-end latency and memory overhead which make it a feasible security solution for the emerging IoT. Previous work has demonstrated techniques to minimize packet headers for similar protocols [11]. We plan to apply these techniques to DTLS in future work. Another focus will be the inclusion of more constrained nodes without a TPM in our architecture, for which we plan to use a variant of the DTLS pre-shared key cipher suites.

## References

[1] I. Leontiadis, C. Efstratiou, C. Mascolo, and J. Crowcroft, "SenShare: Transforming Sensor Networks into Multi-application Sensing Infrastructures," in *Wireless Sensor Networks*, ser. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, Jun. 2012, vol. 7158, pp. 65–81.

[2] ETSI TR 102681, "Machine-to-Machine Communications (M2M); Smart Metering Use Cases," http://www.etsi.org, May 2010.

[3] Z. Shelby, K. Hartke, C. Bormann, and B. Frank, "Constrained Application Protocol (CoAP)," Internet Engineering Task Force (IETF), Jun. 2012.

[4] S. Dawson-Haggerty, A. Tavakoli, and D. Culler, "Hydro: A Hybrid Routing Protocol for Low-Power and Lossy Networks," in *Proceedings of the 1st IEEE International Conference on Smart Grid Communications*, Oct. 2010, pp. 268–273.

[5] D. R. Raymond and S. F. Midkiff, "Denial-of-Service in Wireless Sensor Networks: Attacks and Defenses," *Pervasive Computing*, vol. 7, no. 1, pp. 74–81, Jan.-Mar. 2008.

[6] M. Luk, G. Mezzour, A. Perrig, and V. Gligor, "MiniSec: A Secure Sensor Network Communication Architecture," in *Proceedings of the 6th International Conference on Information Processing in Sensor Networks*, Apr. 2007, pp. 479–488.

[7] V. Gupta, M. Wurm, Y. Zhu, M. Millard, S. Fung, N. Gura, H. Eberle, and S. C. Shantz, "Sizzle: A Standards-Based End-to-End Security Architecture for the Embedded Internet," *Pervasive Mobile Computing*, vol. 1, pp. 425–445, Dec. 2005.

[8] W. Hu, H. Tan, P. Corke, W. C. Shih, and S. Jha, "Toward Trusted Wireless Sensor Networks," *ACM Transanctions on Sensor Networks*, vol. 7, Aug. 2010.

[9] H. Chan, A. Perrig, and D. Song, "Random Key Predistribution Schemes for Sensor Networks," in *Proceedings of the IEEE Symposium on Security and Privacy*, 2003, pp. 197–213.

[10] W. Jung, S. Hong, M. Ha, Y.-J. Kim, and D. Kim, "SSL-Based Lightweight Security of IP-Based Wireless Sensor Networks," in *Proceedings of the International Conference on Advanced Information Networking and Applications Workshops*, May 2009, pp. 1112–1117.

[11] S. Raza, T. Voigt, and U. Roedig, "6LoWPAN Extension for IPsec," in *Proceedings of the Interconnecting Smart Objects with the Internet Workshop*, 2011.

[12] S. Raza, T. Voigt, and V. Jutvik, "Lightweight IKEv2: A Key Management Solution for both the Compressed IPsec and the IEEE 802.15.4 Security," in *Proceedings of the IETF Workshop on Smart Object Security*, 2012.

[13] S. Raza, D. Trabalza, and T. Voigt, "6LoWPAN Compressed DTLS for CoAP," in *Proceedings of the 8th IEEE International Conference on Distributed Computing in Sensor Systems*, May 2012.

[14] B. Moeller, "Security of CBC Ciphersuites in SSL/TLS: Problems and Countermeasures," http://www.openssl.org/~bodo/tls-cbc.txt, May 2004.

[15] E. Barker, W. Barker, W. Burr, W. Polk, and M. Smid, "NIST SP800-57: Recommendation for Key Management - Part 1: General(Revised)," NIST, Tech. Rep., Mar. 2007.

[16] T. Winter, P. Thubert, A. Brandt, T. Clausen, J. Hui, R. Kelsey, P. Levis, K. Pister, R. Struik, and J. Vasseur, "RPL: IPv6 Routing Protocol for Low Power and Lossy Networks," Internet Engineering Task Force (IETF), Mar. 2011.

[17] N. Modadugu and E. Rescorla, "The Design and Implementation of Datagram TLS," in *Proceedings of the Network and Distributed System Security Symposium*, 2004.

[18] R. Jurdak, K. Klues, B. Kusy, C. Richter, K. Langendoen, and M. Brünig, "Opal: A Multiradio Platform for High Throughput Wireless Sensor Networks," *Embedded Systems Letters, IEEE*, vol. 3, no. 4, pp. 121 – 124, Dec. 2011.

[19] J. Großschädl, S. Tillich, C. Rechberger, M. Hofmann, and M. Medwed, "Energy Evaluation of Software Implementations of Block Ciphers under Memory Constraints," in *Proceedings of the Conference on Design, Automation and Test in Europe*, Apr. 2007, pp. 1110–1115.

[20] J. L. Hill and D. E. Culler, "Mica: A Wireless Platform for Deeply Embedded Networks," *Micro, IEEE*, vol. 22, no. 6, pp. 12–24, Nov./Dec. 2002.