R. Reussner, A. Koziolek, R. Heinrich (Hrsg.): INFORMATIK 2020,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2021 1237

Application Layer Security for the IoT

Information Security for CoAP with LCap, Payload Encryption, and HMACs

Marian Buschsieweke[1] Mesut Güneş[2]

Abstract: Security for the mostly constrained devices forming the IoT is an active field of research. In this paper, we propose two CoAP Options, *HMAC1*/*HMAC2* and *Crypt1*/*Crypt2* complementing our previous work on Lightweight Capability Based Access Control (*LCap*). This results in a lightweight, flexible, and complete solution for application layer security for CoAP nodes with severely limited memory. In our evaluation, we show that a pure software implementation without cryptographic hardware acceleration is feasible for practical use on highly constrained IoT devices. Due to mostly idiomatic use of cryptography, existing security analyses apply to our proposal. Our security framework was designed with ample focus on reducing the complexity of the system, which allows lean implementations and simplifies security reviews. This makes *LCap* based security a good fit for security in the IoT.

1 Introduction

An ever increasing number of daily objects are equipped with a microcontroller and network connectivity to form SmartX environments. The architecture of the Internet of Things (IoT), in which any given IoT node can communicate with any other, is highly promising for such smart environments: This ubiquitous availability paves the way for spontaneous cooperation between any willing IoT nodes and allows for emergent systems to be formed. The other side of the coin is that the risk of abuse of IoT nodes is capital. In order to exploit the potential of this architecture while controlling the risk, efforts on providing security and privacy for the IoT are being made in both academia and industry [TB19]. In this paper, a complete application layer security stack is proposed to join this effort.

The remainder of this paper is structured as follows: Promising work on application layer security for IoT nodes is presented in Section 2. In Section 3 we propose the *HMAC1*/*HMAC2* and the *Crypt1*/*Crypt2* CoAP Options to provide integrity, authenticity, data freshness, and confidentiality for messages send using CoAP [SHB14]. These options are complementary to our previously proposed solution for access control, *LCap* [BG18], and combined provide a complete security framework for CoAP nodes. The impact the use of our proposed security framework has on response time and required CPU instruction is evaluated in Section 4. In Section 5, a security evaluation is performed. Finally, a conclusion is drawn in Section 6.

[1] Otto-von-Guericke University Magdeburg, Communication and Networked Systems (ComSys), Faculty of Computer Science, Universitätsplatz 2, 39106 Magdeburg, Germany, marian.buschsieweke@ovgu.de
[2] Otto-von-Guericke University Magdeburg, Communication and Networked Systems (ComSys), Faculty of Computer Science, Universitätsplatz 2, 39106 Magdeburg, Germany, mesut.guenes@ovgu.de

doi:10.18420/inf2020_115

2 Related Work

In the following, the most promising work on access layer information security targeting IoT nodes of class C2 (using RFC 7228 [BEK14] terminology) and is presented below.

2.1 OAuth

The IETF working group Authentication and Authorization for Constrained Environments (ACE) is strongly engaged in work on information security for IoT devices. One approach to access control is the adaption of OAuth for the IoT [Se20]. For this, efficient encodings of OAuth entities such as the CBOR Web Token [Jo18] to encode claims are used. Claeys et al. used these building blocks to employ OAuth 1.0a on IoT nodes [CRT18]. They were able to implement core components of their proposal on a class C2 device (using RFC 7228 [BEK14] terminology). However, without a complete implementation the feasibility of their proposal cannot be conclusively verified. In addition, applicability to the more constrained classes C1 and C0 IoT nodes is crucial for an access control solution for the IoT. So far, it remains unclear whether OAuth is lightweight enough to be run on the more constrained IoT nodes below class C2.

2.2 Object Security for Constrained RESTful Environments (OSCORE)

Object Security for Constrained RESTful Environments (OSCORE) [Se19] implements application layer security for CoAP relying on CBOR Object Signing and Encryption (COSE) [Sc17]. It primarily targets CoAP nodes communicating over proxies with either CoAP or CoAP-mappable HTTP endpoints. In this context secure communication channels on the transport layer have to terminate at the proxy to enable it to perform required modification of the forwarded messages, such as the conversion between CoAP and HTTP. OSCORE moves the actual request and response into the payload using Concise Binary Object Representation (CBOR) for serialization and COSE for security. Only header fields and options required for conformance with CoAP/HTTP or required to be accessible by proxies are additionally exposed in the message carrying an OSCORE request/response. For access control, OSCORE relies on supplementary approaches such as OAuth.

