Future Generation Computer Systems 77 (2017) 40–51

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

SecureSense: End-to-end secure communication architecture for the cloud-connected Internet of Things

Shahid Raza [a,*], Tómas Helgason [a], Panos Papadimitratos [b], Thiemo Voigt [a,c]

[a] RISE SICS, Isafjordsgatan 22, Stockholm, Sweden
[b] Networked Systems Security Group, KTH Royal Institute of Technology, Stockholm, Sweden
[c] Department of Information Technology, Uppsala University, Sweden

HIGHLIGHTS

- Extending IoT capabilities from the PSK-based CoAP to RPK- and certificate-based CoAP.
- Bring CoAP and DTLS-based security in SicsthSense, a cloud platform for the IoT.
- Extensive empirical evaluation of E2E security for the cloud-connected IoT.

ARTICLE INFO

Article history:
Received 15 October 2015
Received in revised form 6 June 2017
Accepted 13 June 2017
Available online 28 June 2017

Keywords:
Security
Internet of Things
IoT
CoAP
DTLS
Cloud

ABSTRACT

Constrained Application Protocol (CoAP) has become the de-facto web standard for the IoT. Unlike traditional wireless sensor networks, Internet-connected smart thing deployments require security. CoAP mandates the use of the Datagram TLS (DTLS) protocol as the underlying secure communication protocol. In this paper we implement DTLS-protected secure CoAP for both resource-constrained IoT devices and a cloud backend and evaluate all three security modes (pre-shared key, raw-public key, and certificate-based) of CoAP in a real cloud-connected IoT setup. We extend SicsthSense– a cloud platform for the IoT– with secure CoAP capabilities, and compliment a DTLS implementation for resource-constrained IoT devices with raw-public key and certificate-based asymmetric cryptography. To the best of our knowledge, this is the first effort toward providing end-to-end secure communication between resource-constrained smart things and cloud back-ends which supports all three security modes of CoAP both on the client side and the server side. SecureSense– our End-to-End (E2E) secure communication architecture for the IoT– consists of all standard-based protocols, and implementation of these protocols are open source and BSD-licensed. The SecureSense evaluation benchmarks and open source and open license implementation make it possible for future IoT product and service providers to account for security overhead while using all standardized protocols and while ensuring interoperability among different vendors. The core contributions of this paper are: (i) a complete implementation for CoAP security modes for E2E IoT security, (ii) IoT security and communication protocols for a cloud platform for the IoT, and (iii) detailed experimental evaluation and benchmarking of E2E security between a network of smart things and a cloud platform.

1. Introduction

Internet Protocol v6 (IPv6) with potentially unlimited address space and its header compression using the 6LoWPAN (IPv6 over Low-power Wireless Personal Area Networks) standard make it possible to connect everyday physical *things*, having a tiny embedded computer and limited storage and communication capabilities, with the Internet. This network of smart things, called 6LoWPAN network, and its interconnection with Internet hosts (standard computers, smartphones, computing clouds, etc.) forms the Internet of Things (IoT). Most smart things are resource-constrained and cannot run heavy-weight applications and therefore connected with cloud backends that host sophisticated intelligent services built using the data produced by smart things. IoT devices are projected to be in billions with heterogeneous capabilities. To ensure interoperability among IoT devices, different IoT protocols are being standardized. In particular, Constrained Application Protocol (CoAP), a lightweight variant of HTTP, is standardized to provide

[*] Corresponding author.
E-mail addresses: shahid@sics.se (S. Raza), tomash@sics.se (T. Helgason), papadim@kth.se (P. Papadimitratos), thiemo@sics.se (T. Voigt).
URL: http://www.ShahidRaza.info (S. Raza).

http://dx.doi.org/10.1016/j.future.2017.06.008
0167-739X/© 2017 Elsevier B.V. All rights reserved.

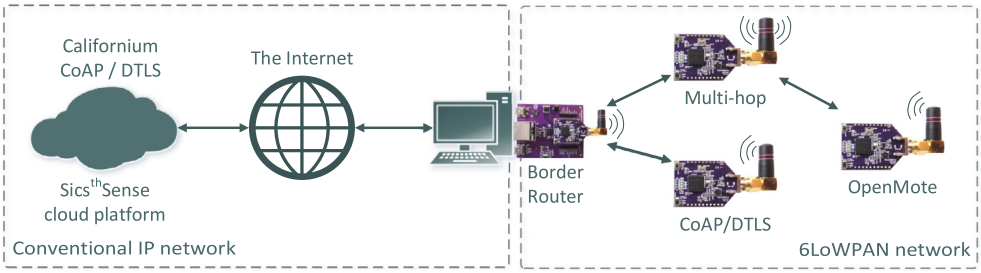

Fig. 1. A SecureSense end-to-end IoT setup securely interconnecting 6LoWPAN networks of resource-constrained IoT nodes (OpenMotes) with a cloud backend (SicsthSense), using the CoAP and DTLS protocols.

web capabilities in the IoT. CoAP has become the de-facto web standard for the IoT.

Real-world Internet-connected IoT deployments require security. CoAP mandates the use of the Datagram Transport Layer Security (DTLS) as the underlying secure communication protocol. Standard-based IoT and security protocols can ensure strong security and interoperability between End-to-End (E2E)-connected applications involving resource-constrained smart things and cloud-backend. For example, a smart grid equipped with cloud computing and intelligent data analytics services can balance the demand and supply of electricity to smart homes if complemented with frequent electricity usage data from smart homes. In such as a setup, smart homes are connected to a grid through a smart meter. It is likely that smart meters are not fully trusted and sensitive private household data must be protected E2E between smart homes devices and a smart grid; we provide such secure communication paradigms in this paper.

We present SecureSense, a communication architecture fully based on standardized protocols, which ensures E2E secure communication between resource-constrained IoT devices and standard Internet hosts. We do not claim that we are the only one to provide secure communication between IoT devices and computing clouds (Section 2 discusses related proposals in detail.); however, we are the first to provide the DTLS-protected secure communication architecture and its implementation and performance evaluation for the cloud-connected IoT with all three security modes of CoAP: Pre-Shared Key (PSK)-based, Raw-Public Key (RPK)-based, and X.509 certificate-based. Fig. 1 shows a SecureSense setup.

On the resource-constrained network side, we extend Lithe [1]– our lightweight PSK-based DTLS and its integration with CoAP for the Contiki OS– with RPK and certificate-based public key crypto that uses Elliptic Curve Cryptography (ECC), which can be used both as the DTLS client and server. On the Cloud-side we extend SicsthSense, our cloud platform for the IoT [2]. Prior to this paper, SicsthSense though supports HTTP and TLS but does not support CoAP and DTLS. We therefore compliment SicsthSense with CoAP and DTLS to make it ready for the IoT protocols.

With these extended capabilities, the major part of this paper presents performance evaluation of SecureSense. In our evaluation, among others, we evaluate the time (that in turn translates to energy) overhead of different CoAP security modes, the time and energy to compute individual DTLS messages, the average DTLS handshake completion time for different security modes, the roundtrip time between a 6LoWPAN node and SicsthSense for multiple hops and multiple data sizes, and the ROM and RAM overhead in a 6LoWPAN node. All SecureSense protocols are standard-based and our implementations of all these protocols are open source and open license, ensuring interoperability among different vendors and adoption by different IoT product and service provider (especially IoT startups) without delving into expensive licensing issues and by avoiding implementation costs.

Following are the main contributions of this paper,

- We extend the capabilities of 6LoWPAN nodes from PSK-based secure CoAP to RPK-based and certificate-based CoAP, which use both our open source and BSD-licensed ECC implementation and the ECC library provided by a hardware crypto.
- We provide DTLS-based security for SicsthSense and integrate it with a CoAP implementation.
- With these novel features, the major part of our contribution is the extensive evaluation of all security modes of CoAP for the cloud-connected IoT.

The rest of the paper is organized as follows. In Section 2, related work is summarized and a brief overview is given of the technologies and concepts used in this paper. In Section 3, we introduce our architecture for SicsthSense and the cloud connected IoT devices. Our implementation is briefly outlined in Section 4. In Section 5, we describe our experimental setup and discuss the evaluation results. Finally, Section 6 concludes this paper.

2. Background and related work

2.1. Cloud connected IoT

Cloud computing allows sharing the resources of powerful servers in data centers with other devices like smart phones and personal computers. Using the same technology for the IoT allows resource-constrained sensor devices to send and store their measurements in a central location accessible by multiple other devices. Cloud computing also makes it possible to perform advance analytics and build sophisticated services using the sensor data from the IoT devices. Multiple cloud platform design guidelines, frameworks and implementations have been put forward for the IoT [3–5]. These solutions are either closed sourced or discussion connectively issues and not addressing security.

