Securing the Internet of Things: A Standardization Perspective

Sye Loong Keoh, Sandeep S. Kumar, and Hannes Tschofenig

Abstract—The Internet-of-Things (IoT) is the next wave of innovation that promises to improve and optimize our daily life based on intelligent sensors and smart objects working together. Through IP connectivity, devices can now be connected to the Internet, thus allowing them to be read, controlled and managed at any time and any place. Security is an important aspect for IoT deployments. However, proprietary security solutions do not help in formulating a coherent security vision to enable IoT devices to securely communicate with each other in an interoperable manner. This paper gives an overview of the efforts in the Internet Engineering Task Force (IETF) to standardize security solutions for the IoT ecosystem. We first provide an in-depth review of the communication security solutions for IoT, specifically the standard security protocols to be used in conjunction with the Constrained Application Protocol (CoAP), an application protocol specifically tailored to the needs of adapting to the constraints of IoT devices. Since Datagram Transport Layer Security (DTLS) has been chosen as the channel security underneath CoAP, this paper also discusses the latest standardization efforts to adapt and enhance the DTLS for IoT applications. This includes the use of (i) raw public key in DTLS, (ii) extending DTLS Record Layer to protect group (multicast) communication, and (iii) profiling of DTLS for reducing the size and complexity of implementations on embedded devices. We also provide an extensive review of compression schemes that are being proposed in IETF to mitigate message fragmentation issues in DTLS.

Index Terms—Internet of Things, Communication Security, Standardization, Machine-to-Machine Communication, Compression Scheme, End-to-End Security

1 Introduction

The notion of Internet of Things (IoT) has been recognized by industrial leaders and media as the next wave of innovation, and pervading into our daily life [9], [12]. Sensors around us are increasingly becoming more pervasive and attempt to fulfill end users' needs, thus providing ease of usability in our everyday activities. Devices deployed in households, industrial automation, and smart city infrastructure are now interconnected with the Internet. This interconnection provides a whole range of data (environmental context, device status, energy usage, etc) that can be collected, aggregated and then shared in an efficient, secure, and privacy-aware manner. As these devices are connected to the Internet, they can be reached, and managed at anytime and any place.

The current landscape of IoT is filled with a very diverse range of wireless communication technologies, such as IEEE 802.15.4 [1], Wifi, Bluetooth Low Energy (BLE) [34], and various other cellular communication technologies. Quite naturally, devices using different physical and link layers are not interoperable with each other. Through an IP router, these devices are, however, able to communicate with the Internet. When the differences in the protocol stack extend beyond the physical and link layer, protocol translation needs to be performed by a gateway device. This harms the deployment of IoT devices because the deployment becomes more complex and expensive with multiple middleboxes along the end-to-end communication path. In order to ensure seamless connectivity between different devices deployed in the market, a convergence towards an all IP-based communication stack is necessary.

Already years ago the Internet Engineering Task Force (IETF) has standardized IPv6 over Low-Power Wireless Personal Area Networks (6LoWPAN) [32], Routing Over Low-power and Lossy-networks (ROLL) [52] and Constrained Application Protocol (CoAP) [51], to equip constrained devices with low memory footprint and computational capabilities to run IPv6 over low-power wireless networks. The ZigBee IP standard [5], which primarily targets the smart energy domain, builds on top of the 6LoWPAN stack [37], [20], [50]. IEEE 802.15.4-based devices used in other industry domains are expected to adopt the 6LoWPAN concept as well since it provides the basis for running IPv6 over low power radios via an adaptation layer, profiling of the IPv6 neighbor discovery mechanism, and compression schemes. Similar adaptations are provided to Bluetooth low energy [39] and DECT Ultra Low Energy [41], two other short-range radio technologies. Meanwhile many IoT devices are using WiFi and are already running the full IP protocol stack. IP protocol can be regarded as the glue to interconnect these heterogeneous wireless

- *S.L. Keoh is with the School of Computing Science, University of Glasgow Singapore.*
  E-mail: SyeLoong.Keoh@glasow.ac.uk
- *S.S. Kumar is with Philips Research Europe, Eindhoven, The Netherlands.*
  E-mail: sandeep.kumar@philips.com
- *H. Tschofenig is with ARM Limited, Cambridge, United Kingdom.*
  E-mail: Hannes.Tschofenig@gmx.net

networks together.

The pervasive device connectivity to the Internet also poses hidden security risks, namely eavesdropping on the wireless communication channel, unauthorized access to devices, tampering with devices, and privacy risks. The inherent nature of constrained devices means that the state-of-the-art cryptographic algorithms and protocols are not easy to deploy on such devices and even more difficult to keep the software up-to-date. The ability to connect, manage and control a device from anywhere and at anytime requires appropriate authentication and authorization measures. Security experts have emphasized the importance of security in IoT deployments and have warned about the insecurity of current deployments [49], [48].

Security for IoT encompasses a wide range of tasks, including embedding keying material during the manufacturing process of the device, provisioning of new keying material during operation, establishing access control policies for access to networks and services, processes for secure software development, use of hardware security modules to protect keys against tampering, software update management, development and selection of efficient cryptographic primitives, etc. Custom security solutions offered by the IoT research community offer mostly point solutions, but this rarely helps to understand the big picture for securing IoT devices.

This papers provides a state-of-the-art snapshot of the standardization efforts for securing IoT in the IETF. We believe that these standardization activities play a crucial role in securing the IoT eco-system, both in terms of improving interoperability of IoT devices in general and to pave the road towards wider industry adoption of security solutions.

The rest of the paper is organized as follows: Section 2 motivates the importance of security standards. Section 3 reviews the various security standardization activities in IETF, and highlights the significance of re-using existing Internet security protocols. Section 4 presents some performance evaluation results and analysis. Section 5 discusses challenges ahead. Finally, Section 6 concludes the paper.

2 Inter-operable Security for IoT

The success of the World Wide Web benefited from a solid foundation built on a standardized protocol stack consisting of the Internet Protocol (IP), the Transmission Control Protocol (TCP), the Transport Layer Security (TLS) protocol, the Hypertext Transfer Protocol (HTTP), and the Hypertext Markup Language (HTML). When Personal Digital Assistants (PDAs) were introduced they were not able to run the full Web stack. Therefore, the Wireless Application Protocol (WAP) [3] was developed to allow interworking with the Web infrastructure. The deployment of WAP was, however, a disappointment overall as it never got anywhere close to the success of the plain HTTP/HTML. Proxying between the different protocols lead to slower innovation since new features deployed on the Web were only available to mobile devices once the gateways were updated. Once mobile devices were capable of supporting the full Web stack, the limited deployment of WAP quickly vanished.

In terms of security, the Wireless Transport Layer Security (WTLS) [4] protocol was standardized to provide communication security for WAP as a TLS counterpart. However, it did not mandate the use of cryptographic and key generation algorithms, thus leading to many insecure algorithms such as 12-round RC5 being implemented and deployed. Although a standardized security protocol was used, the fact that WTLS did not ensure end-to-end security is a problem since the WAP gateway was essentially a man-in-the-middle that had access to the data being transmitted over the Internet. On one hand, it is important that standardization ensures interoperability. On the other hand, it is crucial that the correct (adaptation of) security protocol and security algorithms are also standardized to counter the the Internet threat model [44] and its changes over time [15].