2.3 Capability Based Access Control

The basic idea of Capability Based Access Control (CBAC) is the separation of the decision on and the enforcement of access. It puts the client in charge of proactively obtaining an unforgeable capability token that proves access rights. The authority issuing this token is thus deciding on the access. A server receiving a capability token is only enforcing this decision by verifying the validity, authenticity, integrity, and genuineness of the token. The basic idea was first introduced by Saltzer et al. [SS75] in the context of processes requesting access to resources managed by an operating system. Mahalle et al. [Ma12] proposed transferring this approach as described above to network communication. Chen et al. [CGH16] proposed using a CoAP Option to embed the capability token in the request.

2.4 Delegated CoAP Authentication and Authorization Framework (DCAF)

Gerdes et al. [GBB15] proposed to let constrained servers and clients delegate authorization to a Server Authorization Manager (SAM) and a Client Authorization Manager (CAM), respectively. In the Delegated CoAP Authentication and Authorization Framework (DCAF), an access ticket is issued to the client once the authorization managers agree on granting access. Exactly like a capability token, this ticket proves granted access to a requested resource. But in addition, DCAF delegates key negotiation to the authorization managers, so that a DTLS channel will be established using this pre-shared key. Thus, DCAF provides a full security stack with access control being implemented on the application layer, and integrity, authenticity and confidentiality on the transport layer.

2.5 Lightweight Capability Based Access Control (*LCap*)

Buschsieweke and Güneş introduced the Lightweight Capability Based Access Control (*LCap*) [BG17; BG18], a lightweight, scalable and flexible access control for IoT nodes. Unlike other approaches for CBAC, *LCap* relies solely on symmetric cryptography. In *LCap*, individual keys are exchanged between CoAP servers and *Token Authorities*. These keys are used to sign *LCap Tokens* using a classic HMAC scheme. In addition, every *LCap Token* contains an individual *Token Key* that is encrypted using the keys shared with the *Token Authorities*. This zero round trip key exchange of the *Token Key* allows the use of symmetric cryptography while the number of pre-shared keys a CoAP server needs to store remain constant with growing number of authorized CoAP clients; and can be as low as one. CoAP clients have to provide a valid *LCap Token* in order to access a resource. As proof of possession of the *LCap Token*, a client proves knowledge of the plain text of the *Token Key*, whose cipher text is embedded in the *LCap Token*. The server can decrypt the embedded encrypted *Token Key* using the keys exchanged with the issuing *Token Authority*. *LCap* additionally provides authenticity and integrity by using the *Token Key* to attach an HMAC protecting the CoAP request. Replay attacks are prevented and data freshness is enforced by adding a slow ticking time stamp, the *LCap Epoch*: Within the validity of an *LCap Epoch* CoAP's duplicate detection will reject replays as duplicates. *LCap* uses CoAP Options nested within the *LCap Option* to encode additional side conditions that need to be met in order to gain access. These Suboptions can be marked as critical and elective in the same manner as CoAP Options, so that new side conditions can be defined in a backward compatible manner.

3 *LCap* Based Application Layer Security

LCap by itself implements access control and additionally provides authenticity, integrity, and data freshness for the request. This paper introduces two additional groups of CoAP Options: The *HMAC* Options and the *Payload Encryption* Options. The former provides the same security guarantees for the response, the latter enables confidentiality (for both requests and responses).

Name	#	Type	Description
Time[1]	1	`uint`[2]	Timestamp the HMAC was calculated
Key ID	9	`uint`[3]	ID of key used in the HMAC
Algo	11	`uint`[3]	ID of the HMAC algorithm used

1. This Suboption is not used for HMAC Options in negotiation mode.
2. Unix time stamp.
3. Value is 0 if Suboption is missing.