Leshan [6] is an OMA Lightweight M2M (LWM2M) open source implementation in Java that is currently under development. It provides libraries to help others develop their own lightweight M2M server or client. SicsthSense [2] is a cloud platform that can easily store data from external sources and allows external interaction. SicsthSense centers around the collection and processing of data streams. All streams are a part of a resource that can keep track of multiple streams. Each resource can have defined multiple parsers that are used to split incoming data to correct streams. A stable open-source version is available with all the core functionalities to store data generated by IoT devices. SicsthSense was missing support for lightweight and secure communications solution for IoT devices. Therefore, we extend SicsthSense and integrate it with SecureSense.

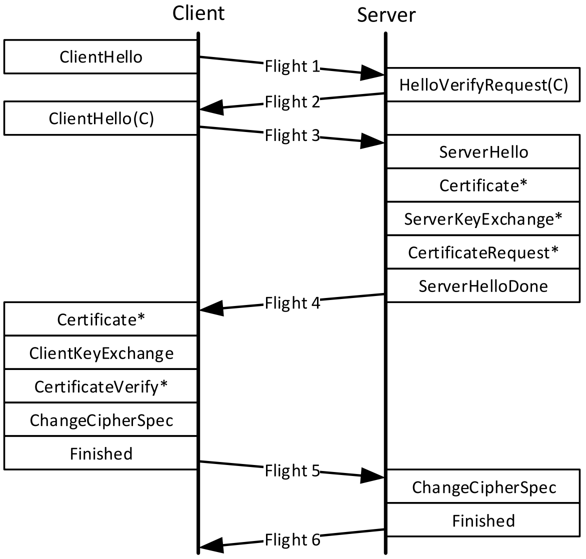

Fig. 2. DTLS handshake message exchange (* only used for some cipher suites).

2.2. CoAP and DTLS

The Internet Engineering Task Force (IETF) recently standardized the Constrained Application Protocol (CoAP) [7], a lightweight variant of HTTP, for the IoT. CoAP runs over UDP and was primarily designed to be used with constrained devices. CoAP makes use of the REST architecture that is common with HTTP while providing low header overhead and parsing complexity. The CoAP standard mandates DTLS for providing communication security between two CoAP end-points. CoAP defines four different security modes: NoSec when DTLS is disabled, Pre-Shared Key (PSK), Raw Public Key (RPK), and Certificate.

Datagram Transport Layer Security (DTLS) [8] is a variant of Transport Layer Security (TLS) designed to work over connectionless UDP instead of TCP. DTLS supports automatic key management and provides data encryption, integrity protection and authentication. It also supports protection against replay and Denial of Service (DoS) attacks. DTLS uses the Handshake protocol to establish a secure session, which is a chatty protocol and exchanges multiple messages during the handshake process. DTLS uses the Record Protocol for cryptographically protecting all messages once the handshake is complete. In the DTLS Handshake protocol, each transmission between a client and a server is defined as a flight shown in Fig. 2. Different cipher suites can be used in the handshake to establish the secure session, and the actual handshake messages therefore vary among cipher suites. In our implementation we use the TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8 and TLS_PSK_WITH_AES_128_CCM_8 cipher suites, recommended by the CoAP standard.

To cope with the Maximum Transmission Unit (MTU) size limitations in IEEE 802.15.4 based networks, the 6LoWPAN header compression mechanisms are standardized [9]. We have already extended the 6LoWPAN header compression mechanisms to the DTLS protocol in Lithe [1]. In Lithe, we have evaluated that the compressed DTLS has lower energy consumption and response time compared to standard uncompressed DTLS.

Public-key cryptography has been implemented and efficiently used with dedicated hardware modules for constrained devices before [10]. An extension to this work adds end-to-end security with two-way authentication implementation but using dedicated hardware module for RSA and Elliptic Curve Cryptography (ECC) and not using the IoT protocols such as CoAP [11]. Due to the smaller key sizes, ECC cryptography is a more suitable for IoT devices than RSA [12]. An ECC cryptography library for Contiki that is both open source and BSD licensed was recently implemented [13]. It has been shown that implicit certificates decrease the transmission and verification overheads of traditional public-key certificates [14], although they will not replace traditional certificates. Another way to decrease the handshake transmission is to use the proposed Cached Information Extension for TLS [15], that allows client to cache server information and therefore that information is not needed in future handshakes. Moreover, a comprehensive session resumption, denial-of-service protection and retransmission mechanisms for DTLS for constrained networks have been introduced in recent paper [16].

Another way to cope with the resource constraint of a sensor node is to delegate the resource intensive DTLS connection establishment to another trusted party. Several delegation approaches have been proposed [17,18], but most of them make use of the border router for delegation. This means that the border router needs to be fully trusted and protected against attacks. Different approach is presented in recent paper by using delegation server instead [19]. However, there the problem is still that the server needs to be fully trusted, and if compromised, all of the sensor nodes in the delegated network are also compromised. Using delegation also means that the E2E security is broken at the delegation entity. When both storage and communication security is required in the IoT, we have presented a solution to reduced the number of cryptographic by combining the storage and communication security operations [20].

Recently we have investigated different attacks against DTLS and improved DTLS resistance against denial of service and enhanced the key provisioning scalability of the DTLS handshake [21]. We have also provided a DTLS-based solution that only uses symmetric cryptography but is scalable to billions of device [22]; however, it requires a trusted third party. All these previous approaches for efficient DTLS are complementary to our work. The focus of this paper is to implement and evaluate DTLS and CoAP between a 6LoWPAN network and a cloud platform enabling E2E security and providing clear performance benchmarks for security while using all standardized protocols.

2.3. IoT hardware

IoT devices are small sensor nodes that are mostly resource-constrained compared with other devices connected to the Internet in terms of energy, storage, and processing resources. These devices should be able to run for a long time on a battery power alone, while at the same time communicating to other devices on the Internet. *Terminology for Constrained-Node Networks* [23] defines three different classes of constrained devices and provide approximate device capabilities. The class 1 devices covers sensor nodes that cannot communicate with typical Internet host using the available standardized Internet security protocols such as DTLS and IPsec. The class 2 covers devices that have just enough resources to securely communicate with Internet hosts using for example the CoAP and DTLS protocols but in the CoAP PSK mode. IoT devices in class 3, with at least 32K of RAM and a couple of hundreds kilobytes of ROM, are able to support DTLS and IPsec with certificates, while still leaving enough space for applications. SecureSense uses OpenMote [24] sensor nodes having capabilities in between the Class 2 and Class 3 devices. OpenMote has enough resources to run standard Internet security protocols while still having the benefit of energy efficiency.

3. SecureSense: Standard-based E2E secure communication in the cloud-connected IoT

It is possible to connect multiple sensor nodes to a cloud platform on the conventional Internet in a number of ways. One way is to separate the two networks using two different protocol stacks. In this case, it is not possible to communicate directly from one network to the other, without translating all the messages at special computers (proxies) at the border of the two networks. Such an architecture requires trusting the intermediaries (gateways, border routers, etc.) which breaking E2E security, effects throughput, and makes the entire setup complex and hard to manage.

In our proposed architecture we chose to use the full IPv6 network stack and standard Internet protocols on both of the networks. Because of the resource constrained devices and low power networks used for the IoT, adaptations have been proposed to the IPv6 standard, called 6LoWPAN [9], to make it more lightweight. When using 6LoWPAN there is still translation needed at the network border, but it is extremely lightweight and does not break E2E security. A 6LoWPAN Border Router (6BR) is used to connect the conventional IPv6 network to the 6LoWPAN network. In addition to processing 6LoWPAN header compression and fragmentation, 6BR also performs network interface translations, for example, from IEEE 802.15.4 to Ethernet.

In Fig. 1 we present how in our solution the SicsthSense cloud platform communicates end-to-end with a sensor nodes in the 6LoWPAN network. Packet is sent from the cloud service and the border router performs compression and fragmentation if necessary of the IP and UDP headers before entering the 6LoWPAN network. Here, it is also possible to perform compression of other protocols; we have already specified the 6LoWPAN compression of DTLS in Lithe [1]. The packet then goes from the 6BR to the sensor node with the particular destination address, through one or more hops. A response goes the same way back and the IP and other headers are then decompressed at the border router. The IEEE 802.15.4 is used at the Physical and Mac layer in a 6LoWPAN network, while on the conventional internet Ethernet or any other physical layer standard can be used.

In general, when selecting a transport layer protocol, there are two protocols to select form, TCP and UDP. In the low-power and lossy 6LoWPAN networks, more packet loss can be expected than in conventional networks and TCP does not function well in such environments, because of its congestion control and connection-oriented nature. Therefore UDP is mostly used in 6LoWPAN network. UDP provides unreliable packet transmission but additional reliability is instead added in the application layer using timers and retransmissions.

In SecureSense we use CoAP as an application layer protocol to enable web capabilities in the IoT. We can use the HTTP protocol, but it is both too heavyweight for IoT devices and is designed to be used with TCP. CoAP uses UDP as a transport protocol but still provides many of the same functionalities as HTTP provides.