There are many standards for Internet security protocols developed in different standardization bodies, such as IEEE (link layer), IETF (network, transport, and application layer), and W3C (web application layer). These different security protocols offer different protection at different layers and complement each other in fulfilling different security goals. The use of these standardized security protocols is at the discretion of the system architects, who are required to analyze the threats and to decide on how to mitigate them. The security considerations sections found in IETF specifications provide helpful guidance for system architects to make a good judgment. For example, IPsec [25] is not mandatory to be used at the network layer, and TLS is only enabled for applications requiring channel security with authentication, integrity and confidentiality. The OAuth 2.0 protocol [17] is only relevant for applications that require delegated authorization to protected resources. This flexibility has been desired because traditionally Internet devices accomplish interoperability by implementing several of these protocols which can typically be updated fairly easily. Therefore, interoperability is often by devices implementing a wide range of security protocols and cryptographic algorithms.

However, it is infeasible to require resource constrained IoT devices to implement all security protocols at all layers. Consequently, it is important to ascertain the threats and risks posed in IoT, and subsequently determine the security protection required that should be deployed across the layers. By mandating such a security protocol implementation for IoT devices, some level of security interoperability can be guaranteed. While one-size fits all may not serve all IoT use cases, security profiles (i.e., a subset of security protocol functionalities and options) can be specified to address the requirements of different IoT applications.

Ideally, a single security protocol suite that provides a

full security suite, namely authentication, authorization, integrity and confidentiality protection, should be standardized. In fact, various such security protocols have already been standardized and adapting them to cater for the required security functionalities for use in IoT would be beneficial. In terms of security, most of the standardized protocols have been through a thorough security analysis. Furthermore, such a standardized protocol when deployed on IoT devices, they can interoperate more easily with existing Internet infrastructure and services. Conversely, designing a completely new security protocol for IoT seems like re-inventing the wheel, and it might be difficult to get market traction whereby a critical mass of devices needs to be achieved in order to have an interoperable IoT.

3 Status Quo of IoT Security Standardization in IETF

This section discusses the current IoT security standardization efforts in IETF. We first introduce the Constrained Application Protocol (CoAP) [51], followed by standardization efforts to adapt the current communication security for use with CoAP. It is noteworthy that a significant amount of effort has been dedicated to optimize DTLS in order to provide transport layer security for CoAP in a style similar to HTTPS. In addition, a standard way of granting permissions and authorizing IoT devices to access each other's resources is being investigated in IETF, by tapping on the experience obtained with the development of OAuth 2.0 [17].

3.1 Constrained Application Protocol (CoAP)

The Constrained RESTful Environments (CoRE) Working Group [21] within the IETF focuses on developing a resource-oriented application framework for constrained IP networks. Resource-oriented means that an application model is offered in which, similar to HTTP on the Internet, data in the form of resources can be stored, retrieved and manipulated via a client-server protocol. The main result of the working group is the development of CoAP [51]. In CoAP, as with HTTP, the Universal Resource Identifier (URI) is used to access the resources on a given host. CoAP is a relatively simple request and response protocol providing both reliable and unreliable forms of communication. For the CoAP protocol, the "coap" URI scheme will be used. A CoAP-enabled device may be acting in a client role, a server role, or both, or sending non-confirmable messages without response.

The reasons that a new protocol is defined for constrained IP networks, instead of simply re-using HTTP is to greatly reduce overhead in implementation complexity (code size) and to reduce the bandwidth requirements. Such data reduction also helps to increase reliability (by reducing link layer fragmentation) and reduce latency in typical low-power lossy wireless networks, such as IEEE 802.15.4 or BTLE.

The CoAP protocol runs on top of UDP. Contrary to HTTP-over-TCP, which supports only unicast, CoAP-over-UDP offers both unicast and multicast (i.e., group communication).

3.2 Communication Security

Since HTTP is protected using TLS [11], it is thus natural to use DTLS [45] to protect CoAP. In this way, end-to-end communication security can be guaranteed between two communicating devices in an IoT environment. The handshake phase is used for authentication and for establishing keying material for channel security.

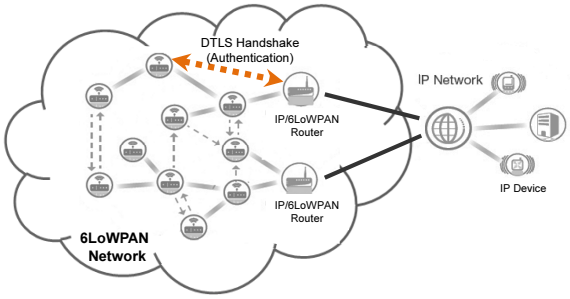

Fig. 1. Using DTLS for Network Access Authentication.

3.2.1 Datagram Transport Layer Security (DTLS)

DTLS is arguably the most suited single security protocol for providing channel security [16], [26], mainly because it is a rather complete security protocol that can perform authentication, key exchange and protecting application data with the negotiated keying material and algorithms.

It is assumed that CoAP-based IoT devices are provided with the necessary long-term keying material during device manufacturing or dynamically during the lifetime of the device via configuration. Based on the configuration, an IoT device will be in one of the four security modes:

- **NoSec**: There is no protocol level security and DTLS is disabled. However, it is recommended that the IP network layer security is provided, e.g., using IPsec. In this mode, the CoAP device simply sends the packets over normal UDP over IP without any transport layer security protection.
- **PreSharedKey**: DTLS is enabled and Pre-Shared Key (PSK)-based authentication is used. The device will be provisioned with a list of keys and each key includes a list of nodes for which this key can be used. In the best security scenario, the CoAP device shares a unique key with each communication partner.
- **RawPublicKey**: DTLS is enabled and the device has an asymmetric key pair, but the public key is not embedded within an X.509 certificate. The device also has a list of nodes it can communicate with.
- **Certificate**: DTLS is enabled and the device has an asymmetric key pair. The X.509 certificate binds

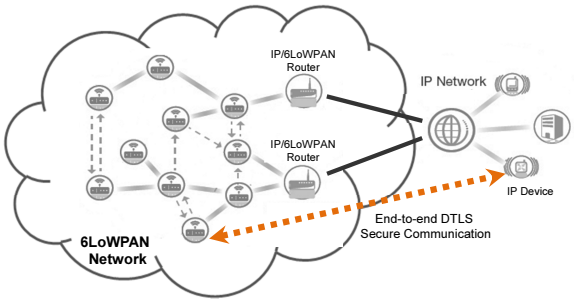

Fig. 2. Using DTLS for End-to-End Channel Security.

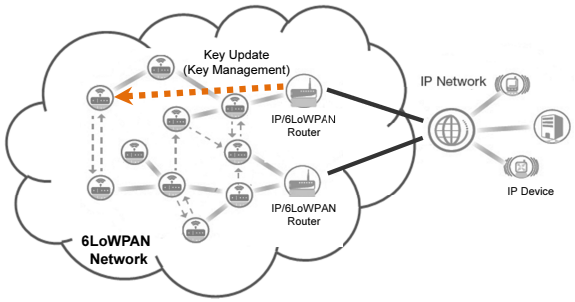

Fig. 3. Using DTLS to facilitate key update and key management

the public key to an identifier and is signed by a certification authority. The device also has a list of trust anchors so that path validation of certificates received from other entities can be performed.