Tab. 1: List of Suboptions used in *HMAC1* and *HMAC2* Options

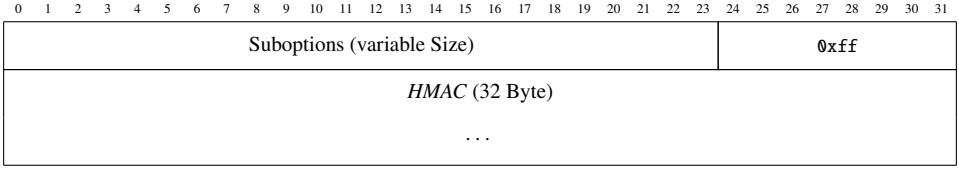


Fig. 1: Format of the *HMAC1*/*HMAC2* Option when carrying an HMAC

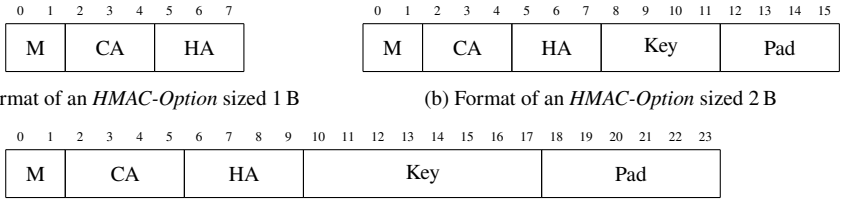
3.1 Integrity and Authenticity Protection

The *HMAC* Options use a simple HMAC-Scheme [KBC97] to protect the integrity and authenticity of the message carrying them. Similar to the terminology in the CoAP Block-Wise Transfer [BS16], the *HMAC2* Option always refers to the response and the *HMAC1* Option refers to the request. The *HMAC1* Option is incompatible with the use of *LCap* in requests and is intended as a lightweight alternative to *LCap* in scenarios access control is not needed. The *HMAC2* Option complements the use of *LCap* by protecting the response.

An *HMAC2* Option in a response or an *HMAC1* Option in a request is referred to as an *HMAC* Option in HMAC mode. In this mode, *HMAC* Options contain an HMAC value to protect the integrity of the IP address and port of both sender and receiver, as well as the whole content of the CoAP message. The cryptographic parameters used for the calculation of the HMAC and the time stamp when the message was created are given using the Suboptions in Fig. 1. These Suboptions use the CoAP Option Format [SHB14], but are stored within the data section of a regular CoAP Option. The encoding of an *HMAC* Option in HMAC mode is depicted in Tab. 1.

An *HMAC2* Option in a request or an *HMAC1* Option in a response is referred to as an *HMAC* Option in negotiation mode. As this name implies, these options are used to negotiate cryptographic parameters of the HMAC, rather than carrying an HMAC. The negotiation mode is particularly useful if the server (or client) defaults to a cryptographic hash algorithm not supported by the client (or server). For obvious reasons, neither the *Time* Suboption nor an HMAC value is added when the HMAC Options are used for negotiation.

The BLAKE2s-256 [Au14] cryptographic hash function is referred to with ID 0 as *Algo*.



(a) Format of an *HMAC-Option* sized 1 B

(b) Format of an *HMAC-Option* sized 2 B

(c) Format of an *HMAC-Option* sized 3 B

M *Mode ID*, specifies the block cipher mode of operation
CA *Cipher ID*, specifies the block cipher algorithm to use
HA *Hash ID*, specifies the cryptographic hash to use to generate the initialization vector (IV)
Key *Key ID*, specifies the key to use for encryption (defaults to zero)
Pad *Padding*, specifies the number of padding bytes added to the input (defaults to zero)

Fig. 2: Formats of the *Crypt1*/*Crypt2* Option depending on the size of the CoAP Option

Alg. 1: Algorithm used to derive the IV

Data: shared key as *key*, message as *msg*, LCap Epoch or HMAC time stamp as *nonce*
Result: derived IV

```
return cryptoash(key, nonce, msg.token, msg.message_id)
```

A value of 0 in the *Key ID* is only allowed in the *HMAC2* Option and refers to the same key used to protect the integrity of the request. In case the request used an *LCap Token*, the *Token Key* is referred to with *Key ID* 0.