The protocols mentioned before provide no security by default and therefore additional mechanisms are needed to secure the data transmission in SecureSense. One way to secure data communications is to do it on the link layer using IEEE 802.15.4 security within the 6LoWPAN networks. A different security solution is then needed when data leaves the 6LoWPAN network to the Internet. Also, IEEE 802.15.4 only provides security over one hop at a time through a 6LoWPAN network. We instead want to provide secure end-to-end transmission of data between a cloud platform and a sensor node. Therefore, the next possible solution is to secure all the IP packets sent to and from a cloud platform using IPsec on the IP layer. This solution is often used for Virtual Private Networks (VPNs) to send IP packets between virtual networks securely. This solution can though cause problems if changes are made to the IP layer of the OS as the IPsec implementation also needs to adapt to these changes. Another solution is to implement a custom security mechanism in the CoAP protocol itself on the application layer. HTTP achieves this by using a separate standardized protocol between the application and transport layers to secure the HTTP traffic. This solution is both easily deployable and upgradable because the security protocol can be independent of how the operating system implements the transport and network layer protocols. The protocol is responsible for setting up all security parameters needed between two parties using a handshake, before the application protocol can start communication with the other party. TLS is the protocol used for HTTP, but like HTTP it uses TCP. Datagram TLS is an alternative that is based on TLS but designed to be used with UDP instead of TCP. SecureSense therefore uses DTLS to secure all CoAP application data between a sensor node and a cloud platform. Here, there is also a low learning curve in understanding how to use secure CoAP with DTLS, because it is similar to securing HTTP with TLS, that is widely used today.

The protocol stack chosen for the SecureSense is in the end similar to what is normally used on the conventional Internet, with the main difference of using more lightweight protocols that also handle the packet loss in the lossy networks better. In Fig. 3, we see how data from the cloud platform is sent E2E through the network using all the protocols.

The border router only needs to inspect packets up to the network layer. It does not need to look at anything in the higher layers, unless it also needs to act as a firewall or these higher layers should also be compressed using 6LoWPAN compression techniques. We also see how the switch from Ethernet on the physical and link layer to the low power 2.4 GHz IEEE 802.15.4 standard is done at the border router. Translations between different physical and link layer protocols on the Internet is indeed done all the time in conventional routers and computers. Finally, we see how the additional 6LoWPAN layer is added before the IP layer to allow it to be compressed when transmission over the 6LoWPAN network. When a packet leaves a 6LoWPAN network to the Internet, all headers are restored to a normal full uncompressed headers.

After having the overall description of SicsthSense, we delve deeper and explain how the two extremes ends, resource-constrained sensor nodes and extremely powerful cloud platforms, are developed.

3.1. Cloud side

One the cloud side of SecureSense, we use SicsthSense, a cloud platform for the IoT, that we have previously developed. We enhance and integrate existing CoAP and a DTLS libraries, developed for standard computers, into SicsthSense. In Fig. 4, we see how SicsthSense processes packets it receives from the network and where we integrate the Californium [25] and Scandium[1] libraries into SecureSense. It also shows how all secure CoAP packets go through the DTLS protocol, while unsecured CoAP can still be used without DTLS. The platform does not accept DTLS packets containing CoAP application data until it is connected to the other peer by completing the DTLS handshake. SicsthSense also support HTTP and TLS. How TLS is handled for HTTP is very similar to DTLS, and therefore, it is not shown in Fig. 4. Because CoAP has similar capabilities to HTTP, many SicsthSense functions already available for HTTP can also be used with CoAP.

SicsthSense also has the possibility to poll data on a regular interval from a specific URL instead of relying on data being pushed to the platform. In Fig. 5 we see how SicsthSense prepares the secure CoAP client when a new polling URL is assigned to a resource. This is done to avoid the need to make a new handshake with the host on every poll and to make sure to close DTLS connection correctly when it is no longer being used. A new secure CoAP connection is only opened if new handshake cannot be avoided, otherwise the old already established session is resumed. SecureSense also makes sure that old secure CoAP connections are closed properly. When a

[1] https://github.com/eclipse/californium.scandium

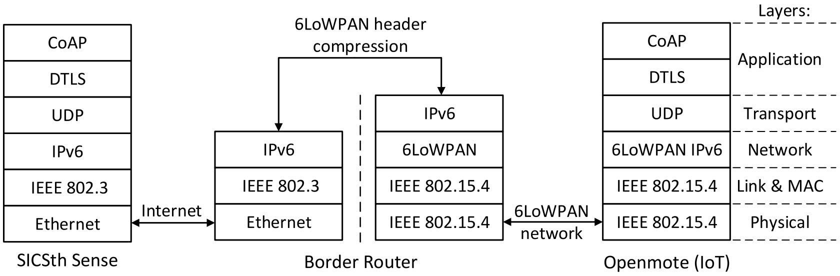

Fig. 3. The protocol stack used between the SicsthSense cloud platform and the OpenMote nodes on the 6LoWPAN network in SecureSense.

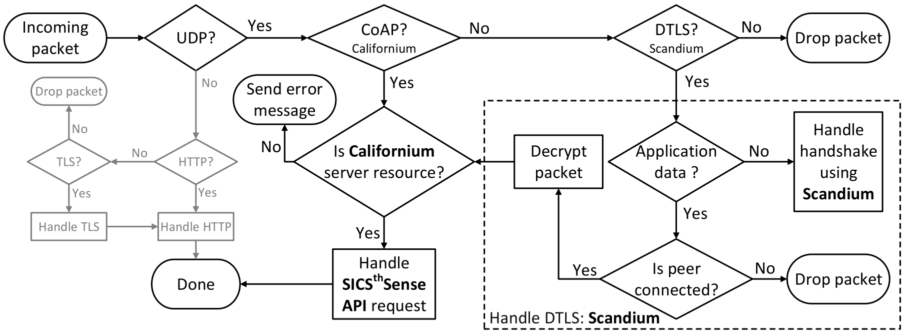

Fig. 4. Flow chart showing how SicsthSense processes incoming packets and libraries integration in SecureSense.

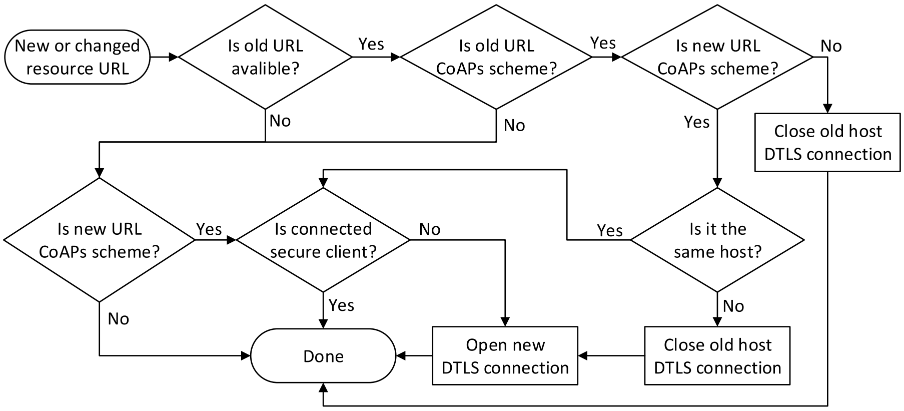

Fig. 5. Flow chart showing how SicsthSense opens new DTLS connection and/or reuses an old DTLS session when resource URL is changed.

poll timer expires, the correct method is selected based on the URL scheme for that resource and the response data is then returned to the SicsthSense parser that imports the data to the correct stream for the resource.

3.2. 6LoWPAN side

On the resource-constrained IoT side, a 6LoWPAN networks is formed and a sensor node, running IPv6, CoAP, DTLS, IEEE 802.15.4, etc., is connecting to SecureSense. On the 6LoWPAN side, we use Contiki, an operating system for the IoT. This sensor node can either be a CoAP client that starts the communication with the SicsthSense cloud platform, or a CoAP server that responds to requests form the SicsthSense cloud platform. Lightweight CoAP with DTLS have already been implemented for Contiki [1]. Our solution enhance the existing libraries to support RPK and full certificate-based cryptography. We also enhance and integrate the existing libraries to support our software-based ECC [13] and provide drivers to support hardware crypto of OpenMote [24]. In Fig. 6 we see how the DTLS is integrated into the procedure of sending CoAP messages in Contiki. We therefore are storing state about whether CoAP requests are secure or not, and then respond to them correctly depending on that state. This is identified by the incoming or outgoing ports used by the CoAP request. Therefore the security state of CoAP is identified by the port number used.

Though developing security protocols for standard computers is also hard, it is very challenging to provide these protocols for constrained devices. Also, the performance benchmarks for energy, throughput, processing resources, ROM/RAM, etc., are more important for battery-powered IoT nodes than conventional computers. Therefore, we provide implementation details for the 6LoWPAN side; also, most of the evaluations are also targeted for 6LoWPAN networks.