By using DTLS as the sole security suite for IoT, the following security protection can be achieved:

Network Access – A 6LoWPAN network is typically protected using a link-layer MAC (L2) key, so that only authorized devices that possess this key can communicate within the network, and data packets that cannot be authenticated using the L2 key will be dropped at the first hop. The 6LoWPAN Border Router (6LBR), a device similar to a network access server in regular WiFi deployments, is ideally responsible for authenticating and authorizing devices prior to authorizing them to join the network.

As shown in Figure 1, DTLS as an authentication protocol can be used to authenticate new devices joining the network either by using the Pre-Shared Key (PSK) mode, raw public-key, or public-key certificate. The result of a successful DTLS handshake creates a secure channel between the new device and the authorizing entity (for example the 6LBR). This secure channel enables the authorizing entity to distribute the L2 key securely to the joining device based on rules which have been configured by the network owner.

If the new device and the authorizing entity are one-hop at the MAC layer, then the DTLS handshake messages are not dropped by the MAC layer. However if new device and the authorizing entity are multi-hop at the MAC layer, the DTLS messages are dropped at the MAC layer since they are not yet protected with the L2 key, leading to a chicken-and-egg situation. Therefore techniques to enable forwarding of these multi-hop network access DTLS handshake messages without being dropped at the MAC layer are required, for example, using the DTLS relaying [30].

Note, however, that network access authentication for Ethernet and Wifi uses the Extensible Authentication Protocol (EAP) with an EAP method encapsulated inside, which performs the actual authentication and key exchange protocol run. Various EAP methods have been standardized in the IETF. Re-using a TLS-based EAP method for network access authentication ensures that code can be re-used by the constraint device. The direct use of DTLS for network access, as proposed in [30], is currently at an early discussion stage.

Secure Communication Channel – Although communication within a multi-hop 6LoWPAN network may be protected hop-by-hop at the link layer it does not provide end-to-end security required by many applications. For example, devices in the 6LoWPAN may interact with Internet services or devices located at different networks. This requires an end-to-end security solution so that application messages that leave the 6LoWPAN through the 6LBR continue to experience communication security.

As shown in Figure 2, a DTLS end-to-end session can be established between two communicating devices, one inside the 6LoWPAN and the other outside, to securely transport application data (CoAP messages). The application data is protected by the DTLS Record Layer, i.e., authenticated and encrypted with a fresh, and unique session key.

Key Management – As DTLS has the capability of renewing session keys, this mechanism can be utilized to support key management in a 6LoWPAN network. During the *Network Access* phase, the 6LBR distributes a L2 key during the network access authentication procedure. It is thus possible to re-use the same channel to facilitate key management, enabling the L2 key to be updated by the 6LBR when necessary. Figure 3 shows the use of DTLS to facilitate key management by using individual DTLS sessions with each device in the network. This key distribution capabilities can be re-used in the design of the group key management solution.

3.2.2 IPsec

IPsec provides channel security at the IP layer, making it possible to ensure end-to-end security between pairs of communicating devices. As IPSec operates at the network layer, it has the advantage of protecting all higher-layer protocols. Therefore, many researchers [42], [47],

[46] consider IPsec a desirable security solution for IoT. The Authentication Header (AH) and Encapsulating Security Payload (ESP) are responsible for providing security services for protecting data traffic. The AH protocol provides integrity and data origin authentication for IP datagrams and protects against replay. The ESP protocol provides authenticity, integrity and confidentiality to the IP packets. In order for AH or ESP to function, it requires a Security Association (SA), i.e., keying material and various other parameters. These SAs can be established dynamically using the Internet Key Exchange (IKEv2) protocol.

An IPsec AH and ESP implementation [42] is available for Contiki OS. For AH, HMAC-SHA1-96 and AES-XCBC-MAC-96 have been implemented. For ESP, the AES-CBC is used for encryption and HMAC-SHA1-96 is used for authentication.

3.2.3 Minimal Internet Key Exchange (IKEv2)

For a dynamic establishment of IPsec Security Associations, IKEv2 is used and [29] proposes a minimal IKEv2 for use with resource constrained devices. Several optional features of IKEv2, such as NAT traversal, IKE SA rekey, child SA rekey, multiple child SAs, deleting child/IKE SAs, configuration payloads, EAP authentication and cookies are omitted from the profile resulting in a more lightweight implementation.

The minimal IKEv2 only uses the first two message exchanges called IKE_SA_INIT and IKE_AUTH to create IKE SA and the first child SA. An IoT device, which supports minimal IKEv2, can initiate the message exchange, but will not able to respond to any other requests. It is most likely that the minimal IKEv2 only supports exactly one set of cryptographic algorithms, and authentication is based on shared secrets. Authentication based on X.509 public-key certificates is not supported in the minimal IKEv2 specification.

3.2.4 Host Identity Protocol (HIP)

HIP [14] introduces a shim layer between the IP and transport layer in the form of a cryptographic namespace called host identities (HIs). The *HIP Base EXchange (BEX)* uses the cryptographic HIs in a mutual authenticated Diffie-Hellman (DH) key exchange to establish a symmetric secret between the Initiator and Responder. It relies heavily on public-key cryptography.

The *HIP Diet EXchange (DEX)* [38] defines a lightweight alternative to the BEX that aims to remove the more expensive cryptographic primitives, such as signatures, hash functions, or the use of the ephemeral DH. HIP-DEX thus sacrifices some security properties, such as perfect forward secrecy and the choice of crypto suites. With lower bandwidth requirements, it can deal with higher packet loss due to an aggressive retransmission scheme. HIP-DEX still provides DoS protection by means of a puzzle mechanism and also allows for password-based authentication.

3.2.5 Fragmentation

All the communication security protocols explicitly require the communicating parties to exchange identifiers, random numbers, keying materials, or even certificates in order to establish a secure communication session. With an IEEE 802.15.4 radio, for example, the MAC datagram packet size is only 127 bytes. The MAC frame header size alone consumes 25 bytes with no security and up to 46 bytes with AES-CCM-128 security. Thus, only 102 bytes (or with the best security enabled only 81 bytes) are left for an IP packet. This is exacerbated by the need to provision another 48 bytes for the IPv6 and UDP headers, leaving only approximately 64 bytes for application data and the necessary security protections. It is inevitable that certain messages in DTLS, IPsec, and IKEv2 would not fit into a single packet and must be fragmented into multiple packets for delivery. Fragmentation of packets causes problems because fragments may be lost, arrive out-of-order, or they need to be retransmitted, and at the receiving end these fragments must be reassembled.

In DTLS, when the $ClientHello$ message encapsulates the full size *Random* field and *Cookies* in the protocol, it would not fit into an IEEE 802.15.4 packet and has to be fragmented into two fragments. In IPsec, when AH and ESP are used, they would consume 54 bytes of the packet. In fact, it is not possible to fit them into one packet, and similarly causing fragmentation. One approach to solve this problem is to avoid fragmentation altogether by employing compression techniques to reduce the message size [18], [43], [42]. There is a standard 6LowPAN header compression [20] that defines the encoding format, LOWPAN_IPHC, to compress Unique Local, Global, and multicast IPv6 Addresses based on shared state within contexts, as well as LOWPAN_NHC for encoding of the next header compression. These header compression scheme effectively removes header fields that are implicitly known to all nodes in the 6LoWPAN network. The idea is to apply these encoding techniques and derive a general header compression scheme that can be used to compress DTLS headers and IPsec headers as well.