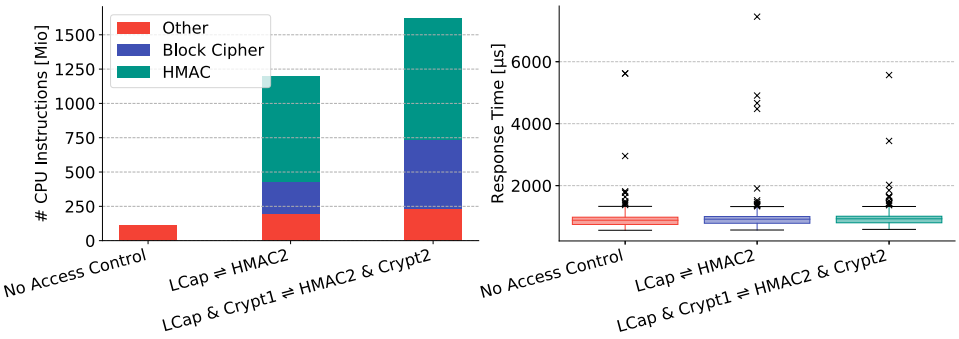
3.2 Confidentiality of the Payload

Payload Encryption of CoAP messages is provided by the *Crypt1* and *Crypt2* Options. The naming convention is the same as used for the HMAC Options: *Crypt1* always refers to the request and *Crypt2* to the response. Again, both CoAP Options are allowed for both request and response.

A *Crypt1* Option in a request or a *Crypt2* Option in a response is referred to as Crypt Option in encryption mode. In this mode, the option specifies the cryptographic parameters used for encrypting the payload of the CoAP message. The encoding of *Crypt1* and *Crypt2* Options is depicted in Fig. 2.

A *Crypt1* Option in a response or a *Crypt2* Option in a request is referred to as Crypt Option in negotiation mode. Again, these are used to indicate the preferred configuration to the communication partner. The format of the Crypt Option in encryption mode as shown in Fig. 2 is also used in negotiation mode. However, the padding is always set to zero during negotiation.

Using *Payload Encryption* implies the use of either the *LCap* Option or the *HMAC1*/*HMAC2*. The motivation for this is on the one hand that there is practically no use case for confidentiality without integrity and authenticity. On the other hand, this allows reusing the



(a) Total CPU instruction required by the server

(b) Response time

Fig. 3: Benchmark for different security settings when using AES-128 as block cipher, BLAKE2s-256 as cryptographic hash function, and CFB as mode of operation

replay protection of the *LCap* Option or the *HMAC* Options to compute IVs, rather than increasing the size of CoAP messages by explicitly transferring it. The cryptographic hash function specified using *Hash ID* is used to compute the IV as shown in Alg. 1. The result is truncated as needed for the selected block cipher.

A *Cipher ID* of 0 refers to AES-128, a *Mode ID* of 0 specifies cipher feedback (CFB) as mode of operation, and a *Key ID* of 0 refers to the same key that was used in the *HMAC* Option or in the *LCap Token* that was used to protect the integrity of the message.

4 Performance Evaluation

4.1 Benchmark Setup

A desktop class PC was used to run both the CoAP client and server, so that they could communicate using the local network device. Thus, the communication is not affected by network congestion and channel properties. The client sent 32 767 `PUT` requests to the URI-Path `/led` with the payloads `1` and `0` in turns. For each of the following security parameters benchmarks were run:

1. No application layer security for both request and response
2. *LCap*-Option in the request, HMAC2-Option in the response
3. *LCap*- and Crypt1-Option in the request, HMAC2- and Crypt2-Option in the response

For each configuration a benchmark was run with different combinations of the used block cipher, the used mode of operation, and the used cryptographic hash function. The response time was measured and the total number of required CPU instructions the server required to handle all 32 767 requests were recorded using `callgrind`.

4.2 Benchmark Results

As shown in Fig. 3a, the required CPU instructions increase by a factor of ≈ 11.7 when protecting the request using the *LCap*-Option and the response using the HMAC2-Option. When additionally the payload of both request and response are encrypted, the CPU instruction increase by a factor of ≈ 14.5. The additionally required CPU instructions are mostly spent on the cryptographic primitives, namely the used block cipher and the used cryptographic hash function. The remaining additional CPU instructions are required to construct and parse the additional CoAP Options as well as the verification of the *LCap* attributes such as whether the token is used within its period of validity.

In Fig. 3b the response time depending on the used security options is shown. These box plots indicate that on desktop class hardware the impact caused by the use of application layer security is completely negligible. Apparently, the response time is I/O bound for every setting on the used hardware.