4. Implementation

In this section we present the SecureSense implementation and highlight our contributions.

4.1. Platforms and software

Contiki [26] is an open source operating system used for networked embedded devices with focus on low-power IoT devices.

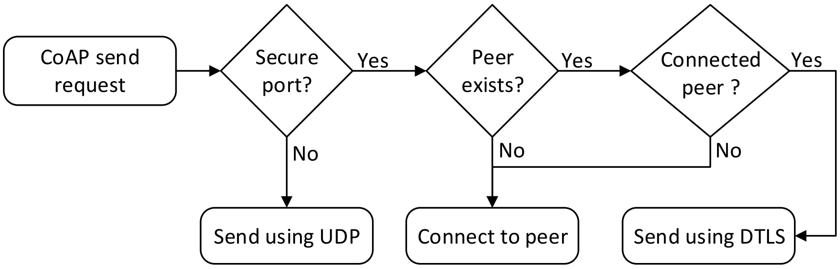

Fig. 6. Flow chart showing how CoAP in Contiki sends data with and without security.

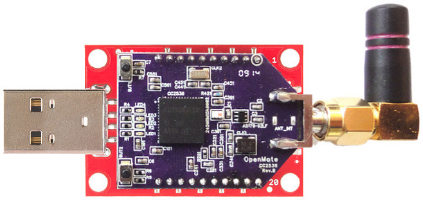

Fig. 7. OpenMote connected to a Xbee USB dongle.

Contiki has a small memory footprint because it uses event-driven kernel with multiple threading models on top of it. It has been deployed on many platforms and therefore also used with several different CPUs.

OpenMote, Fig. 7, is a new wireless node from OpenMote Technologies [24] that uses the CC2538 system on chip from Texas Instruments [27]. OpenMote was selected as it provides good features for development and testing. OpenMote has four LEDs, two buttons and an antenna connector for an external antenna. The CC2538 chip was chosen as it already has a good support available in Contiki. The CC2538 is a chip that has a powerful ARM Cortex-M3-based microcontroller that has a rated clock speed up to 32 MHz. It is combined with 32 KB on-chip RAM and 512 KB on-chip flash storage. Finally, it has an IEEE 802.15.4 radio that can be used for 6LoWPAN networks. The CC2538 chip can easily handle the Contiki operating system with enough resources left for security and other applications to be implemented. The chip has also option for powerful hardware accelerators for the following security cryptographic systems and standards: AES-128/256, SHA256, and ECC-128/256. This allows for both better performance and power savings, by freeing authentication and encryption tasks from the CPU.

tinyDTLS [28] is a library that provides simple and lightweight implementation of DTLS. It is targeted at simple embedded devices like wireless sensor nodes. We choose lightweight tinyDTLS for SecureSense rather than larger, more well-known libraries like OpenSSL.

Erbium (Er) is a low-power REST Engine developed for Contiki [29]. The engine is included in the Contiki operating system and has an embedded CoAP implementation based on the latest RFC [7]. DTLS has already been integrated into Erbium before using an older version of both tinyDTLS and Erbium [1]. Our solution upgrades this integration to the latest version of both tinyDTLS and Erbium, and adds additional support for RPK and certificate-based crypto.

Californium [25] is a Java based CoAP framework that is targeted at back-end services with a scalable architecture. It provides an API for RESTful Web services that support most CoAP features. The Californium library is actually divided into five sub-libraries and the core library only provides the CoAP function. *Scandium* is one of the libraries that implements DTLS 1.2 to secure the CoAP application data in Californium. It can be used with pre-shared keys, raw public keys or certificates. SecureSense uses this framework to add both CoAP and DTLS support to the SicsthSense cloud platform.

4.2. Integrating CoAP and DTLS in SicsthSense

SicsthSense already has a powerful HTTP API to be able both to push and pull data stored in the cloud. To be useful for the IoT the new CoAP protocol has been integrated to the platform using the Californium library. Because SicsthSense already has a HTTP implementation, it was used as a base for the CoAP addition. By using the Californium library a CoAP server was added to the SicsthSense cloud platform. To make the CoAP server provide the same functionality as the HTTP server, we implemented five new CoAP resource types for Users, Resources, Resource data, Streams and Stream data. The new SecureSense CoAP APIs allow users to create, update and pull data from the SicsthSense cloud platform, in a similar way as existing HTTP APIs are used.

SicsthSense is also able to pull data from specific HTTP addresses at a regular interval. Therefore a CoAP client support was also added to the cloud platform using Californium. The Californium CoAP client was interfaced with the SicsthSense poller. When a timer expiries to indicate that data should be pulled, the protocol scheme of the URI is checked, and then the correct function is selected based on the URI. If it is a CoAP URI, a new CoAP client is started (unless it has already been started), and the request is sent. The client then waits for the response of the request. This is similar to what had already been done to select between HTTP and HTTPs, but now two more options with CoAP and CoAPs have been added.

SicsthSense can easily be used by IoT devices without security. To make it possible to transfer data also securely from devices to the cloud, the Scandium library was used to add DTLS support to Californium. A new secure endpoint for CoAP was created to provide a way to use pre-shared keys and public/private keys using Java key stores containing certificates. The keys and certificates are loaded from these key stores using specific format and then provided to the client and server instances. In addition to configure these security parameters in SecureSense, it is also possible to configure things like retransmission count and timeout, that is necessary to configure for different IoT environments. One of the differences in our integration of CoAP compared with HTTP, is that when using the secure client, the CoAP's state is stored between each data polling, instead of creating new state for each pool. Therefore, there is no need for performing the DTLS handshake on every pool which saves resources accordingly.

4.3. Complementing tinyDTLS

The default ECC cryptographic library used in tinyDTLS is not open licensed, we therefore implement a BSD licensed ECC library for the Contiki [13] that only uses non-patented ECC curved specified in RFC6090. We adapt the functions used by the DTLS protocol for this new library. When generating or verifying signature, tinyDTLS already calculates the SHA256 of the data to be

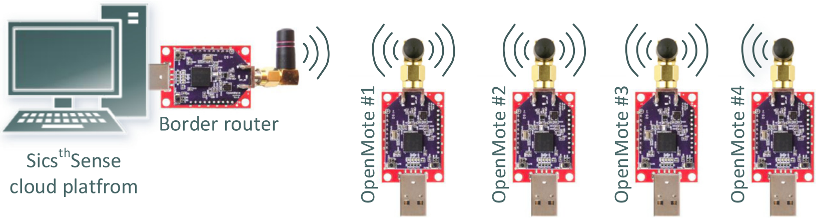

Fig. 8. The experimental setup consisting of up to four OpenMotes that communicate through single or multiple hops to a border router that is in turn connected to a standard computer running the SicsthSense cloud platform.

signed or verified, before calling any ECC function. The new ECC library is on the other hand expecting raw data to its functions and it then calculates the SHA1 hash of the data in this functions. Because of this and the fact that the ECC cipher suite uses SHA256, new functions were created that skip these SHA1 calculations and instead takes in a SHA256 value of a data directly as a parameter. Therefore, SHA hash calculations in the new ECC library in tinyDTLS are not needed. Having all the SHA hash calculations in the same place makes it also simpler to upgrade in the future.

We also add support for the CoAP RPK-based mode and the X.509-based certificates mode. We link these additions with our new ECC library and add support for SHA256 hash functions. We update tinyDTLS to support the new certificate and certificate verify messages. Furthermore, we change the DTLS hello messages, for both client and server, to have the correct extension that indicates the use of X.509 certificates instead of RPKs, as explained in RFC 7250 [30]. Our implementation can use both RPKs and certificates at the same time, and the actual mode is dynamically selected based on what both parties support. Last but not least, we update the certificate request message and add the ASN.1 formatted ID of the CA expected from the client.

4.4. Hardware crypto acceleration

We port the ECC cryptography drivers to Contiki for the CC2538 chip from Texas Instruments and integrate them with our ECC library in a way that makes it simple to select between the hardware and software implementation. When using the hardware acceleration, mathematical functions of the ECC library are replaced with hardware functions when compiling for the CC2538 chip. There are hardware accelerated functions for multiplication, comparison, modular and inverse modular for big integers. There are also hardware accelerated functions for addition and scalar multiplication of ECC points. A driver for AES and SHA256 hash hardware acceleration for the CC2538 chip has already been ported to Contiki and therefore it was also possible to implement hardware support in tinyDTLS for these cryptographic standards.

5. Evaluation

This paper follows the empirical methodology that begins with a concrete problem and is used to evaluate the impact of one particular variable of a phenomenon by keeping the other variables controlled. The basic problem we tackle is: can we use the government-grade certificate-based security in resource-constrained battery-powered IoT devices and what is the overhead of security when using all standardized protocols? This section answers this question.