DTLS Header Compression [43], [18] – The Record Layer header adds 13 bytes to every application message that is transmitted, while the Handshake header adds 12 bytes to each handshake message. The 6LoWPAN_NHC for DTLS can reduce the Record and Handshake headers to 5 and 3 bytes respectively [43]. This scheme only works on fresh DTLS handshakes, as successive re-handshakes encrypt the handshake header using the existing negotiated ciphersuites.

Figure 4 shows the 6LoWPAN_NHC encodings for various DTLS headers. Figure 4(a) denotes the encodings for the Record and Handshake headers (LoWPAN_NHC_RHS), and encodings for the Record header only (LoWPAN_NHC_R). The DTLS *version* (V), *Epoch*

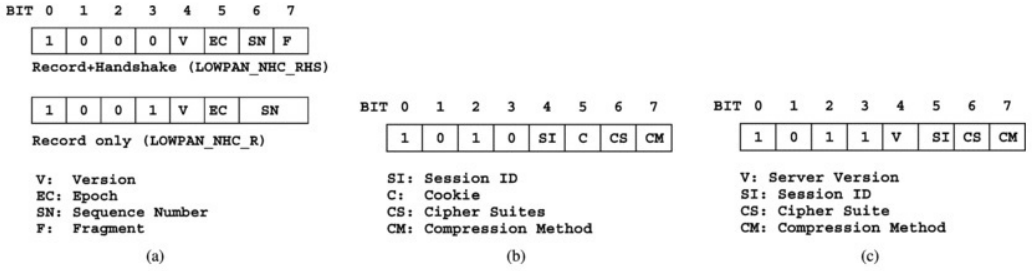

Fig. 4. LOWPAN_NHC encodings for different DTLS headers and Handshake messages [43].

(EC), *Sequence Number* (SN) and *Fragments* (F) can be compressed. Typically, the EC value is either 0 or 1, hence only a 8-bit *Epoch* is used most of the time. The 2-bit representation of SN in the LOWPAN_NHC_R encoding allows for 16-bit, 24-bit, 32-bit or 48-bit *Sequence Number* to be used. As for F, when it is set to 0, the handshake message is not fragmented and the fields *fragment_offset* and *fragment_length* are omitted. If set to 1, the fields *fragment_offset* and *fragment_length* are carried inline.

Figure 4(b) shows the encoding of the $ClientHello$ message (LOWPAN_NHC_CH). The *Random* field in the $ClientHello$ is always carried inline, whereas the *Version* field is always omitted. With this compression scheme, essentially only the *Random* field needs to be transmitted and all other fields can be omitted. This is because the *Session ID* field is 0 when a new handshake is initiated, and *cookie* is an optional field. The *Ciphersuite* and *Compression Method* can be pre-configured to have their respective default values and hence do not need to be negotiated. Figure 4(c) shows the encodings of the $ServerHello$ message (LOWPAN_NHC_SH). It is very similar to LOWPAN_NHC_CH. All other handshake messages in DTLS cannot be compressed and must be carried inline.

3.3 Enhancement and Adaptation to DTLS

Although the general consensus in the community is to re-use the existing Internet security to protect the IoT ecosystem, none of them can be used without adaptation and further enhancements to the devices with severe constraints. Currently, the DTLS protocol receives the most attention from the IoT community.

A new IETF working group called "DTLS In Constrained Environment (DICE)" [22] was approved in August 2013 to:

- Define a DTLS profile that is suitable for IoT applications and is reasonably implementable on many constrained devices.
- Define how DTLS Record Layer can be used to protect multicast messages, assuming that devices in a multicast group are provisioned a group key *a priori*, though the DTLS handshake may be changed to support distribution of group keys in the future.

Prior to DICE, there have been numerous research on enhancing DTLS for use in IoT environments, this section discusses some notable works on securing the IoT using DTLS in IETF.

3.3.1 Raw Public-Key Support

The Pre-Shared Key (PSK) mode of operation in DTLS only supports partial inter-operability between IoT devices because device manufacturers would need to pre-share some keying materials with each other in order to allow for their devices to securely communicate with each other. To establish such a trust in a multi-vendor environment can be difficult. The other end of the spectrum is the X.509-based Public-Key Infrastructure (PKIX) to enable IoT devices to authenticate each other. However, given that most of the IoT devices are resource constrained and the network bandwidth is limited, to support PKIX in an IoT ecosystem is challenging even though it seems to be the preferred choice in many smart metering deployments.

Although many research studies have shown that Elliptic Curve Cryptography (ECC) [33] can be implemented on a resource constrained device, they often do not take into consideration that there are other essential software components and protocol implementations, such as the IPv6 protocol stack, the 6LoWPAN shim layer, DNS-related functionality, DTLS, and the actual application code. Furthermore, the use of certificates would surely result in fragmentation of DTLS Handshake messages if they need to carry a certificate chain to exchange credentials between devices.

The use of raw public keys with DTLS [53] has been standardized to alleviate the burden of IoT devices from storing and transmitting X.509 certificates when performing the DTLS handshake protocol. This allows the devices to be pre-configured (during the manufacturing time) with public keys of dedicated servers that the device needs to communicate with, as well as a public key for the device itself. The binding of the public-key with a server would need to be validated *out-of-band* by the manufacturers during the device configuration time, therefore allowing the devices to authenticate the servers by exchanging raw public-keys instead of X.509 certificates. Other forms of out-of-band validation, such

as QR codes, also exist. The most suitable validation technique depends on the deployment.

When devices execute the DTLS handshake protocol using raw public keys, the newly defined TLS extensions of *client_certificate_type* and *server_certificate_type* must be used. The raw public keys are encapsulated in a SubjectPublicKeyInfo structure [10], that only contains the raw keys and an algorithm identifier.

[53] proposed a DTLS Handshake protocol using raw public keys. The IoT device acting as the DTLS Client initiates the DTLS Handshake protocol, indicating that it is capable of processing raw public-keys when sending the $ClientHello$ message with the *RawPublicKey* extension. The server fulfills the client's request, indicates this via the *RawPublicKey* value in the *server_certificate_type* payload, and provides a raw public key into the Certificate payload back to the client. In case that client authentication is required, the Server can send a $CertificateRequest$ message to the client, demanding the client to send its raw public key for authentication. The Client, who has a raw public-key pre-provisioned, returns it in the $Certificate$ payload to the Server.

3.3.2 Group Communication Security

Group communication can be used to convey messages to a group of devices without requiring the sender to perform time- and energy-consuming multiple unicast transmissions to reach group members. Although there have been a lot of efforts in IETF to standardize mechanisms to secure group communication, they are not necessarily suitable for the IoT environment. For example, the MIKEY Architecture [6] is mainly designed to facilitate multimedia distribution, while TESLA [40] is proposed as a protocol for broadcast authentication of the source and not for protecting the confidentiality of multicast messages.