5 Security Evaluation

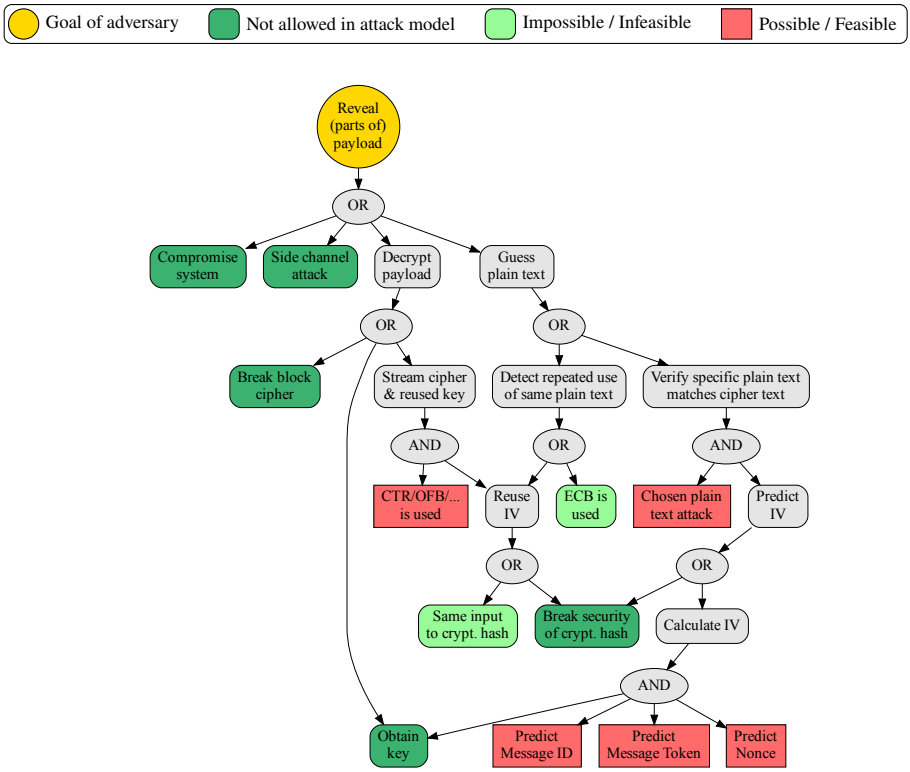
5.1 Attack Model

In the following, the security of the *Payload Encryption* and the *HMAC1*/*HMAC2* Options proposed in the paper are analyzed. For that it as assumed that the adversary is able to intercept any message, record any message, and inject and alter messages at will. It is however assumed that the adversary is unable to compromise any of the communicating nodes and no implementation flaws are present in communicating systems, including those leaking details on side channels. Finally, it is assumed that it is infeasible for the adversary to break the security of the used cryptographic hash function and the used block cipher.

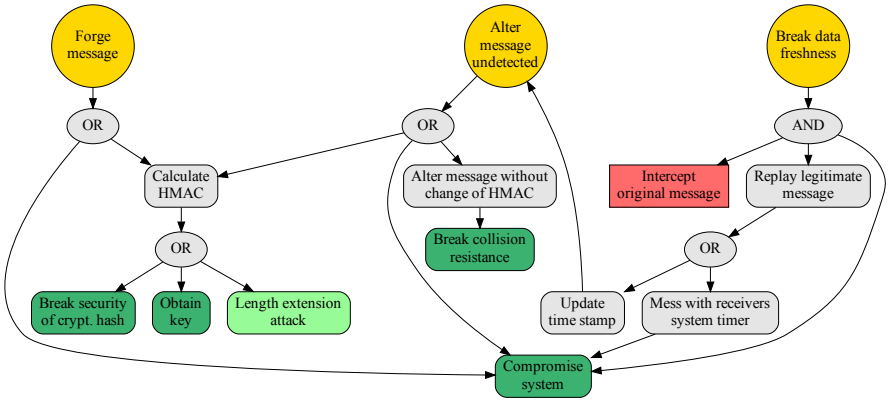
This attack model, thus, rules out the increasingly relevant attack vector of side channel attacks as well as any attack on the used cryptographic primitives. The reason for this is that side channel attacks inherently target a specific implementation rather than the actual specification this paper presents. Hence, these attacks are out of scope of this paper. Similarly, the security of cryptographic building blocks is out of scope of this paper. Instead we refer to the plethora of cryptanalysis published for the widely used block ciphers and cryptographic hash functions.