We evaluate different security modes in CoAP, when using DTLS between a node in 6LoWPAN network and a cloud platform. Our experimental setup is shown in Fig. 8. It consists of four OpenMotes connected to the 6LoWPAN Border Router (6BR) (also an OpenMote) and forms the 6LoWPAN network, and a standard computer that hosts the SicsthSense cloud platform and is connected to the 6LoWPAN border router through the serial port. OpenMotes are wirelessly connected to each other and each of them runs the IoT stack shown in Fig. 3. To ensure that the messages are actually communicated through multiple nodes we implement a static routing path from leave nodes to the 6BR. We evaluate SecureSense with up to four hops in 6LoWPAN networks.

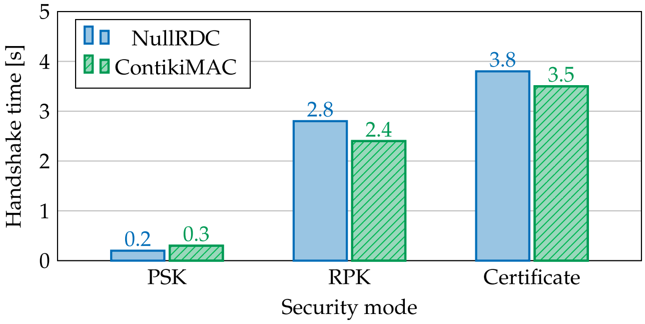

Fig. 9. Average DTLS handshake completion time between an OpenMote, running as the DTLS client, and the SicsthSense cloud platform, running as the DTLS server, for all the three modes of secure CoAP over single hop and with two different MAC protocols.

5.1. Handshake protocol in SecureSense — time overhead

5.1.1. DTLS handshake completion time

The most resource hungry operation in DTLS is the handshake process; we therefore measure the time each handshake process takes. The handshake completion time is measured on an OpenMote, using hardware accelerated cryptography for AES, ECC and SHA256, when communicating to a SicsthSense cloud platform through the 6BR. Two different radio duty cycling layers are used, ContikiMAC and NullMAC, The latter never turns off the radio.

Fig. 9 shows that ContikiMAC actually takes less time than NullMAC for the RPK and certificate-based CoAP security modes. This is counterintuitive as NullMAC is expected to send a packet immediately while ContikiMAC may have to wait for its peer to wake up before it can transmit. Also, packets may get lost in low-power and lossy 6LoWPAN networks. ContikiMAC has a built-in retransmission mechanism that may lead to faster retransmission of lost packets.

In this experiment, most of the time is spent on the resource-constrained OpenMote because the cloud platform is running on a more powerful hardware. The cloud can therefore process packets and run cryptographic functions much faster than OpenMote. In Fig. 9 we see that both the RPK and Certificate-based modes take much longer time than PSK; we show the cause of this behavior in the next section.

5.1.2. Individual handshake message processing time

Fig. 10 shows the time DTLS *server* take to process incoming messages and also the time it takes to prepare new outgoing

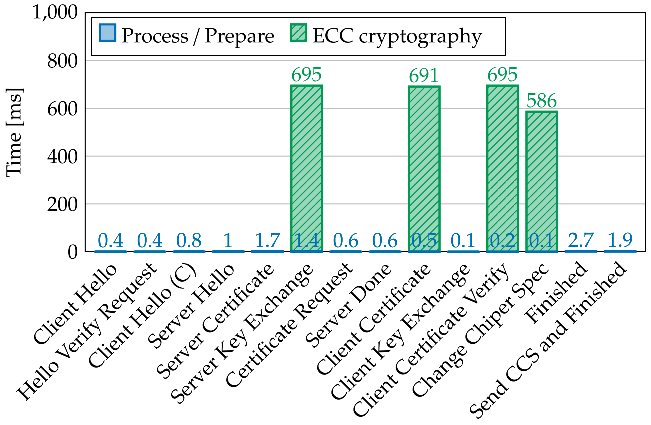

Fig. 10. Time needed to prepare and process individual handshake messages using the CoAP certificate-based security mode, when the DTLS *server* is running in a sensor node.

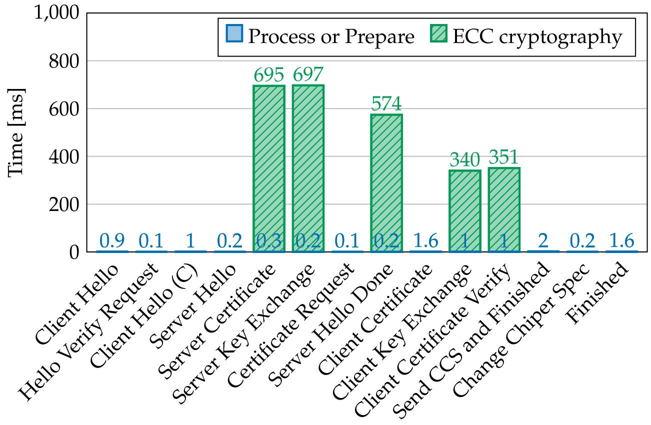

Fig. 12. Time needed to prepare and process individual handshake messages using the CoAP certificate-based security mode, when the DTLS *client* is running in a sensor node.

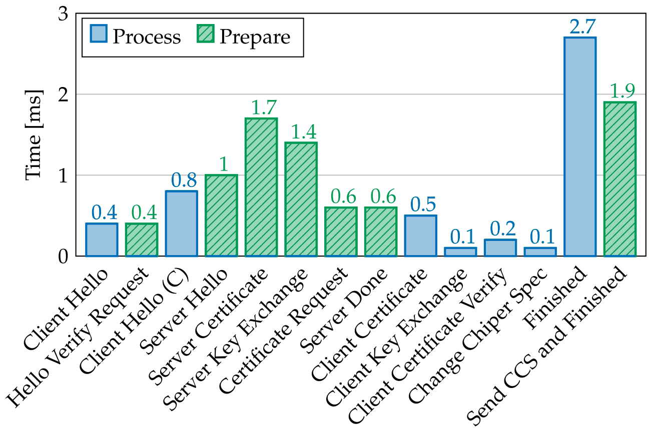

Fig. 11. Time needed to prepare and process individual handshake messages using the CoAP certificate-based security mode but without the ECC functions, when the DTLS *server* is running in a sensor node.

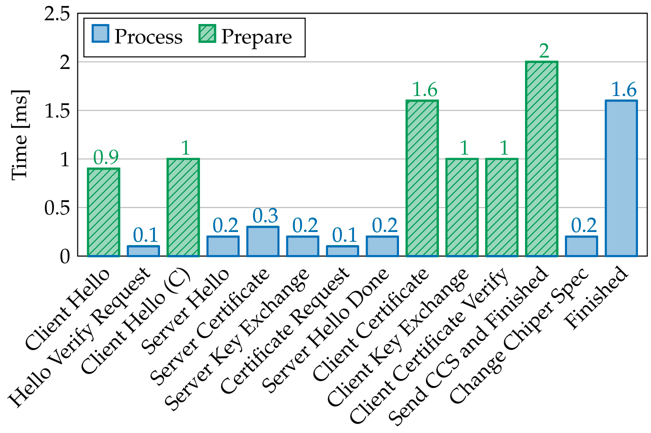

Fig. 13. Time needed to prepare and process individual handshake messages using the CoAP certificate-based security mode but without the ECC functions, when the DTLS *client* is running in a sensor node.

messages during the handshake in security mode 3 that uses X.509 certificates. This also includes the time it takes to process the record header and decrypt packets if needed, using the AES hardware crypto. These time measurements do not include the time it actually takes to send or receive the messages. For these measurements the certificate-based security mode is used, but the only major difference in RPK is that the ECC signature in the Certificate messages is not verified there. The results show that the ECC functions and the calculation of the Master Secret take majority of the overall time. In PSK security mode, no ECC functions are needed and also not all handshake messages are needed, hence the overall time is much shorter. These results also show that the additional and large-sized handshake messages for certificate-based mode do not primarily contribute to the longer handshake time when compared with the PSK mode.

In Fig. 10, ECC operations take most of the time and the time other messages take are suppressed and not clearly visible. Therefore, in Fig. 11 we show in more detail how much time it takes to process or prepare messages when the time of the ECC functions is skipped. Fig. 11 shows that most of the messages take a very short time, between 0.1 and 2.7 ms. The reason the Finished message takes longer time is because the handshake final hash calculations are required and the Finished messaged is sent encrypted using the new master secret key. We also see that the Server Certificate and Server Key Exchange messages take a longer time. That is because they include more data and therefore it takes longer time to prepare this data than for the other messages.

Figs. 12 and 13 also show the time individual handshake messages take when the DTLS *client* is running in a sensor node, with and without ECC operations, respectively. The results are very similar to the server results other than that the ECC functions are more spread among messages.

In the next section we investigate which ECC functions take the longest time and compare the impact of the hardware acceleration on the ECC functions.