Assuming that DTLS is the default mandatory must-implement security protocol for securing IoT, it is conceivable that DTLS can be extended to facilitate CoAP-based group communication [27]. Reusing DTLS for different purposes while guaranteeing the required security properties. This avoids the need to implement multiple security protocols and this is especially beneficial when the target deployment consists of resource-constrained embedded devices. [27] proposes an extension to the DTLS Record Layer to encrypt and integrity protect multicast messages assuming that all devices in the group already have a Group Security Association (GSA) pre-configured. The GSA consists of keying material (e.g., group key), security policies, and security parameters to use. When sending a group message, the DTLS Record Layer is used so that multicast messages are encrypted with the group key and protected using a Message Authentication Code (MAC) according to the chosen ciphersuite. The authenticated encrypted message is passed down to the lower layer of the IP protocol stack for transmission to the multicast address.

It is likely that there are multiple senders in a multicast group, and it is important to enable all senders in the group to securely send information using a common group key, while preserving the freshness and integrity of the messages. Each sender can derive a $SenderID$ based on the device's IPv6 or MAC address, or even randomly. The $SenderID$ must be unique for all senders within the specific multicast group. The existing DTLS Record Layer header is adapted [27] such that the 6-byte sequence number field is split into a 1-byte $SenderID$ field and a 5-byte $truncated\ sequence\ number$ field. Each sender in the group uses its own unique $SenderID$ in the DTLS record layer header when sending a multicast message to the group. It also manages its own $epoch$ and $truncated\ sequence\ number$ in the "server write" connection state, hence they do not need to synchronize them with other senders in the group. The main reason to partition the sequence number space according to the sender is to avoid sequence number re-use. In the AES-CCM mode of operation, the sequence number is used as part of the nonce, thus it is crucial to ensure that a particular sequence number is not re-used. If multiple senders use the same sequence number or nonce, and the same group key to encrypt different messages, then the message confidentiality cannot be guaranteed.

Listeners in a multicast group, need to store multiple "client read" connection states for the different senders linked to the $SenderID$s. The keying material is the same for all senders however the $epoch$ and the $truncated\ sequence\ number$ of the last received packets need to be kept differently for different senders. The listeners first perform a "server write" key lookup by using the multicast IP destination address of the packet. By knowing the keys, the listeners decrypt and check the MAC of the message. This guarantees that no one has spoofed the $SenderID$, as it is protected by the MAC. Subsequently, by authenticating the $SenderID$ field, the listeners retrieve the "client read" connection state which contains the last stored $epoch$ and $truncated\ sequence\ number$ of the sender, which is used to check the freshness of the message received. The listeners must ensure that the $epoch$ is the same and $truncated\ sequence\ number$ in the message received is higher than the stored value, otherwise the message is discarded. As each sender manages its own $epoch$ and $sequence\ number$, listeners are confident that these values are reliable. Once the authenticity and freshness of the message have been checked, the listeners can pass the message to the higher layer protocols. The $epoch$ and the $sequence\ number$ in the corresponding "client read" connection state are updated as well.

3.3.3 DTLS Profile for IoT

The Lightweight Implementation Guidance (LWIG) Working Group [23] in IETF has been chartered to collect experiences from implementors of IP stack in constrained devices, in particular, techniques for reducing complexity, memory footprint, or power usage.

[31] looks at the security part of the communication protocol stack and offers input for implementers and system architects to illustrate the costs and benefits of various TLS/DTLS features for use in IoT. The findings of this document serve as an important input to the DICE Working Group [22] to define a DTLS profile for IoT, thus removing complex functionalities that are not required and retaining only those that are applicable to the IoT ecosystem. In addition to that, there are various assumptions and design decisions [31] that could affect the definition of a DTLS profile for IoT:

- *Data confidentiality* – Many TLS ciphersuites provide a variant for NULL encryption [13]. If confidentiality protection is not required, a carefully chosen set of algorithms may have a positive impact on the code size. Re-use of crypto-libraries (within TLS but also among the entire protocol stack) will also help to reduce the overall code size.
- *Hardware support* – Certain hardware platforms offer support for a random number generator as well as cryptographic algorithms (e.g., AES). These functions can be re-used and the amount of required code can thus be reduced. Using hardware support not only reduces the computation time but can also save energy due to the optimized implementation.
- *(D)TLS features* – (D)TLS is a very flexible protocol that can be extended in various ways, for example through ciphersuites. Already the base protocol offers sophisticated functionality for improving the performance, for example by using session resumption. Depending on the application requirements, some of these features may not be needed, and hence reducing the code size of the implementation. In the case of DTLS, generic fragmentation and reordering requires large buffers to reassemble the messages, which might be too heavy for some devices.
- *Credentials* – (D)TLS supports Pre-Shared Keys (PSK), Certificates, and Raw Public Keys. As highlighted in Section 3.2.5, the use of X.509 certificates would lead to message fragmentation. If deployments of IoT does not involve such credentials, then the ASN.1 library as well as the certificate parsing and processing can be omitted. Similarly, if only PSKs are used, then the big integer implementation can be omitted.
- *Server Centric* – Resource-constrained sensor nodes running CoAPS might be server only, allowing for devices status to be probed and queried. The constrained side will most likely be only implementing a single ciphersuite. Flexibility is given to a more powerful counterpart that supports many different ciphersuites for various connected devices.

[28] and [19] made the first attempt to define a DTLS profile for IoT, based on the design decisions described above. This Internet-draft aims to ensure that a compact "IoT profile" should allow for a compact implementation (in terms of code size and RAM) and simplified DTLS handshake between IoT devices.

Ciphersuites – Most constrained IoT devices cannot support multiple cipher implementations due to code space requirements. It can be beneficial to choose a few ciphersuite profiles that could cover the security requirements of most IoT applications. In choosing these ciphersuite profiles, reuse of the same crypto primitives to achieve different security functionality can further reduce implementation code space.

For symmetric cipher, confidentiality and authentication functionality can be achieved by using the AES in CCM mode of operation. Further, the AES-CCM operation is built-in on many 802.15.4 hardware chips, thus further reducing the need in code and also accelerating the computation. [35] indicates different ciphersuites based on AES-CCM for TLS.

For public key based handshake, ECC is very suitable for constrained devices. However, there are multiple options in terms of field types and curves that can be chosen for a ciphersuite [36], [8]. Additionally, for certificate based ciphersuites, choosing the certificate signing algorithm to be also ECC based avoids the need for an additional crypto primitive implementation on the constrained devices. Selecting a default field, curve and algorithm as a public key based IoT ciphersuite would ensure security of IoT applications and can substantially reduce the negotiation required in the handshake phase.

DTLS Extensions – Further improvements to DTLS in constrained environments can be made by choosing some of the TLS extensions [2] that are always supported by the end-points. Some of these extensions have been designed for constrained networks which can be used to define the DTLS IoT profile.

The "Maximum Fragment Length Negotiation" extension enables a smaller fragment sizes that would reduce the amount of fragmentation at the lower layers. "Client Certificate URLs" extension reduces the need for sending the certificates in the handshake message, thus reducing bandwidth requirements and fragmentation due to large certificates. Other extensions that may be useful are the "Trusted CA Indication", "Truncated HMAC" and "Certificate Status Request".

By choosing a mandatory set of extensions as part of the DTLS IoT profile will make DTLS more efficient in constrained environments.

