INDIGO: Secure CoAP for Smartphones

Enabling E2E Secure Communication in the 6IoT

Daniele Trabalza[1], Shahid Raza[1], and Thiemo Voigt[1,2]

[1] Swedish Institute of Computer Science, Stockholm, Sweden
[2] Department of Information Technology, Uppsala University, Sweden
{daniele,shahid,thiemo}@sics.se

Abstract. With the inception of 6LoWPAN, it is possible to connect wireless sensor networks (WSN) and smart objects with the Internet using the IPv6 protocol, hence forming the IPv6-based Internet of Things (6IoT). Since the links in the 6IoT are lossy, UDP rather than TCP is mostly used for the communication between things. For the same reason, CoAP, a connection-less variant of HTTP, is being standardized as the web protocol for the 6IoT. Due to the sensitivity of the potential applications and presence of humans in the loop, End-to-End (E2E) security between constrained devices and hosts on Internet is one of the main requirements in the 6IoT. Secure CoAP (CoAPs) is used to provide end-to-end security in the CoAP-based 6IoT.

Smartphones with sensing capabilities, direct human interaction, Internet connectivity, and relatively powerful processing and storage capabilities, are going to be an integral part of the 6IoT. In this paper we design, implement, and evaluate CoAPs for Android powered smartphones. We call our CoAPs INDIGO. To the best of our knowledge this is the first work that provides CoAPs support in smartphones. We implement and evaluate all cryptographic cipher suites proposed in the CoAP protocol, including the certificate-based authentication using Ecliptic Curve Cryptography (ECC). We also present novel application scenarios that are enabled by INDIGO on smartphones.

1 Introduction

The Internet of Things (IoT) or strictly speaking IPv6-connected IoT (6IoT) is a hybrid network of IPv6 over Low-power Wireless Personal Area Networks (6LoWPANs) [1] or IP-connected WSNs, and the conventional Internet. At one end the devices in the 6IoT are too resource constrained, for example wireless sensors, and on the other end the device can be any Internet connected device such as a smartphone, a PC or a cloud. Smartphones are becoming an integral part of the 6IoT mainly because they are the most convenient and readily available tools that humans can use to connect to the 6IoT in order to interact with smart objects or *things*.

The Constrained Application Protocol (CoAP) [2] is a new web protocol designed to complement the 6IoT with web capabilities. Smart objects in the 6IoT are resource constrained and form low-power and lossy networks. As the links are lossy it is hard to maintain a reliable connection using the Transmission Control Protocol (TCP). Hence, the User Datagram Protocol (UDP) is mostly used in the 6IoT. The Hypertext Transfer Protocol (HTTP), however, cannot work over UDP and therefore a new web protocol is required. Towards this end, CoAP is being standardized. CoAP works over UDP and is a connection-less protocol where reliability is achieved with acknowledgements. CoAP proposes to use Datagram Transport Layer Security (DTLS) as a security protocol for automatic key management, data confidentiality, and data integrity. Similar to secure HTTP (HTTPS), CoAP with DTLS support is termed as secure CoAP (CoAPs). CoAPs is being standardized to secure the future 6IoT. The 6IoT devices can securely access web resources using CoAPs as: **coaps://myIP:port/res.xml**, similar to **https://myIP:port/res.xml**

In order to securely access these resources and interact with smart objects it is important to have CoAPs capabilities on the Internet connected devices. In this paper we design and implement the CoAPs protocol, that we call INDIGO, for Android smartphones. Having CoAPs capabilities in smartphones enables numerous applications where smartphones can be securely used in the context of the 6IoT; we discuss these applications in Section 4. We also evaluate the overhead of INDIGO's Handshake and Record protocols and the additional processing and communication requirements of CoAPs.

The next section gives an overview of the technologies used in INDIGO. Section 3 describes the capabilities of INDIGO and details its client and server components. We discuss applications enabled by INDIGO in Section 4. In Section 5 we detail INDIGO's implementation for the Android OS. Section 6 presents our performance evaluation of INDIGO. We discuss related work in Section 7 and conclude the paper in Section 8.

