Performance and overhead evaluation of OSCOAP and DTLS

Martin Gunnarsson[1], Tobias Andersson[1], Ludwig Seitz[1]

[1] RISE SICS AB
Box 1263, Kista 16429, Sweden
{martin.gunnarsson, tobias.andersson, ludwig.seitz}@ri.se

October 2017

RISE SICS Technical Report T2017:05

Abstract. In this report we compare the OSCOAP protocol to CoAP over DTLS-PSK to evaluate their performance in constrained devices. **Keywords:** IoT M2M Security OSCOAP DTLS CoAP Constrained Devices LLN

1 Introduction

In this report we analyse the OSCOAP[1] protocol in comparison to DTLS[2], and a baseline implementation of CoAP[3], without any security mechanisms. We have done this to evaluate the feasibility of supplementing and/or replacing DTLS as the protocol of choice to secure communication in constrained devices operating on Low power and Lossy Networks (LLN).

2 OSCOAP

This section aims to present OSCOAP, the purpose of the protocol and the standards which it builds upon. OSCOAP is short for Object Security for the Constrained Application Protocol. It is an IETF standards proposal from Ericsson and RISE SICS. OSCOAP builds on several existing IETF standards, we will briefly explain them here. CBOR{citeRFC7049 - Compact Binary Object Representation, is a standard that provisions a compact way to encode data. It is meant as a replacement or supplement to existing data encoding schemes. CBOR is especially suited to constrained devices since it offers compact messages and simple processing. Building on CBOR is another IETF standard, COSE[4] - CBOR Object Signing and Encryption. COSE uses CBOR encoding to provide, among other things, encrypted and signed messages. OSCOAP ties this together by using COSE to protect CoAP messages. There are several advantages to using this approach.

The CoAP messages can be selectively protected to protect sensitive parts while keeping CoAP's forward-proxy functionality working. Another benefit is that by using CBOR to encode messages the per message overhead can be minimized. CBOR encodes fields as compact as possible, this property avoids sending redundant data. OSCOAP is currently under development and is hosted on Github[1], this document refers to an older version of the OSCOAP draft, namely version 4, any further references in this document to OSCOAP shall be interpreted as references to draft-ietf-core-object-security-04[1].

3 System Description

The comparisons are made for the cc2538dk development boards from Texas Instruments, the boards feature a 32-bit ARM Coretex-M3 CPU and 32KB of RAM and 512KB of flash memory (ROM)[2]. The cc2538 system on a chip has hardware acceleration of SHA2 and AES-128, but these are not used in

[1] https://github.com/core-wg/oscoap

[2] http://www.ti.com/product/CC2538/datasheet

this comparison. We have compared a baseline system with Contiki-OS and a CoAP server, a server with CoAP and DTLS (COAPS) and a server with OSCOAP.

On each server there is a simple REST-resource that returns the string "Hello World" to the client. The client will send a GET-message to that server every 20s.

We have compared three set-ups in this report, a baseline system with plain CoAP, a current best practise system with CoAP protected by DTLS[2], often called COAPS, and OSCOAP.

The baseline system is the er-coap library shipped with Contiki-OS[3]. We have made no alterations to the library and to Contiki-OS. The DTLS + COAP system and OSCOAP system also uses Contiki-OS.

We have compiled tinydlts[4] without ECC and other extensions. DTLS uses the profile "Datagram Transport Layer Security (DTLS) Profiles for the Internet of Things", this is a profile for usage of DTLS in constrained devices. The only included TLS-cipher suite in our comparison is TLS_PSK_WITH_AES_128_CCM_8.

The OSCOAP library used is the SICS implementation[5], it only supports the mandatory to implement AES_CCM_64_64_128 algorithm. OSCOAP also uses SHA2 for key derivation.

We have configured the nodes to use static keys, DTLS uses PSK and OSCOAP uses keys derived from a static Master-Secret.

In this comparison we have compared OSCOAP as specified in draft-ietf-core-object-security-04 and DTLS as specified in RFC6347[2] with the following DLTS profile: "Transport Layer Security (TLS) Datagram Transport Layer Security (DTLS) Profiles for the Internet of Things"[5]. This is a profile ,for DTLS, that forcuses on constrained devices, the crypto algorithms are limited and there are provisions to use PSK instead of a more resource demanding RSA or ECC authentication.

4 Comparisons

4.1 Network Overhead

The per packet overhead for DTLS + COAP is made up of the DTLS-record protocol. The DTLS-record protocol contains the fields shown with their lengths in Table 1.

The per-message overhead for OSCOAP depends on the type of message, request or response. Many parameters that are sent in the request are implicit

[3] https://github.com/contiki-os/contiki
[4] https://github.com/cetic/tinydtls
[5] https://github.com/Gunzter/contiki-oscoap

Table 1: DTLS-Record Layer per message overhead

Field	Size (Bytes)
Content Type	1
Version	2
Epoch	2
Sequence Number	6
Length	2
Nonce (Explicit)	8
ICV	8
Total:	29

in the response and do not have to be sent again thanks to the request - response mapping. Several parameters are also encoded to the shortest possible length. Table 3 show the fields in a OSCOAP request and their minimal lengths. Sequence number, also called Partial IV, and Key Id are encoded in this way in the request. For the responses these parameters are known to the client implicitly and does not have to be retransmitted. In Table 4 we show the fields sent in a OSCOAP response and the length of the fields.