Fine-tuning DTLS functionality – Savings can also be done by choosing not to implement certain DTLS functional logic that is not expected to be used in most IoT applications. The RESUME protocol is useful for IoT applications. On one hand, it simplifies the Handshake protocol if devices need to re-use the

previous security parameters, on the other hand it increases the complexity of the DTLS Handshake state machine. It is very likely that the DTLS sessions in IoT application are meant to be long-lived, and re-handshake or resumption of previous session could be avoided if possible. Additionally, reducing the number of error handling logic as part of the Alert protocol is advocated. These reduced functionalities should not in any way affect the security of the DTLS but only reduce the flexibility that was designed into DTLS as a web protocol but may not be required in IoT applications.

Furthermore, timer values for retransmission can be adjusted to prevent unnecessary congestion due to the underlying lossy network which can be aggravated due to large flight messages being resent at short intervals.

To conclude this section, the DTLS IoT profile can be a combination of ciphersuites, DTLS extensions and fine-tuning functionality that makes it suitable for constrained devices and networks.

4 Performance Evaluation and Analysis

This section summarizes the performance measurements available from various implementations presented in the IETF. in order to support the claims in this paper.

4.1 DTLS for IoT Security

Most the prototype implementations on embedded system platform such as Contiki OS, only support the PSK mode of operation. One notable DTLS implementation is based on Redbee Econotag hardware which features a 32-bit CPU, 128 KB of ROM and 96 KB of RAM and an IEEE 802.15.4 enabled radio with an AES hardware co-processor [26], [16]. It is observed that it is feasible to use DTLS to provide authentication and end-to-end security in IoT. The developed prototype was based on TinyDTLS [7] library and included most of the extensions and the adaptation as follows:

- The cookie mechanism was disabled in order to fit messages to the available packet sizes and hence reducing the total number of messages when performing the DTLS handshake.
- Separate delivery was used instead of flight grouping of messages and redesigned the retransmission mechanism accordingly.
- The "TinyDTLS" AES-CCM module was modified to use the AES hardware coprocessor.

Table 1 presents the codesize and memory consumption of the prototype differentiating (i) the state machine for the handshake, (ii) the cryptographic primitives, and (iii) the DTLS record layer mechanism. The use of DTLS appears to incur large memory footprint both in ROM and RAM for two reasons. First, DTLS handshake defines many message types and this adds more complexity to its corresponding state machine. The logic for message re-ordering and retransmission also contributes to the complexity of the DTLS state machine. Second, DTLS uses SHA2-based crypto suites which is not available from the hardware crypto co-processor.

TABLE 1
Memory requirements for DTLS in KB [26]

	DTLS	
	ROM	RAM
State Machine	8.15	1.9
Cryptography	3.3	1.5
Key Management	1.0	0
DTLS record layer	3.7	0.5
Total	**16.15**	**3.9**

The DTLS protocol implementation was further examined and evaluated by tuning the packet loss ratio as some UDP packets are bound to get lost due to network congestion and limited network bandwidth with IEEE 802.15.4. In particular, the impact of packet loss on message delay, success rate and number of messages exchanged in the handshake were examined. According to [16], in a network with packet losses, DTLS performs badly because the security handshake might have failed due to the lost messages. Consequently, this had increases the delays. [16] shows the different outcomes for the percentage of successful handshakes as a function of timeouts and packet loss ratios. As expected, a higher packet loss ratio and smaller timeout (15s timeout) result in a failure probability of completing the DTLS handshake.

Delays in network access and communication are intolerable since they lead to higher resource consumption. As the solution relied on PSK, the handshake protocol only incurred a short delay of a few milliseconds when there was no packet loss. However, as the packet loss ratio increased, the delay in completing the handshake became significant because loss packets must be retransmitted. Finally, another important criterion is the number of messages exchanged in the presence of packet loss. A successful handshake could incur up to 35 or more messages to be transmitted when the packet loss ratio reaches 0.5. This is mainly because the DTLS retransmission is complex and often requires re-sending multiple messages even when only a single message has been lost.

An 802.15.4 IoT network would typically have a packet loss ratio between 0.2 and 0.3, which implies that using DTLS handshake to provide network access authentication and key management would be feasible. Furthermore, in most deployment scenarios, the DTLS handshake is performed only once to establish a security channel to distribute keying materials and later on to renew the session key. Consequently, deploying DTLS in IoT as the sole security protocol suite is a viable approach.

4.2 Avoiding Fragmentation through Compression

Experiments and analysis had been conducted by [43] to investigate whether header compression schemes can be

used to avoid packet fragmentation in DTLS handshake protocol altogether in an IoT network.

The aim is to avoid packet fragmentation because when a DTLS handshake message needs to be fragmented, it increases the protocol complexity as all fragments must be retransmitted (if lost), re-ordered (if arrived out of order), and finally re-assembled to construct the DTLS handshake message. Consequently, there is an extra effort of retransmission, re-ordering and re-assembly for each fragmented DTLS handshake message. However, if each DTLS handshake message together with the relevant UDP, IP, and 802.15.4 headers can fit into an 802.15.4 packet, only the lost message needs to be retransmitted and the receiver only needs to re-order the messages received.

According to the results in [43], using 6LoWPAN-NHC compression mechanism can significantly reduce the length of DTLS headers. Reducing the header bits also results in the reduction of radio transmission time. Table 2 shows the number of bits reduced and the total space saving for DTLS headers and messages.

TABLE 2
Number of Bits sent and Space Saving for DTLS [43]

DTLS headers	Without Comp. [Bit]	With Comp. [Bit]	Space Saving
Record	104	40[1]	62%
Handshake	96	24[1]	75%
Client Hello	336[2]	264[2]	23%
Server Hello	304	264[3]	14%
Certificate Request	40	0	100%

[1] An extra byte was used to encode both the Record and Handshake headers.
[2] Some fields have variable length, only bits that are always sent were considered.
[3] Did not compromise on security and sent full size *random number*. All other fields were omitted.

The Record header was compressed by 64 bits (8 bytes), thus a saving of 62% of space for each message. For the Handshake header, a 75% of saving was achieved, where it only required 24 bits (3 bytes). With this, the combined DTLS Record and Handshake headers only constitutes 8 bytes. This means that approximately 56 bytes of payload can be encoded. As a result, for PSK-mode of operation, all DTLS handshake messages can fit into a 802.15.4 packet without requiring fragmentation. For example, the compressed *Client Hello* and *Server Hello* messages cost 33 bytes each. By adding the 8 bytes header to each message, it only consumes 41 bytes, which is less than the maximum size allowed, i.e., 64 bytes. Furthermore, by avoiding fragmentation altogether when performing DTLS handshake, it also reduces the communication overhead in the network.

Analysis also revealed that the overhead incurred through in-node computation for compression and decompression of DTLS headers is almost negligible [43]. Based on the Contiki's energy estimation module, on average 4.2 μJ of energy is consumed for compression [43]. In fact, when DTLS handshake messages are not fragmented, it results in less packet transmission.

TABLE 3
Average Energy Consumption for DTLS Packet Transmission [43]

Compression	Client side [μJ]	Server side [μJ]	Total [μJ]
Without	1756.66	1311.65	3068.31
With	1467.54	1143.47	2611.01

Interestingly, when compression is applied, on average 15% less energy is used to transmit (and receive) compressed packets. This is due to smaller packet sizes achieved through compression. Table 3 shows the average energy consumption for packet transmission during DTLS handshake.