5.1.3. *Impact of hardware and software crypto on time*

The time difference between using the software implementation of the ECC functions is compared to using the hardware accelerated implementation. Fig. 14 shows that the software implementation takes much longer time than using hardware acceleration or up to 18 times longer. Here only the client is shown as the server gives similar results. As these ECC functions take majority of the overall handshake time, we see how important it is to have them hardware accelerated. The ECC function that takes the longest time is the one that verifies an ECC signature. The function that calculates the master secret from its private key and the public keys of both parties takes the second longest time. The two other functions take similar time: the one that generates a public and private key pair and the one that creates a signature.

5.2. *Handshake protocol in SecureSense — Energy Overhead*

In the next experiments, we calculate the energy consumption of each DTLS handshake flight for all CoAP security modes, for both

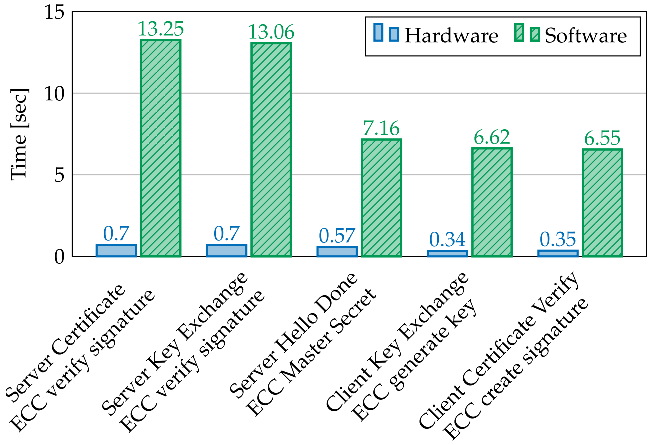

Fig. 14. Comparison of software and hardware ECC functions when handling handshake messages on the DTLS client side running on OpenMote.

Table 1
Power values for the CC2538 chip at 3 V.

Mode	Current	Power
CPU	13 mA	39 mW
LPM	0.6 mA	1.8 mW
Tx	28.3 mA	85 mW
Rx_idle	24 mA	72 mW
Rx_active	20 mA	60 mW

client and server. The ContikiMAC RDC driver is used to provide accurate energy consumption. As the time measurements only tell how long the radio was used to transmit or listen, non-dependent on what power mode the chip is in, it is assumed that the radio is used all the time when the chip is in CPU mode. If there is still radio time left after subtracting the CPU time, it is subtracted from the low power mode (*LPM*) time. This is only done if the CPU time is shorter than the total radio time. To make the calculation simpler, the chip is only allowed to go to the first low-power mode of the three available modes. The energy used is calculated from the time spent in each power state. The Energest and Powertrace applications for Contiki [31], used for the time measurements, provides the time in number of ticks (T) instead of seconds. The timer in Contiki produces 32.768 ticks per second (T_S) on OpenMote and therefore the energy is calculated with this formula:

$$Energy = Time \cdot Power = \left(\frac{T}{T_S}\right) \cdot (I \cdot V) .$$

The time when the chip is using the radio, is split in three different power modes. The first mode is when transmitting data (*Tx*), the second mode is when listening to the network (*Rx_idle*) and the last when actually receiving data from the network (*Rx_active*). The equation used for the energy consumption calculations depends on the difference between the CPU and radio time. Therefore if $(t_{CPU} - t_{radio}) > 0$:

$$\begin{aligned} Energy = {} & [(t_{CPU} - t_{radio}) \cdot P_{CPU}] + [t_{LPM} \cdot P_{LPM}] \\ & + [t_{Tx} \cdot P_{Tx}] + [t_{Rx\_active} \cdot P_{Rx\_active}] \\ & + [t_{Rx\_idle} \cdot P_{Rx\_idle}] . \end{aligned}$$

Otherwise if $(t_{CPU} - t_{radio}) \leq 0$:

$$\begin{aligned} Energy = {} & [(t_{LPM} - (t_{radio} - t_{CPU})) \cdot P_{LPM}] \\ & + [t_{Tx} \cdot P_{Tx}] + [t_{Rx\_active} \cdot P_{Rx\_active}] \\ & + [t_{Rx\_idle} \cdot P_{Rx\_idle}] . \end{aligned}$$

The CC2538 chip runs on 3-V and the values for the current that it draws in different power modes are provided in a document by Texas Instruments [27], and displayed in Table 1. The current for the *Tx* mode is provided both for radio power of 0- and 7-dBm. The OpenMote is configured to transmit at 3-dBm. The value used in the energy estimation is calculated from the two values provided, by assuming that the current changes linearly with the transmission power.

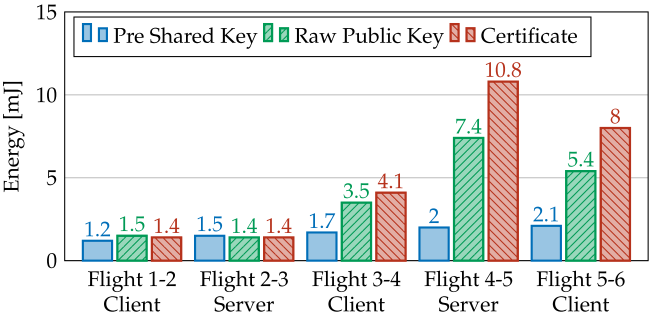

Fig. 15. Energy consumption for different handshake flights, both when a sensor nodes is running the DTLS client and the DTLS server, for all the CoAP security modes.

Because OpenMote only uses CPU mode and no radio during the handshake, the energy consumption can easily be calculated from the time measurements provided earlier using the simple energy formula. On the other hand when calculating the energy consumption of different flights, the time measurements cover the time from when the first message in a flight is sent and until the message is received from the next flight. Here the processing of messages for the next flight has not started. Therefore two flights are covered in one measurement. This is done to avoid time synchronization between client and server when each flight starts on one end and finishes on the other end. Because of this constraint, it is not possible to measure only the first flight received and the last flight sent by the server.

Fig. 15 shows that the longer time for the ECC operations (shown in Figs. 10 and 12) translates into higher energy consumption for ECC operation. We also see how the longer certificate messages, in flight 4 and 5, in the certificate-based mode affect the energy consumption compared to the RPK mode. For the same flights, more messages and data is transmitted in these two security modes than for the PSK mode; and the effect it has on the energy consumption can clearly be seen in Fig. 15. However, for other flights the difference is not significant between security modes.

5.3. DTLS record protocol in SecureSense — round trip time

We measure the round trip time for transmitting data payloads of different sizes using both simple UDP packets and DTLS Record layer-secured packets. Round trip time is the time it takes to send data from a client to a server and then sending the same data back from the server to the client. We also compared the difference between single-hop and multi-hop communications, using the setup shown in Fig. 8.

Fig. 16 shows the difference between sending data secured with DTLS and simple UDP through single-hop. Here, CSMA MAC driver is used for reliability. NullRDC is used instead of ContikiMAC to avoid time delays and packet loss, caused by ContikiMAC. Here both the software (SW) and hardware (HW) implementations for AES are used. The figure shows that the round-trip time is longer in the secure data transmission for all data sizes. As expected the hardware implementation takes a shorter time than the software implementation. When sending 64 bytes of data, the difference between UDP and DTLS is even bigger than for the other data sizes.

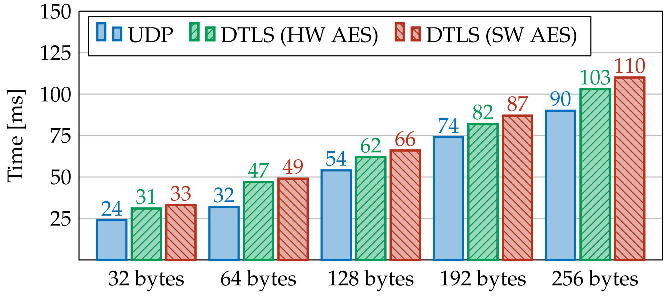

Fig. 16. Round Trip Time for SecureSense for sending insecure and DTLS-secured data over a single hop for different data sizes, with and without hardware crypto.

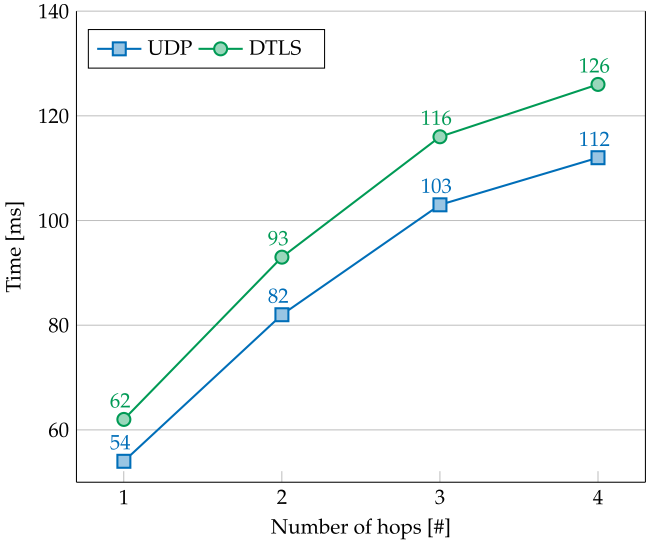

Fig. 17. Round Trip Time for SecureSense for sending insecure and DTLS-secured data of 128-bytes for different number of hops, with hardware crypto.