5.2 Security Evaluation of the *Payload Encryption*

In Fig. 4a an attack tree is shown that analyzes potential attacks on the confidentiality of the payload when using *Payload Encryption*. As *Payload Encryption*, with the exception of how the IV is obtained, is an idiomatic use of symmetric encryption, plenty of existing security analysis is present. Thus, most attack vectors apply to symmetric cryptography in general and are excluded from the adversary abilities in Section 5.1 as out of scope. The major difference to textbook use of symmetric cryptography is the computation of the IV, rather



(a) Attack tree for *Payload Encryption*



(b) Attack tree for *HMAC1*/*HMAC2* Options

Fig. 4: Attack trees analyzing attack vectors on the *Payload Encryption* and *HMAC1*/*HMAC2* Options

than choosing it randomly (using a high entropy source of randomness) and transferring it explicitly. With the use of ECB as block cipher mode of operation in *Payload Encryption* being forbidden, all remaining attack vectors listed in 4a depend on either the same IV to be generated more than once or being predictable by the adversary. Predicting the IV requires predicting the cryptographic hash of the concatenation of the key, the "Nonce" (either the *Epoch* of the *LCap* Option or the time stamp of the *HMAC*-Option used together with the *Payload Encryption*), the Message Token, and the Message ID. With the exception of the key, all parts of the input are easy to predict by an adversary. Assuming enough bits in the key are unknown to the adversary, it still is infeasible for the adversary to predict the hash due to the security guarantees of the cryptographic hash function.

As CoAP relies on the Message ID for duplicate detection, the same Message ID cannot be reused for the duration the communication partner is expecting duplicates. By the time the same Message ID can be reused again, the "Nonce" (the *LCap Epoch* or the *HMAC*) time stamp is different. Thus, the IV of different messages is never calculated using the same input. The chance of a cryptographic hash function with a high collision resistance yielding the same hash value again for these distinct input values is therefore close enough to zero, that it becomes infeasible for an adversary to count on this.

5.3 Security Evaluation of the *HMAC1/HMAC2* Options

Possible attacks on the message integrity, authenticity or data freshness are analyzed in the attack tree in Fig. 4b. The *HMAC* Option relies on idiomatic use of a cryptographic hash function that relies on the classic HMAC [KBC97] construct to prevent length extension attacks, so that e.g. the SHA-256 hash function can securely be used. The addition of a time stamp that is also covered by the HMAC, a receiver of a message protected by an *HMAC* Option is able to determine and enforce the freshness of the received data.

6 Conclusion

In this paper we have complemented our existing work on *LCap*, a lightweight implementation of Capability Based Access Control (CBAC) that additionally provides integrity, authenticity and data freshness for CoAP requests: We introduced the *HMAC2* Option to extend the same security to CoAP responses and proposed the *Crypt1* / *Crypt2* Option to provide confidentiality by encrypting the payload of CoAP messages. As both constructs are idiomatically applying symmetric cryptography to protect CoAP messages, preexisting security analyses mostly apply to the introduced *Payload Encryption* and *HMAC2* Option. Hence, the security implications and best practises are well understood. In addition, we provide a novel scheme to securely compute the initialization vector (IV) used for the *Payload Encryption*, which frees communication partners of explicitly sending IVs. Our performance evaluation shows that pure software implementations without cryptographic accelerators perform well, making it suitable for use in highly constrained IoT devices. In combination, *LCap*, *HMAC2*, and *Payload Encryption* form a complete application layer security framework for constrained CoAP nodes.

References

[Au14] Aumasson, J.-P.; Meier, W.; Phan, R. C.-W.; Henzen, L.: BLAKE2. In: Information Security and Cryptography. Springer Berlin Heidelberg, pp. 165–183, 2014, URL: https://doi.org/10.1007/978-3-662-44757-4_9.

[BEK14] Bormann, C.; Ersue, M.; Keränen, A.: Terminology for Constrained-Node Networks, RFC 7228, May 2014, URL: https://rfc-editor.org/rfc/rfc7228.txt.