2 Background

Here we discuss the technologies involved in the development of INDIGO.

2.1 IPv6-Connected Internet of Things (6IoT)

The IPv6-connected Internet of Things (6IoT) is a heterogeneous network of tiny devices (*things*) that sense the physical world. The things usually form a WSN or 6LoWPAN network [1] that is connected to the Internet through the Internet Protocol version 6 (IPv6). Things in WSNs/6LoWPANs are resource constrained low power devices. Networks of such devices are also called Low-power and Lossy Networks (LLNs). The things in the 6IoT can be any electronic appliance, smartphone, a standard computer, etc. This heterogeneity in the 6IoT makes it much more challenging to securely connect all 6IoT devices unless standardized protocols exist. The Internet Engineering Task Force (IETF) is actively working on the standardization of 6IoT technologies and has already standardized protocols such as CoAP [2], 6LoWPAN [1] [3], RPL [4].

Due to the lossy links among constrained devices in the 6IoT it is hard to maintain a reliable connection using TCP. Therefore UDP is mostly used in LLNs. The connection-oriented web protocol HTTP cannot work over UDP. Therefore a new protocol is needed and hence CoAP has been designed. To enable seamless End-to-End (E2E) communication and security all devices in the 6IoT should understand each others language. In the 6IoT 6LoWPAN Border Routers (6BRs)/sink can be used between 6LoWPANs/WSNs and the Internet to perform protocol conversion between the two realms. A 6BR may implement a proxy that can convert the 6IoT technologies, such as CoAP, to Internet technologies, for example HTTP. However, such proxies may not ensure E2E security; for example, 6BR proxies cannot ensure E2E security if Transport Layer Security (TLS) is used on the Internet side and DTLS is used in the 6LoWPAN network.

2.2 DTLS and CoAP

DTLS [5] is a variant of the TLS protocol that is designed to run over UDP. DTLS is a mandatory part of CoAPs. It borrows the Handshake and Record protocols from TLS. The Record protocol encapsulates and secures other protocols, such as the *Handshake*, *Alert*, *ChangeCipherSpec*, and application data. Due to the unreliability of the UDP protocol, DTLS incorporates mechanisms to retransmit and assemble packets. The DTLS Handshake protocol is a complex process and exchanges different messages to negotiate the cryptographic cipher suites, compression methods, security keys, certificates, etc. Once the Handshake protocol is completed the application can send secure messages using the Record protocol and negotiated keys and cipher suites. The Record protocol ensures the freshness, integrity, and confidentiality of the application data.

The Constrained Application Protocol (CoAP) [2] is a new web protocol. It is a subset of the HTTP protocol optimized for Machine-to-Machine (M2M) applications running on resource-constrained nodes. CoAP works over the UDP protocol where message exchanges are asynchronous, in contrast to the HTTP protocol that sends synchronous messages. CoAP borrows HTTP features such as Uniform Resource Identifiers (URIs) but uses packed binary representations, and uses only GET, POST, PUT and DELETE messages in a request/response model. The reliability in CoAP is achieved through *confirmable* messages that use acknowledgments. Secure CoAP (CoAPs) uses DTLS to secure CoAP messages. If a URI specifies the CoAPs protocol ($coaps : //myIP : port/resource.xml$), DTLS protocol processing is performed before the CoAP message is passed to the UDP layer. At the receiving end, the DTLS protocol is placed between the UDP and CoAP layers.