The reason is that the link layer frame gets larger than the 127 byte MTU of IEEE 802.15.4, for the secure transmission. Therefore the DTLS packet is fragmented on the 6LoWPAN layer, which results in longer round-trip time. Of the 127 bytes available for the frame, only 104 bytes can actually be used for the payload of the frame in Contiki, the rest is reserved for the frame header. Uncompressed IP and UDP headers are normally 48 bytes in total, while in Contiki they are compressed to 18 bytes using 6LoWPAN. Therefore when sending with UDP, 18 bytes are appended to the application data. On the other hand, when sending with DTLS, 47 bytes are appended to the application data, because of the additional DTLS header of 13 bytes and the encrypted application data with HMAC that take up the last 16 bytes. Because of this, only DTLS is fragmented, as its frame payload size is larger then 104 bytes.

In Fig. 16, we also see that when the application data size increases, the time difference between UDP and DTLS also becomes larger. This is because it takes longer time to encrypt larger packets, while the additional DTLS packet overhead is of static size and the number of 6LoWPAN fragments is the same for all sizes other than 64 bytes.

Fig. 17 shows how the round-trip time changes with the number of hops for 128 bytes of data. Here only the hardware AES implementation is used. For each hop the round-trip time increases faster for DTLS than UDP. That is because the DTLS packets are larger and therefore more data is forwarded at each hop, which results in longer processing time at every node.

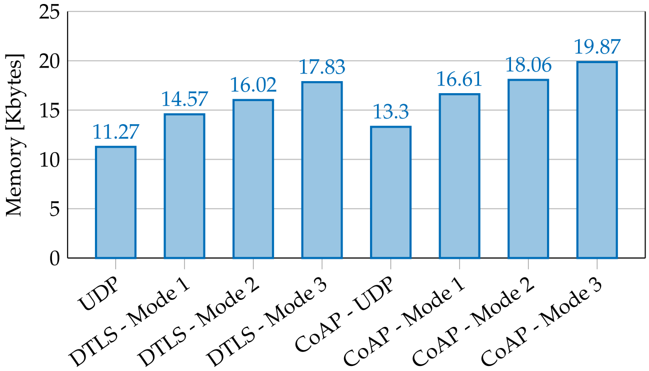

Fig. 18. Average static RAM memory usage for client and server using different configurations.

5.4. SecureSense storage overheard

Recall the sensor node in a 6LoWPAN network has limited RAM and ROM resources, we therefore measure these for SecureSense. We measure static RAM memory usage for different configurations. We use *arm-none-eabi-size* tool get the static memory consumption of a complied Contiki application. For simplification, the average value of both client and server is used, as both use similar amount of RAM. The stack size is configured to be 4096 bytes, while the implementation works with stack size as small as 3072 bytes. For simplification, the stack will not be included in the RAM usage. Both TCP and RPL are disabled in the applications to save memory space, but they use in total around 1200 bytes extra RAM, about half of that each.

Fig. 18 shows that when using CoAP in certificate-based mode, there is only around 8 Kbytes of RAM left, of the total 32 Kbytes, for other applications, when considering stack size of 4 Kbytes. However, this is enough for most IoT applications that run on a sensor node. Lower memory usage could be achieved by making the stack smaller, or through other optimizations of the OpenMote platform. The figure shows also that there is a steady increase in RAM usage when going from the PSK mode to the Certificate-based mode. When the CoAP PSK mode is added on top of UDP, the RAM increase is 3.3 Kbytes while for certificate-based mode the increase is 6.6 Kbytes. The difference here is because of the extra ECC functionality in the RPK and certificate-based modes. When adding the CoAP protocol, the RAM increase is around 2 Kbytes. When adding security to CoAP the increase is almost the same as when using DTLS with UDP. The increase in RAM usage, from the UDP application to using CoAP in certificate-based mode, is around 75%. Even with this increase it is possible to run additional applications when using any of the CoAP security modes.

5.5. SecureSense: Scalability and robustness

SecureSense is built upon standard protocols that are designed to scale for billions of IoT devices. Though the individual protocols are scalable, the scalability of the system as a whole is proportional to the capabilities of involved end hosts. The SecureSense implementation can handle multiple connections at a time as long as the hardware resources of the involved hosts allow. For instance, OpenMote with just 32 kB of RAM is able to run CoAP with full certificate-based mode. We have evaluated the SecureSense overhead in terms of energy, processing time and storage, which gives a clear indication of resource requirements in constrained devices. On the cloud-side, our SicsthSense implementation allows multiple active connections and its scalability is also directly proportional to the capabilities of the hosting device.

Our implementations of the involved SecureSense protocols (such as CoAP, DTLS, and IPv6) are standard compliant. We do not propose any new protocol but use existing standards and built an end-to-end cloud-connected IoT system, implement it in an IoT setup and evaluate its overhead. The security analysis of the SecureSense individual protocols is already discussed in the proposed standards [8,7,32] as well as in other existing literature such as [33,1,34], which holds true in our work. Though the individual chosen protocols are robust and scalable to billions of devices, implementation bugs are possible. Software security analysis of the Contiki OS and our implementations requires an extensive amount of work and is outside the scope of this paper. In an ongoing EU project [35], we are looking at the software security of the Contiki OS and the IoT protocols implemented in Contiki.

Evaluation Synthesis.

Our experimental evaluation seeks to characterize in detail the overhead of the implemented solution, notably determining (i) latencies to complete operations, (ii) energy consumption, and (iii) memory usage. They all depend on computation and communication overhead, and consequently the mode of operation of the implemented protocols and the involved cryptographic primitives; and the used platform design.

Our findings ascertain the feasibility and more so the practicality of the implemented solutions. They highlight the challenging, in terms of any of the three metrics, aspects, possibly stressing the small-footprint platforms we use, and are typically expected to be broadly deployed for IoT applications. Last but not least, they shed light on a gamut of implementation details and are comprehensive in their consideration of features.

In brief, the DTLS handshake is found to be challenging due to the public key cryptography involved, yet feasible, with latencies for the full process less than 4 s. As those transactions are infrequent, and symmetric key cryptography, vastly more efficient, for the bulk of communications, the scheme is viable. Using symmetric key cryptography results in a mild increase of the latencies as the data sizes increase.

Related, the use of cryptographic hardware for the public key (elliptic curve) operations is clearly beneficial reducing delays by more than order of magnitude. The power consumption measurements are correlated to the overheads due to cryptographic operations and transmission of security data, notably certificates; clearly higher for public key operations. The memory consumptions varies for different modes, with substantial increase due to security; yet not at an encumbering level.

6. Conclusions

Connecting everyday physical objects with the Internet and accessing and controlling them through any host on the Internet, such as computing cloud, expected to bring huge change in the way we live and work today. Security could become a nightmare in this transition if not enforced at the design phase of building this Internet of Things (IoT). We have presented SecureSense that adds security at the core of cloud-connected IoT. SecureSense uses standardized Internet protocols and provides secure E2E data communication directly between an IoT device and a cloud platform. We have integrated the IoT protocol CoAP and the security protocol DTLS into the SicsthSense cloud platform and also provided these protocols in 6LoWPAN networks, enabling all three security modes of CoAP.

We have provided a detailed empirical evaluation of SecureSense using real IoT hardware and a cloud platform. Our evaluations showed that though asymmetric cryptographic operations have the biggest impact on the performance of SecureSense, the overhead can be drastically reduced using off-the-shelf sensor nodes that come with hardware crypto. Our evaluations set the performance benchmarks for all the three security modes of CoAP and show that SecureSense is a viable E2E communication security solution for the cloud-connected IoT, in term of energy, time, and storage overhead. We conclude that it is possible to use the strong government-grade certificate-based security in battery-powered IoT devices (having just 32K of RAM), and the timing and energy overhead is acceptable for most IoT applications.

Acknowledgments

This research has partly been funded by the Strategic Innovation Program for Internet of Things (SIP-IoT), a joint venture of VINNOVA, Formas och Energimyndigheten; and partly by the EU H2020 project NobelGrid under grant no. 646184.

References

[1] S. Raza, H. Shafagh, K. Hewage, R. Hummen, T. Voigt, Lithe: Lightweight secure CoAP for the internet of things, IEEE Sensors J. 13 (10) (2013) 3711–3720.