5 Future Work

5.1 DTLS Profile and TLS Version 1.3

It is rather clear that the security community in IETF is relying on DTLS as the standard security protocol for IoT, although a lot more needs to be done to better adapt the protocol (which was initially designed for the Web) for deployment on embedded devices. The DICE WG [22] will define a profile which includes only a subset of the DTLS functions.

In order to keep up with the latest attacks and development of cryptographic algorithms, security protocols are constantly being updated and upgraded. Ideally, the DTLS profile defined in DICE WG should not be the end product for IoT security. It is the intention to use the DTLS profile as one of the inputs to the design and requirements of (D)TLS version 1.3, so that the new (D)TLS version can be used to protect the application layer messages of the Internet, the Web and IoT.

5.2 Software and Key Provisioning

Provisioning IoT devices with software and keys is an important but complex process. Providing a possibility to update software components and the entire firmware image is important to ensure that the devices are always running the latest software versions. This update cannot, like on the PC environment, happen with the support of end users but has to be performed unattended. Part of this software provisioning process is also the ability to provision keying material to devices. This is an extremely sensitive step since as adversary that can compromise the process can take control of the device. With the increase in the number of devices, it means that more cryptographic keys need to be managed.

5.3 Authorization and Centralized Device Management

End users and system administrators are keen on managing devices and their access rights in a seamless way that allows the integration with already existing infrastructure. A smart home, for example, requires a user

to install various devices in the home and to manage their access rights properly. Developing standards for such an authentication and authorization infrastructure, by re-using and tailoring trusted third party identity and access control infrastructure, is currently discussed on the ACE mailing list [24] with the goal to form a new working group in the IETF. This work is seen as important to ensure seamless user experience.

5.4 Hardware Improvements

According to Moore's Law, the capabilities (processing speed, memory capacity, sensors) of devices are improving at roughly exponential rate. The IoT community is currently facing interesting challenges: devices that offer more memory and substantial processing power come at higher hardware cost and software development for those devices is much easier since many of the off-the-shelf tools and programming languages can be re-used. Unfortunately, they consume more energy and the low power radio technologies always cause challenges due to their limited bandwidth. For devices with lower processing capabilities, less memory, and low power consumption, it still remains a challenge to run complete Internet protocol stacks on them.

In the last decade, the mobile phones vendors had been working hard to adapt network protocols, security suites, and to simplify applications protocols, so that they could run on PDAs and mobile devices. However, ever since the iOS and Android based smart phones were released, these adapted protocols and security suites have become obsolete, mainly because these smart devices are capable of running the full IP communication stack that conforms to the current Internet standards. Given time, the computational capabilities of embedded systems will increase tremendously and be able to runn the full IP protocol stack. The challenge remains to see if history repeats in that the transition we saw in the mobile industry will also happen to IoT world, and whether CoAP/6LoWPAN/DTLS profiles are just interim solutions for the IoT ecosystem.

6 Conclusions

A standardized security protocol is indispensable for the success of Internet-of-Things (IoT). When every object in our daily life is connected to the Internet, they must speak the same (security) protocol to ensure interoperability. The standardization efforts in IETF is therefore a very important effort to make IoT a reality.

Replicating the success of TLS in the Internet in the context of IoT is a challenging process, primarily because DTLS was not designed for constrained environment. However, the community is working towards a single security suite that is based on DTLS to provide security functionalities to the IoT devices. Previous research has revealed that DTLS optimization is required in order to reduce the complexity of its state machine and several optional functional logics could be omitted for IoT deployment.

Standardizing the communication security for IoT is the first step towards an inter-operable IoT. There are concerns about device bootstrapping, key management, authorization, privacy and message fragmentation issues in the IoT as well. However, not all solutions for these security problems need to be standardized, some of them could remain as proprietary solutions targeted to specific application domains. We advocate that device bootstrapping and key management should be standardized soon in the future to provide a common management interface to facilitate secure device commissioning and configuration. This would allow for large scale deployment of IoT to be feasible.

Finally, the DICE WG has been tasked to define the adaptation and enhancements to DTLS. Together with the TLS, CoRE and LWIG working groups in IETF, it is expected that a standardized security framework for the IoT that is inter-operable with the existing Internet can be a reality in the near future.

References

[1] IEEE Standard for Information Technology- Telecommunications and Information Exchange Between Systems- Local and Metropolitan Area Networks- Specific Requirements Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (WPANs). Technical report, 2006.

[2] D. Eastlake 3rd. Transport Layer Security (TLS) Extensions: Extension Definitions. RFC 6066, January 2011.

[3] Open Mobile Alliance. Wireless Application Protocol (WAP 2.0) Architecture Specification, 2001.

[4] Open Mobile Alliance. Wireless Transport Layer Security (WTLS), 2001.

[5] Zigbee Alliance. Zigbee-IP Specification, March 2013.

[6] J. Arkko, E. Carrara, F. Lindholm, M. Naslund, and K. Norrman. MIKEY: Multimedia Internet KEYing. RFC 3830 (Proposed Standard), August 2004. Updated by RFCs 4738, 6309.

[7] O. Bergmann. TinyDTLS - A Basic DTLS Server Template.

[8] S. Blake-Wilson, N. Bolyard, V. Gupta, C. Hawk, and B. Moeller. Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS). RFC 4492 (Informational), May 2006. Updated by RFCs 5246, 7027.

[9] Cisco. The internet of things. Jan 2014.

[10] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, and W. Polk. Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. RFC 5280 (Proposed Standard), May 2008. Updated by RFC 6818.

[11] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246 (Proposed Standard), August 2008. Updated by RFCs 5746, 5878, 6176.

[12] Ericsson. More than 50 billion connected devices. *Ericsson White Paper 284 23-3149 Uen*, Feb 2011.

[13] P. Eronen and H. Tschofenig. Pre-Shared Key Ciphersuites for Transport Layer Security (TLS). RFC 4279, December 2005.

[14] R. Moskowitz et al. Host Identity Protocol Version 2 (HIPv2). Internet-draft, IETF, 2012.

[15] S. Farrell and H. Tschofenig. Pervasive Monitoring is an Attack. Internet-Draft draft-farrell-perpass-attack-06.txt, Internet Engineering Task Force, 2 2013. Work in progress.

[16] O. Garcia-Morchon, S.L. Keoh, S. Kumar, P. Moreno-Sanchez, F. Vidal-Meca, and J.H. Ziegeldorf. Securing the IP-based internet of things with HIP and DTLS. In *Proceedings of the sixth ACM conference on Security and privacy in wireless and mobile networks*, WiSec '13, pages 119–124, Budapest, Hungary, 2013. ACM.

[17] D. Hardt. The OAuth 2.0 Authorization Framework. RFC 6749 (Proposed Standard), October 2012.