2.3 Android-Enabled Smartphones

Android is an open source operating system that is designed primarily for smartphones. Android-powered smartphones are abundant in number, available in different cost range, and actively in production by different vendors. Due to wider range and availability, smartphones in general and Android-powered phones in particular have a huge potential to be used in the 6IoT; we discuss potential applications in Section 4. Different CoAP implementations exist for Android phones to enable communication with WSNs/6LoWPANs. Also, there exist DTLS implementations in the C programming language. However, to the best of our knowledge no DTLS/CoAPs implementations exist for an Android device nor does there exist full CoAPs support for any platform. Android applications are developed using Java APIs for Android. Java Native Interface (JNI) that enables the integration of C language code in Java could be used in Android; however, it increases the complexity and decreases the performance of Android applications.

3 INDIGO: CoAPs for the Android OS

We present INDIGO, a CoAPs support for the Android OS. In this section we elaborate the capabilities of INDIGO and the description of INDIGO client and server.

3.1 INDIGO Capabilities

We implement the full DTLS protocol with Record, Alert and Handshake protocols, and integrate it with the CoAP protocol. INDIGO is a complete client/server system that enables CoAPs support in the Android OS and can also be used in standard computers. INDIGO supports all cryptographic cipher suites which are standardized in the CoAP protocol [2]. Recall that the CoAPs protocol is a CoAP protocol secured with DTLS. The DTLS protocol in INDIGO is standard compliant [5] and supports certificate-based mutual authentication between DTLS client and server. INDIGO has following capabilities:

– Applications can seamlessly send/receive reliable and unreliable CoAP or CoAPs messages; INDIGO exports these capabilities to the applications and differentiates between CoAP and CoAPs. For example, to securely access a dummy resource *stockholm_temp.xml* an application can use *coaps://myIP:port/stockholm_temp.xml*, and the underlying INDIGO secures the communication links and performs necessary tasks which are described in Section 3.2 and 3.3.
– INDIGO can act as a CoAPs client or a CoAPs server on an Android device.
– It supports the standardized DTLS Handshake protocol with retransmission mechanisms when a client or server does not receive the intended response messages within timeouts.
– The DTLS Handshake protocol supports certificate-based mutual authentication and uses the Elliptic Curve Digital Signature Algorithm (ECDSA) for asymmetric encryption/digital signature. The key exchange protocols supported during the Handshake protocol are Pre-shared Key (PSK), Elliptic Curve Diffie Hellman (ECDH), and Elliptic Curve Diffie Hellman Ephemeral (ECDHE). ECDH has a fixed DH key which means that one side during the handshake process does not change the key for consecutive handshakes.

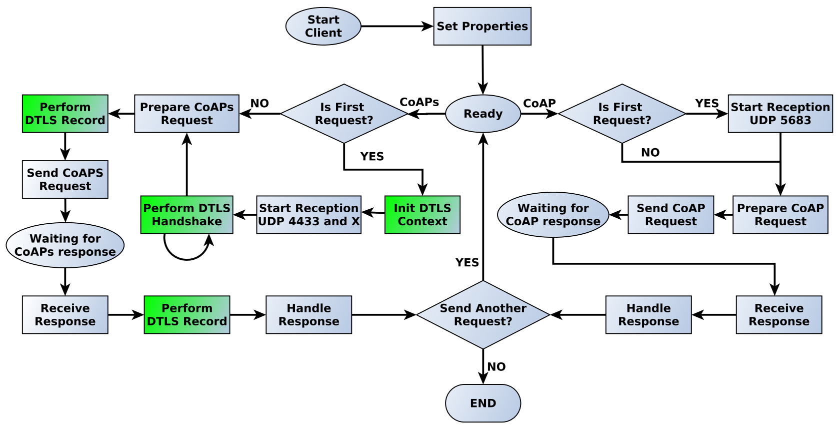


Fig. 1. States and flows when an application uses INDIGO Client to send/receive CoAP or CoAPs requests

On the other hand, Ephemeral mode generates distinct Diffie Hellman key for every handshake which ultimately provides forward secrecy; however, it is less efficient than ECDH as it requires more cryptographic operations.