[BG17] Buschsieweke, M.; Günes, M.: Securing Critical Infrastructure in Smart Cities: Providing Scalable Access Control for Constrained Devices. In: PIMRC'17 - Workshop on "Personalised Mobile Applications for Smart Cities and Smart Citizens (PMA 2017)". Montreal, Canada, Oct. 2017.

[BG18] Buschsieweke, M.; Günes, M.: Access Control for Medical Devices: Tweaking LCap for Health Informatics. In: IEEE Global Communications Conference (GlobeCom), Workshop on Wireless Energy Harvesting Communication Networks. Abu Dhabi, UAE, Dec. 2018.

[BS16] Bormann, C.; Shelby, Z.: Block-Wise Transfers in the Constrained Application Protocol (CoAP), RFC 7959, Internet Engineering Task Force, Aug. 2016, URL: http://www.ietf.org/rfc/rfc7959.txt.

[CGH16] Chen, B.; Güneş, M.; Huang, Y.-L.: CoAP Option for Capability-Based Access Control for IoT-Applications. In: Proceedings of the International Conference on Internet of Things and Big Data. Scitepress, 2016, URL: https://doi.org/10.5220%2F0005950902660274.

[CRT18] Claeys, T.; Rousseau, F.; Tourancheau, B.: Securing Complex IoT Platforms with Token Based Access Control and Authenticated Key Establishment. In: International Workshop on Secure Internet of Things (SIOT). Oslo, Norway, Feb. 2018, URL: https://hal.archives-ouvertes.fr/hal-01596135.

[GBB15] Gerdes, S.; Bergmann, O.; Bormann, C.: Delegated CoAP Authentication and Authorization Framework (DCAF), draft-gerdes-ace-dcaf-authorize-04, Internet Engineering Task Force, Oct. 2015, URL: https://tools.ietf.org/html/draft-gerdes-ace-dcaf-authorize-04.

[Jo18] Jones, M.; Wahlstroem, E.; Erdtman, S.; Tschofenig, H.: CBOR Web Token (CWT), RFC 8392, May 2018, URL: https://rfc-editor.org/rfc/rfc8392.txt.

[KBC97] Krawczyk, H.; Bellare, M.; Canetti, R.: HMAC: Keyed-Hashing for Message Authentication, RFC 2104 (Informational), Updated by RFC 6151, Internet Engineering Task Force, Feb. 1997, URL: http://www.ietf.org/rfc/rfc2104.txt.

[Ma12] Mahalle, P. N.; Anggorojati, B.; Prasad, N. R.; Prasad, R.: Identity driven capability based access control (ICAC) scheme for the Internet of Things. In: ANTS'12. Institute of Electrical & Electronics Engineers (IEEE), Dec. 2012, URL: http://dx.doi.org/10.1109/ANTS.2012.6524227.

[Sc17] Schaad, J.: CBOR Object Signing and Encryption (COSE), RFC 8152, Internet Engineering Task Force, July 2017.

[Se19] Selander, G.; Mattsson, J.; Palombini, F.; Seitz, L.: Object Security for Constrained RESTful Environments (OSCORE), RFC 8613, July 2019, URL: https://rfc-editor.org/rfc/rfc8613.txt.

[Se20] Seitz, L.; Selander, G.; Wahlstroem, E.; Erdtman, S.; Tschofenig, H.: Authentication and Authorization for Constrained Environments (ACE) using the OAuth 2.0 Framework (ACE-OAuth), Internet-Draft draft-ietf-ace-oauth-authz-33, Work in Progress, Internet Engineering Task Force, Feb. 2020, URL: https://datatracker.ietf.org/doc/html/draft-ietf-ace-oauth-authz-33.

[SHB14] Shelby, Z.; Hartke, K.; Bormann, C.: The Constrained Application Protocol (CoAP), RFC 7252, Updated by RFC 7959, Internet Engineering Task Force, June 2014, URL: http://www.ietf.org/rfc/rfc7252.txt.

[SS75] Saltzer, J.; Schroeder, M.: The protection of information in computer systems. Proceedings of the IEEE 63/9, pp. 1278–1308, 1975, URL: https://doi.org/10.1109%2Fproc.1975.9939.

[TB19] Tschofenig, H.; Baccelli, E.: Cyberphysical Security for the Masses: A Survey of the Internet Protocol Suite for Internet of Things Security. IEEE Security & Privacy 17/5, pp. 47–57, 2019.