[18] K. Hartke. Practical Issues with Datagram Transport Layer Security in Constrained Environments. Internet Draft draft-hartke-dice-practical-issues-00, IETF, 2012.
[19] K. Hartke and H. Tschofenig. A DTLS 1.2 Profile for the Internet of Things. Internet-Draft draft-ietf-dice-profile-00, IETF, 2014.
[20] J. Hui and P. Thubert. Compression Format for IPv6 Datagrams over IEEE 802.15.4-Based Networks. RFC 6282 (Proposed Standard), September 2011.
[21] IETF. Constrained RESTful Environments (CORE) WG, 2013.
[22] IETF. DTLS In Constrained Environment (DICE) WG, 2013.
[23] IETF. Lightweight Implementation Guidance (LWIG) WG, 2013.
[24] IETF. Authentication and Authorization for Constrained Environments (ace) Mailing List, April 2014.
[25] S. Kent and K. Seo. Security Architecture for the Internet Protocol. RFC 4301, December 2005. Updated by RFC 6040.
[26] S.L. Keoh, S.S. Kumar, and O. Garcia-Morchon. Securing the IP-based Internet of Things with DTLS. Internet-Draft draft-keoh-lwig-dtls-iot-02, IETF, 2013.
[27] S.L. Keoh, S.S. Kumar, O. Garcia-Morchon, and E. Dijk. DTLS-based Multicast Security for Low-Power and Lossy Networks. Internet-Draft draft-keoh-dice-multicast-security-01, IETF, 2013.
[28] S.L. Keoh, S.S. Kumar, and Z. Shelby. Profiling of DTLS for CoAP-based IoT Applications. Internet-Draft draft-keoh-dtls-profile-iot-00, IETF, 2013.
[29] T. Kivinen. Minimal IKEv2. Internet-Draft draft-ietf-lwig-ikev2-minimal-01, IETF, 2013.
[30] S.S. Kumar, S.L. Keoh, and O. Garcia-Morchon. DTLS Relay for Constrained Environments. Internet-draft, IETF, 2013.
[31] S.S. Kumar, S.L. Keoh, and H. Tschofenig. A Hitchhikerś Guide to the (Datagram) Transport Layer Security Protocol. Internet-Draft draft-ietf-lwig-tls-minimal-00, IETF, 2013.
[32] N. Kushalnagar, G. Montenegro, and C. Schumacher. IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs): Overview, Assumptions, Problem Statement, and Goals. RFC 4919 (Informational), August 2007.
[33] An Liu and Peng Ning. Tinyecc: A configurable library for elliptic curve cryptography in wireless sensor networks. In *Proceedings of the International Conference on Information Processing in Sensor Networks (IPSN '08)*, pages 245–256, 2008.
[34] P. Mcdermott-Wells. What is Bluetooth? *Potentials, IEEE*, 23(5):33–35, 2005.
[35] D. McGrew and D. Bailey. AES-CCM Cipher Suites for Transport Layer Security (TLS). RFC 6655 (Proposed Standard), July 2012.
[36] J. Merkle and M. Lochter. Elliptic Curve Cryptography (ECC) Brainpool Curves for Transport Layer Security (TLS). RFC 7027 (Informational), October 2013.
[37] G. Montenegro, N. Kushalnagar, J. Hui, and D. Culler. Transmission of IPv6 Packets over IEEE 802.15.4 Networks. RFC 4944, September 2007. Updated by RFCs 6282, 6775.
[38] R. Moskowitz. HIP Diet EXchange (DEX). Internet Draft draft-moskowitz-hip-rg-dex-06, IETF, 2012.
[39] J. Nieminen, T. Savolainen, M. Isomaki, B. Patil, Z. ZachShelby, and C. Gomez. Transmission of IPv6 Packets over BLUETOOTH Low Energy. Internet-Draft draft-ietf-6lowpan-btle-12, Internet Engineering Task Force, 02 2012. Work in progress.
[40] A. Perrig, D. Song, R. Canetti, J. D. Tygar, and B. Briscoe. Timed Efficient Stream Loss-Tolerant Authentication (TESLA). RFC 4082, June 2005.
[41] P. PeterMariager, J. Petersen, and Z. Shelby. Transmission of IPv6 Packets over DECT Ultra Low Energy. Internet-Draft draft-mariager-6lowpan-v6over-dect-ule-03.txt,, Internet Engineering Task Force, 07 2015. Work in progress.
[42] S. Raza, S. Duquennoy, T. Chung, D. Yazar, T. Voigt, and U. Roedig. Securing communication in 6lowpan with compressed ipsec. In *Proceedings of the International Conference on Distributed Computing in Sensor Systems and Workshops (DCOSS)*, 2011.
[43] S. Raza, H. Shafagh, K. Hewage, R. Hummen, and T. Voigt. Lithe: Lightweight secure coap for the internet of things. *Sensors Journal, IEEE*, 13(10):3711–3720, 2013.
[44] E. Rescorla and B. Korver. Guidelines for Writing RFC Text on Security Considerations. RFC 3552, July 2003.
[45] E. Rescorla and N. Modadugu. Datagram Transport Layer Security Version 1.2. RFC 6347 (Proposed Standard), January 2012.
[46] R. Riaz, Ki-Hyung Kim, and H.F. Ahmed. Security analysis survey and framework design for IP connected LoWPANs. In *Proceedings of the International Symposium on Autonomous Decentralized Systems (ISADS '09)*, pages 1–6, 2009.
[47] R. Roman and J. Lopez. Integrating wireless sensor networks and the internet: a security analysis. *Internet Research*, 19(2), 2009.
[48] R. Roman, P. Najera, and J. Lopez. Securing the internet of things. *IEEE Computer*, 44(9):51–58, 2011.
[49] Bruce Schneier. The internet of things is wildly insecure - and often unpatchable. *Wired*, Jan 2014.
[50] Z. Shelby, S. Chakrabarti, E. Nordmark, and C. Bormann. Neighbor Discovery Optimization for IPv6 over Low-Power Wireless Personal Area Networks. RFC 6775, November 2012.
[51] Z. Shelby, K. Hartke, and C. Bormann. Constrained Application Protocol (CoAP). Internet Draft draft-ietf-core-coap-18, IETF, 2013.
[52] T. Winter, P. Thubert, A. Brandt, J. Hui, R. Kelsey, P. Levis, K. Pister, R. Struik, JP. Vasseur, and R. Alexander. RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks. RFC 6550 (Proposed Standard), March 2012.
[53] P. Wouters, H. Tschofenig, J. Gilmore, S. Weiler, and T. Kivinen. Using Raw Public Keys in TLS and DTLS. Internet-Draft draft-ietf-tls-oob-pubkey-10, IETF, 2013.

Sye Loong Keoh is an Assistant Professor in the School of Computing Science, University of Glasgow Singapore. He obtained his PhD in computing science from Imperial College London in 2005. From 2008-2013, he was a Senior Scientist at Philips Research Eindhoven. His research interests include policy-based management, trust and security for wireless sensor networks, smart grid and pervasive computing.

Sandeep S. Kumar is a Senior Scientist in Philips Research Eindhoven since 2006. He received his PhD from Ruhr-University of Bochum (HGI, eurobits) in applied data security in 2006. He has worked extensively in the areas of applied data security and cryptography in embedded systems. His research focuses are data security, secure distributed systems and networked controls.

Hannes Tschofenig is employed by ARM Limited and spends most of his time on Internet standardization, particularly in the area of security, privacy and emergency services. His favorite organization is the IETF where he co-chairs the OAuth WG, developing solutions for secure and privacy-friendly data sharing. As an active IETF participant with over 50 RFCs, he gets excited about the intersection of technology and regulation.