- The DTLS Record protocol supports Counter with Cipher Block Chaining MAC (CCM) mode for confidentiality and integrity of CoAP messages. For the confidentiality of CoAP messages, the Advanced Encryption Standard (AES) with key size of 128*bits* (AES-128) or with 256*bits* (AES-256) can be used in Counter mode in CCM; and for integrity, the Cipher Block Chaining (CBC) mode in CCM is used with the same AES-128 or AES-256 protocol where the most significant 64*bits* of the last encrypted block are used as Message Authentication Code (MAC).
- SHA-256 is supported and used to generate or verify the hashes of all DTLS messages as specified in the DTLS protocol. Also, SHA1 is used in the CertificateVerify message to calculate and verify the hash of all prior messages. INDIGO does not include ClientHello and HelloVerifyRequest messages in the hash calculation as specified in the DTLS protocol.
- The above capabilities make INDIGO compliant to the CoAP protocol [2] as it supports the following mandatory cipher suites,
  `TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8` and
  `TLS_PSK_WITH_AES_128_CCM_8`.

3.2 INDIGO Client

An application on one machine can establish a secure connection with a remote application, and can act as an INDIGO client or server. When INDIGO runs as a client it can be used to request CoAP or CoAPs resources; Figure 1 shows the complete flow and different states of the INDIGO client. As a client, INDIGO starts with setting necessary properties for CoAP and DTLS such as port number

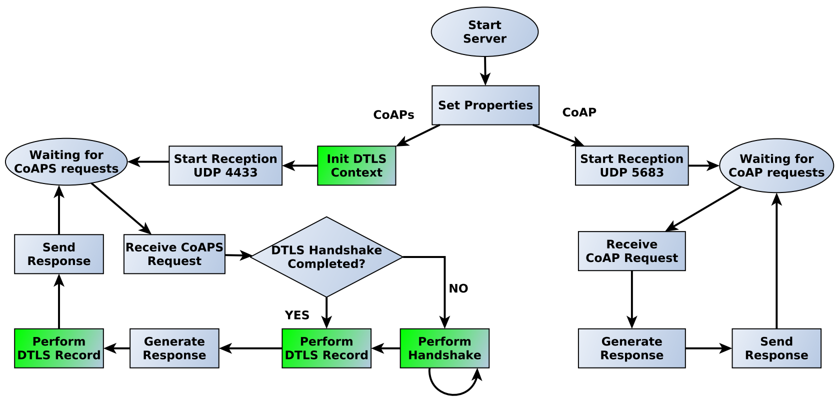


Fig. 2. States and flows when an application uses INDIGO Server to receive and respond to CoAP or CoAPs requests

and timeout. Then based on the protocol in the URI it either starts CoAP or CoAPs processing. If an application decides to use CoAP for a request, INDIGO prepares the CoAP request, sends it, and goes to waiting state to wait for the CoAP response. If it is the first CoAP request since the client started, INDIGO starts the reception process for the CoAP response. Upon reception of the CoAP response INDIGO processes it, returns the result to the application, and lets the application decide whether it wants to send another CoAP or CoAPs request.

If an application decides to use the CoAPs protocol for a request INDIGO creates a CoAPs request, performs DTLS Record protocol processing to add confidentiality and integrity to the message, sends the CoAPs request, and waits for the response. If it is the first CoAPs request since the client started, INDIGO initializes the DTLS context, starts the reception process for the Handshake responses at port 4433 and for the CoAPs responses at port IANA_TBD_PORT, and performs the complete DTLS Handshake protocol. The CoAPs port is not yet assigned by IANA [2]; the actual CoAPs port should later replace IANA_TBD_PORT. Upon reception of the CoAPs response INDIGO verifies the integrity of the messages and decrypts the confidential data using the Record protocol.