[2] RISE SICS - sense.sics.se, SicsthSense Cloud Platform Suite, https://github.com/sics-iot/sicsthsense.

[3] Digi International, Digi Device Cloud (Etherios Device Cloud). http://www.digi.com/products/cloud/digi-device-cloud.

[4] LogMeIn Inc (LOGM), Xively enterprise IoT platform. https://xively.com/whats_xively/.

[5] L. Jiang, L. Da Xu, H. Cai, Z. Jiang, F. Bu, B. Xu, An IoT-oriented data storage framework in cloud computing platform, IEEE Trans. Ind. Inform. 10 (2) (2014) 1443–1451.

[6] Eclipse, Leshan, an OMA Lightweight M2M (LWM2M) implementation. https://github.com/eclipse/leshan.

[7] Z. Shelby, K. Hartke, C. Bormann, The Constrained Application Protocol (CoAP), RFC 7252, 2014. http://www.ietf.org/rfc/rfc7252.txt.

[8] E. Rescorla, N. Modadugu, Datagram Transport Layer Security Version 1.2 RFC 6347, 2012. http://www.ietf.org/rfc/rfc6347.txt.

[9] J. Hui, P. Thubert, Compression Format for IPv6 Datagrams over IEEE 802.15.4-Based Networks, RFC 6282. Request for Comments, IETF, Internet Engineering Task Force. 2011. http://www.ietf.org/rfc/rfc6282.txt.

[10] W. Hu, P. Corke, W.C. Shih, L. Overs, secfleck: A public key technology platform for wireless sensor networks, in: Wireless Sensor Networks, Springer, 2009, pp. 296–311.

[11] T. Kothmayr, C. Schmitt, W. Hu, M. Brunig, G. Carle, A DTLS based end-to-end security architecture for the Internet of Things with two-way authentication, in: Local Computer Networks Workshops (LCN Workshops), 2012 IEEE 37th Conference on, IEEE, 2012, pp. 956–963.

[12] Certicom Research, Standards for efficient cryptography, SEC 1: Elliptic Curve Cryptography, version 2.0, 2009.

[13] O.P. nol Piñol, S. Raza, J. Eriksson, T. Voigt, BSD-based elliptic curve cryptography for the open internet of things, in: The 7th IFIP International Conference on New Technologies, Mobility and Security, NTMS'15, Paris, France, 2015.

[14] M. Campagna, SEC 4: Elliptic Curve Qu-Vanstone Implicit Certificate Scheme (ECQV), Version 1.0, Certicom Research, 2013.

[15] S. Santesson, H. Tschofenig, Transport Layer Security (TLS) Cached Information Extension, Internet-Draft draft-ietf-tls-cached-info-19, IETF Secretariat, 2015.

[16] R. Hummen, H. Wirtz, J.H. Ziegeldorf, J. Hiller, K. Wehrle, Tailoring end-to-end IP security protocols to the Internet of Things, in: Network Protocols (ICNP), 2013 21st IEEE International Conference on, IEEE, 2013, pp. 1–10.

[17] J. Granjal, E. Monteiro, J. Sa Silva, End-to-end transport-layer security for Internet-integrated sensing applications with mutual and delegated ECC public-key authentication, in: IFIP Networking Conference, 2013, IEEE, 2013, pp. 1–9.

[18] S. Fouladgar, B. Mainaud, K. Masmoudi, H. Afifi, Tiny 3-TLS: A trust delegation protocol for wireless sensor networks, in: Security and Privacy in Ad-Hoc and Sensor Networks, Springer, 2006, pp. 32–42.

[19] R. Hummen, H. Shafagh, S. Raza, T. Voig, K. Wehrle, Delegation-based authentication and authorization for the IP-based internet of things, in: Sensing, Communication, and Networking (SECON), 2014 Eleventh Annual IEEE International Conference on, IEEE, 2014, pp. 284–292.

[20] I.E. Bagci, S. Raza, U. Roedig, T. Voigt, Fusion: coalesced confidential storage and communication framework for the IoT, Secur. Commun. Netw. (2015). http://dx.doi.org/10.1002/sec.1260.

[21] M. Tiloca, C. Gehrmann, L. Seitz, On improving resistance to denial of service and key provisioning scalability of the dtls handshake, Int. J. Inf. Security (2016) 1–21.

[22] S. Raza, L. Seitz, D. Sitenkov, G. Selander, S3K: Scalable security with symmetric KeysDTLS key establishment for the Internet of Things, IEEE Trans. Autom. Sci. Eng. 13 (3) (2016) 1270–1280.

[23] C. Bormann, M. Ersue, A. Keranen, Terminology for Constrained-Node Networks, RFC 7228, 2014. http://www.ietf.org/rfc/rfc7228.txt.

[24] P. Tuset-Peiró, X. Vilajosana, OpenMote Technologies. http://www.openmote.com.

[25] M. Kovatsch, M. Lanter, Z. Shelby, Californium: Scalable cloud services for the internet of things with CoAP, in: Internet of Things (IOT), 2014 International Conference on the, IEEE, 2014, pp. 1–6.

[26] A. Dunkels, O. Schmidt, N. Finne, J. Eriksson, F. Österlind, N. Tsiftes, M. Durvy, The Contiki OS: The Operating System for the Internet of Things, 2011.

[27] Texas Instruments, CC2538 Powerful Wireless Microcontroller System-On-Chip for 2.4-GHz IEEE 802.15.4, 6LoWPAN, and ZigBee® Applications. http://www.ti.com/lit/ds/swrs096d/swrs096d.pdf. (Accessed December 12 2016.

[28] O. Bergmann, TinyDTLS Software Library Implementation, TZI Uni Bremen. http://tinydtls.sourceforge.net.

[29] M. Kovatsch, S. Duquennoy, A. Dunkels, A low-power CoAP for Contiki, in: Mobile Adhoc and Sensor Systems (MASS), 2011 IEEE 8th International Conference on, IEEE, 2011, pp. 855–860.

[30] P. Wouters, H. Tschofenig, J. Gilmore, S. Weiler, T. Kivinen, Using Raw Public Keys in Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS), RFC 7250, 2014. http://www.ietf.org/rfc/rfc7250.txt.

[31] A. Dunkels, J. Eriksson, N. Finne, N. Tsiftes, Powertrace: Network-Level Power Profiling for Low-Power Wireless Networks, Swedish Institute of Computer Science, 2011.

[32] S. Deering, R. Hinden, Internet Protocol, Version 6 (IPv6) Specification, RFC 2460 (Draft Standard), updated by RFCs 5095, 5722, 5871, 6437, 6564, 6935, 6946, 7045, 7112, 1998. URL http://www.ietf.org/rfc/rfc2460.txt.

[33] Y. Sheffer, R. Holz, P. Saint-Andre, Summarizing Known Attacks on Transport Layer Security (TLS) and Datagram TLS (DTLS), RFC 7457 RFC 7457 (Informational), Request for Comments, IETF, Internet Engineering Task Force, 2015. http://www.ietf.org/rfc/rfc7457.txt.

[34] C.E. Caicedo, J.B. Joshi, S.R. Tuladhar, IPv6 security challenges, IEEE Comput. 42 (2) (2009) 36–42.

[35] Inria France (coordinator), H2020 Vessedia Project (2016–2019): Verification Engineering of Safety and Seuciry Critical Dynamic Industrial Applications.

Shahid Raza is the Director of Security Lab and an expert researcher at the RISE SICS Stockholm where he has been working since 2008. Shahid has completed his industrial Ph.D. from the Mälardalen University Västerås and SICS Stockholm in 2013. He also holds a Master of Science degree from KTH The Royal Institute of Technology, Stockholm. Raza's research interests include security issues in wireless sensor network in general and the Internet of Things (IoT) in particular.

Tómas þór Helgason is a researcher at RISE SICS working primarily on the Internet of Things security. Tómas received his B.Sc. degree in Electrical Engineering and Computer Science in 2013 from University of Iceland. He earned his M.Sc. degree in Electrical Engineering from KTH Royal Institute of Technology, Stockholm, Sweden in 2015 with focus on network services and security.

Panos Papadimitratos earned his Ph.D. degree from Cornell University, Ithaca, NY, in 2005. He then held positions at Virginia Tech, EPFL and PoliTo. Panos currently leads the Networked Systems Security Group at KTH, Stockholm, Sweden. His research agenda includes a gamut of security and privacy problems, with emphasis on wireless networks. He was the program co-chair for the ACM WiSec and the TRUST conferences and will serve as the general chair of the WISec (2018) and PETS (2019) conferences. Panos is a member of the Young Academy of Europe.

Thiemo Voigt manages the Networked Embedded Systems group at the Swedish Institute of Computer Science. He shares his time between SICS and Uppsala University where he is Professor for Wireless Sensor Networks. His main interests are in networking and system issues in wireless sensor networks and the Internet of Things.