Table 2: OSCOAP per message overhead

Table 3: OSCOAP Request

Field	Size (Bytes)
Option	1-2
Compressed COSE Header	1
Partial IV (Seq)	1-7
Key Id	>2
ICV	8
Total:	>14

Table 4: OSCOAP Response

Field	Size (Bytes)
Option	1
Compressed COSE Header	1
ICV	8
Total:	10

In summary we can see the results in Table 5.

Table 5: Message overhead comparison

Sequence Number:	'05'	'1005'	'10005'
DTLS 1.2 (Bytes)	29	29	29
OSCOAP Request (Bytes)	14	16	19
OSCOAP Response (Bytes)	10	10	10

4.2 Memory Utilization

When comparing memory utilization of the different libraries a straight up comparison of DTLS and OSCOAP, as done in Table 6, is not fair since DTLS includes, apart from the DTLS-record protocol, also the DTLS-handshake protocol. Table 6 details the fields of the executable on the device. A executable program is divided into several parts called .text, .bss and .data. The text segment .text is static memory and this segment contains among other things code and static variables. The field .data is data that is initialized to zero when the program starts, it will not take up any space on ROM but will be stored in RAM when the program is running. This is used for a stack, if present, and variables stored on the programs heap. Lastly the .bss segment, this is data that is initialized to a value when the program starts, it thus needs to be stored in ROM as well as in RAM. OSCOAP functions as the DTLS-record protocol and is intended to be used with a handshake protocol such as EDHOC[6]. DTLS also provides a variety of cipher suites, the OSCOAP library is limited to the mandatory to implement AES-CCM-64-64-128 algorithm. To make a comparison of the performance of the core functionality in DTLS and OSCOAP we have elected to leave out the cipher functions, hash functions and the DTLS record protocol, the functions removed are detailed in Table 7.

Table 6: Memory utilization of CoAP, OSCOAP and COAPS, (Bytes)

	COAP	OSCOAP	COAP + DTLS (COAPS)
Client:			
.text	44534	51364	61244
.bss	505	634	597
.data	9499	9751	11697
RAM	10004	10385	12294
ROM	45039	51998	61841
Server:			
.text	47535	54364	64309
.bss	505	2108	2073
.data	9499	9515	11323
RAM	10004	11623	13396
ROM	48040	56472	66382

In Table 8 we show the net memory consumption.

Table 7: Ciphers, Hash functions and DTLS-Handshake protocol.
.t is short for .text, .d = .data, .b = .bss (Bytes)

	COAPS Server	COAPS Client	OSCOAP Server	OSCOAP Client
Symmetric cipher	.t: 5614	.t: 5614	.t: 1448	.t: 1500
Hash algorithm	.t: 830	.t: 830	.t: 860 D: 32	.t: 830, .d: 32
Handshake	.t: 2968, .d: 12, .b: 225	.t: 2968, .d: 12	N/A	N/A

Table 8: Memory utilization without DTLS-handshake, crypto- and hash-algorithms (Bytes)

	COAP	OSCOAP	COAP + DTLS (COAPS)
Client:			
.text	44534	49034	51832
.bss	505	602	585
.data	9499	9751	11697
RAM	10004	10385	12282
ROM	45039	49636	52417
Server:			
.text	47535	52056	54897
.bss	505	2078	2061
.data	9499	9515	11098
RAM	10004	11591	11098
ROM	48040	54132	56958

5 Conclusions

We have concluded that OSCOAP provides advantages both in terms of memory usage on the device and also in bytes saved when messages are transmitted. This is mainly done by utilizing CBOR variable length fields that can shorten messages. One example of this is that sequence numbers can be encoded in as few bytes as possible. CBOR is also efficient to process, so it does not impact the code size of libraries using it in a significant way. Shortening the length of transmitted messages require less time for radio in transmit mode, for a constrained device this can lead to saving power and extending the usable time on a battery powered device.

References

1. G. Selander., J. Mattsson., F. Palombini., L. Seitz.: Object security of coap (oscoap). Internet-Draft draft-ietf-core-object-security-04, IETF Secretariat (2017) http://www.ietf.org/internet-drafts/draft-ietf-core-object-security-04.txt.
2. E. Rescorla., N. Modadugu.: Datagram transport layer security version 1.2. RFC 6347, RFC Editor (2012) http://www.rfc-editor.org/rfc/rfc6347.txt.
3. Z. Shelby., K. Hartke., C. Bormann.: The constrained application protocol (coap). RFC 7252, RFC Editor (2014) http://www.rfc-editor.org/rfc/rfc7252.txt.

4. J. Schaad.: Cbor object signing and encryption (cose). RFC 8152, RFC Editor (2017)
5. H. Tschofenig., T. Fossati.: Transport layer security (tls) / datagram transport layer security (dtls) profiles for the internet of things. RFC 7925, RFC Editor (2016)
6. G. Selander., J. Mattsson., F. Palombini.: Ephemeral diffie-hellman over cose (edhoc). Internet-Draft draft-selander-ace-cose-ecdhe-07, IETF Secretariat (2017) http://www.ietf.org/internet-drafts/draft-selander-ace-cose-ecdhe-07.txt.