3.3 INDIGO Server

An application can use INDIGO as a server to respond to CoAP or CoAPs requests; Figure 2 shows the complete flow and different states of the INDIGO server. To this end INDIGO sets necessary properties for CoAP and DTLS such as CoAP/CoAPs port number, timeout, DTLS port, if it needs to act as a client or sever; and initializes the DTLS context in order to retrieve the certificates and keys used for encryption and authentication. It binds CoAP to port 5683, CoAPs to port IANA_TBD_PORT, and DTLS to port 4433. Then it goes to waiting state to listen for CoAP/CoAPs requests. If it receives a CoAP request it simply retrieves the requested resources, generates the corresponding response, and sends it to the client. Then it listens for new CoAP requests. On the other hand if the INDIGO server receives a CoAPs request it ensures that the handshake performed earlier is still valid and/or performs a full handshake with the client by sending a *Hello Request* message [5] [6]. If needed, it performs DTLS Record protocol processing to decrypt and verify the CoAPs request, generates the response and sends it to the client. Then it listens for new CoAPs requests.

4 Applications of INDIGO

Current smartphones are versatile devices with processing and storage capabilities though slightly less but similar to standard computers; a smartphone is a mini computer. However, unique features such as physical portability, close human interaction, sensing capabilities with having accelerometer, proximity, gyroscope, camera, sound, compass, GPS, Wi-Fi, GPRS/3G/4G, Bluetooth, even Near Field Communication (NFC), enable smartphones to be used in an enormous number of applications and scenarios. In the context of 6IoT a smartphone can be used as a sensing device, a 6LoWPAN Border Router (6BR), or an actuator or controller. Of course it can also be used as a typical computer to access/provide data to other Internet hosts and WSNs/6LoWPANs.

Smartphone as Sensor. In urban areas smartphones are the enabling technology for opportunistic and participatory urban sensing [7][8]. Due to the large availability and vast capabilities smartphones can be used in traffic congestion control and management, security and emergency services, access control, NFC payments, location detection, etc. Though in all these scenarios smartphones are the actual sensing devices they may complement these applications with the input from typical sensors that have limited resources. Recall that the 6IoT is an extremely heterogeneous network where many devices are resource constrained and expected to use standardized CoAP/CoAPs protocol. Therefore, it is important that all devices, in order to communicate and provide E2E security, should use the same secure protocol.

Smartphone as 6BR or Sink. We envision it beneficial to use a smartphone as a 6BR to connect 6LoWPAN networks or as sink to connect WSNs with the Internet especially when there is no wired infrastructure. In developing countries such as Pakistan and India the mobile coverage spans to more areas than the wired Internet access. Also, in remote areas such as forest, glaciers, deserts, mountains, etc. there is no wired Internet. Android-enabled smartphones have quite cheap models available that can be used as a 6BR in the 6IoT setup or a sink in traditional sensor networks to connect 6LoWPANs/WSNs deployed in the remote locations to the back-end servers. The sensors in 6LoWPAN networks can establish E2E secure connections with smartphones using the CoAPs protocol; or they can even establish these secure connections with the hosts on the Internet through the smartphone. In that case the phone only acts as a 6BR but cannot see the content of messages. The potential applications where a smartphone can be used as 6BR/router can be patients surveillance and monitoring where doctors can use smartphones to get latest patient information from the body area sensors network, military applications, flood monitoring and response system, glacier monitoring in the Himalayas region, animal tracking and monitoring, etc.

Smartphone as Actuator and/or Controller. The typical use of smartphones in the 6IoT context is to act as an actuator and/or controller. For example, in a smart home application a smartphone can be used to turn on and off thermostats or any other appliances. It can be used as interface to display and/or manage electricity consumption in smart grid/meter/plug applications, and in any 6IoT application where human interaction is needed or make it more useful smartphones are a valid solution. There are already sensor devices available such as *ube*'s smart devices[1] which use the Android OS as the main operating system in the sensor. *ube* uses a 32-bit ARM processor that runs the Android OS with a full IP stack and communicates over Wi-Fi by connecting to a local wireless router. They use smartphones as an integral part of the whole system which controls and actuates the sensors.

In all of these applications and scenarios security is one of the main requirements; imagine if someone could turn on your thermostat or can turn off your lightening system.

5 Implementation

INDIGO provides DTLS client and server implementations in the Android/Java programming language and integrates them with the CoAP protocol. Currently INDIGO borrows CoAP protocol capabilities from Californium[2]- a Java implementation of CoAP; and with little effort it can be integrated with any CoAP implementation favorably in Java. INDIGO's DTLS implementation is self-containing and designed as a standalone module that can be used by different applications even without CoAP. However, as our focus is to enable secure CoAP (CoAPs) we also provide a full integrated (CoAP + DTLS) system and use it in our evaluation. INDIGO's DTLS module communicates with the CoAP protocol through Java's publish-subscribe pattern using multiple threads which allows *simultaneous* and asynchronous operations. In this way more applications can use the same implementation by registering as *observer* when required. If the application desires a synchronous operation it is always possible to serialize the subscriber's method.

Java Cryptography Extension (JCE) APIs in plain Java neither provide DTLS APIs nor full certificate-based cryptography. Therefore, in addition to JCE, we also use cryptographic APIs from SpongyCastle[3] to develop INDIGO. We implement the full DTLS protocol from scratch; however, we borrow AES, SHA, and ECC APIs from JCE. We also develop CCM_8 mode from scratch. We plan

[1] `http://www.myube.co/`
[2] `http://people.inf.ethz.ch/mkovatsc/californium.php`
[3] `https://github.com/rtyley/spongycastle`

Table 1. Total time of INDIGO Handshake protocol with CPU time of individual Handshake flights inside a smartphone

Smartphone as DTLS client			**Smartphone as DTLS server**		
Process	Avg Time (ms)	Std Dev	Process	Avg Time (ms)	Std Dev
Full Handshake	3387.2	393.3	Full Handshake	4881.4	489.6
Flight 1	20.6	6.0	Flight 2	2.4	21.0
Flight 3	1.0	0	Flight 4	698.6	40.3
Flight 5	300.4	24	Flight 6	3.2	1.2

to publish INDIGO as open source. Though INDIGO is designed primarily for the Android OS it can be used in standard computers. In our evaluation we run INDIGO in both a PC and in a smartphone.

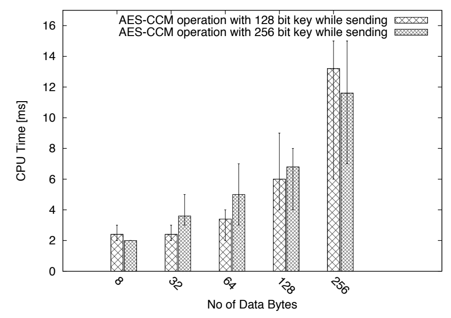
6 Evaluation

In this section we evaluate the overhead of the Handshake and Record protocols in INDIGO comparing the CPU usage of different cryptographic algorithms used in the DTLS Record protocol, total time of the Handshake protocol and the CPU time of individual flights in Handshake, and system-wide round trip times of CoAP and CoAPs requests. We cannot compare our CoAPs implementation against other solutions since there are no other CoAPs implementations available.

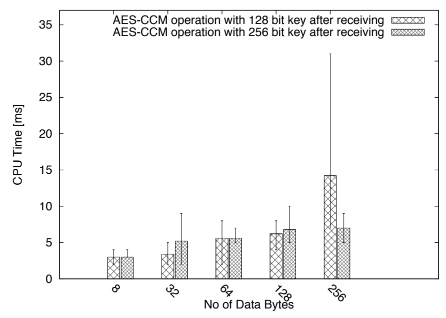
6.1 DTLS Handshake Protocol Overhead

Unlike the Record protocol, the Handshake protocol is not performed every time a client or a server wants to exchange messages. Communication end points perform the full Handshake protocol only when a new UDP connection is established or when a client or server wants to renegotiate an already established connection.

In our first experiment we measure the time required to perform a full Handshake protocol *(i)* when a smartphone acts as an INDIGO client and *(ii)* when a smartphone acts as an INDIGO server. The experimental setup consists of a smartphone and a standard computer connected through Wi-Fi. One device acts as a client and the other as server. Table 1 shows the total time spent on the Handshake when the smartphone acts as a client and a server. The total Handshake time in both cases should be equal as the same operations (total flights) are performed. However, it differs because the standard computer we use for evaluation is more powerful than the smartphone (Google Nexus with Android 4.0). The total time is effected by unpredictable Wi-Fi and Android internal process management and CPU scheduling. In order to get clearer picture of the actual *Handshake* protocol overhead, we measure the overhead of each Handshake flight [5] when it is performed inside a smartphone. Table 1 shows that the flight 4 and 5 are processing intensive flights; these flights consist of key exchange protocols and perform asymmetric cryptographic operations.



(a) AES-CCM operations with 128*bits* key performed before sending CoAPs message.



(b) AES-CCM operations with 256*bits* key performed after receiving CoAPs message.

Fig. 3. The cryptographic operations in the Record Protocol show that different key sizes have similar overhead

6.2 Cryptographic Overhead in DTLS Record Protocol

The Record protocol ensures the confidentiality and integrity of application data using AES in CCM mode. The overhead of AES CCM mode is added in each datagram on both the sending and receiving ends. Hence it is important to know how much time a CPU spends on processing cryptographic operations. We use different key sizes. As shown in Figure 3 the overhead of AES with different key sizes is almost same considering that there is high standard deviation and smartphones are not resource constrained devices.

6.3 Round Trip Time for CoAPs Messages

Once the secure connection is established by the Handshake, nodes only execute the Record protocol to secure the subsequent messages. In this experiment

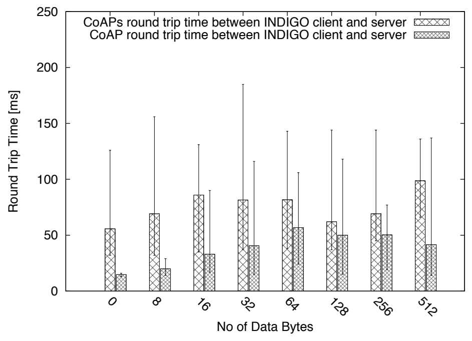


Fig. 4. The round trip time for CoAPs secured with the Record protocol is, as expected, higher than for the CoAP protocol

we measure the round trip time (RTT) which is the time from the creation of a CoAP/CoAPs request until the CoAP/CoAPs response is received and processed. The intermediate steps are shown in Figure 1 and 2. Figure 4 shows the RTT for a CoAPs request and compares it with a CoAP request. Recall that the high standard deviation is because of unpredictable Wi-Fi and Android internal process scheduling as many processes run inside an Android device.

7 Related Work

We are not the first to argue that the E2E security is necessary for the 6IoT, but we are the first to enable secure CoAPs communication between the constrained devices in the 6LoWPANS and smartphones. To the best of our knowledge currently there is no CoAPs support available for any platform. However, there are CoAP implementations available for different platforms such as Californium, jCoAP[4] and Earbium[5]. Also, DTLS support is available in the C language which is provided by OpenSSL[6], CyaSSL[6] GnuTLS[6]. In constrained devices CoAPs can be provided, for example, by integrating Earbium[5] and DTLS implementation in OpenSSL.

Brachmann et al. [9] elaborate the practical security issues in CoAP/6LoWPAN networks and the feasibility to use DTLS for E2E security and secure multicast. Kothmayr et al. [10] investigate the use of Trusted Platform Module (TPM) to provide DTLS in 6LoWPAN networks where they rely on hardware support for the RSA algorithm. We have previously presented solutions to compress DTLS to make it feasible for constrained devices [11]. Communication in the 6IoT can be secured at lower layers by, for example, using IPsec [12] or 802.15.4 security [13].

8 Conclusion

In order to securely connect and integrate smartphones in the 6IoT it is necessary to use standardized security solutions. CoAPs, the standardized secure web protocol for the 6IoT, is not yet available in smartphones. Towards this end, we have designed, implemented, and evaluated CoAPs for the Android OS. We have shown that smartphones with CoAPs capabilities can be used in many applications and hence we expect that CoAPs will play an important role in the future 6IoT. In the future, we plan to implement CoAPs for constrained devices and will evaluate the full 6IoT system consisting of sensors and INDIGO-powered smartphones.

Acknowledgements. This work has been partly performed within the SICS Center for Networked Systems (CNS), and partially funded by CALIPSO (FP7-ICT-2011.1.3-288879).

[4] `http://code.google.com/p/jcoap/`

[5] `http://people.inf.ethz.ch/mkovatsc/erbium.php`

[6] `http://www.openssl.org`, `http://www.yassl.com`, `http://www.gnu.org/software/gnutls`

References

1. Kushalnagar, N., Montenegro, G., Schumacher, C.: IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs): Overview, Assumptions, Problem Statement, and Goals. RFC 4919 (August 2007)
2. Shelby, Z., Kartke, K., Bormann, C., Frank, B.: Constrained Application Protocol (CoAP). draft-ietf-core-coap-12 (October 2012)
3. Hui, J., Thubert, P.: Compression Format for IPv6 Datagrams over IEEE 802.15.4-Based Networks. RFC 6282 (September 2011)
4. Winter, T., Thubert, P., Brandt, A., Hui, J., Kelsey, R., Levis, P., Pister, K., Struik, R., Vasseur, J., Alexander, R.: RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks. RFC 6550 (March 2012)
5. Rescorla, E., Modadugu, N.: Datagram Transport Layer Security Version 1.2. RFC 6347 (Proposed Standard) (January 2012)
6. Dierks, T., Rescorla, E.: The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246 (Proposed Standard) (August 2008); Updated by RFCs 5746, 5878, 6176
7. Campbell, A., Eisenman, S., Lane, N., Miluzzo, E., Peterson, R., Lu, H., Zheng, X., Musolesi, M., Fodor, K., Ahn, G.: The rise of people-centric sensing. IEEE Internet Computing 12(4), 12–21 (2008)
8. Cuff, D., Hansen, M., Kang, J.: Urban sensing: out of the woods. Communications of the ACM 51(3), 24–33 (2008)
9. Brachmann, M., Garcia-Morchon, O., Kirsche, M.: Security for practical coap applications: Issues and solution approaches. In: Proc. of the 10th GI/ITG KuVS Fachgespraech Sensornetze, FGSN 2011 (2011)
10. Kothmayr, T., Schmitt, C., Hu, W., Brunig, M., Carle, G.: A dtls based end-to-end security architecture for the internet of things with two-way authentication. In: 2012 IEEE 37th Conference on Local Computer Networks Workshops (LCN Workshops), pp. 956–963. IEEE (2012)
11. Raza, S., Trabalza, D., Voigt, T.: 6LoWPAN Compressed DTLS for CoAP. In: Proceedings of the 8th IEEE International Conference on Distributed Computing in Sensor Systems (IEEE DCOSS 2011), Hangzhou, China (May 2012)
12. Raza, S., Duquennoy, S., Chung, A., Yazar, D., Voigt, T., Roedig, U.: Securing communication in 6lowpan with compressed ipsec. In: 7th International Conference on Distributed Computing in Sensor Systems (DCOSS 2011), Barcelona, Spain (2011)
13. Raza, S., Duquennoy, S., Höglund, J., Roedig, U., Voigt, T.: Secure Communication for the Internet of Things - A Comparison of Link-Layer Security and IPsec for 6LoWPAN. In: Security and Communication Networks. Wiley (January 2012)