CoAP Option for Capability-Based Access Control for IoT-Applications

Borting Chen[1,2], Mesut Güneş[2] and Yu-Lun Huang[1]
[1]*Institute of Electrical and Control Engineering, National Chiao Tung University, Hsinchu City, Taiwan*
[2]*Institute of Computer Science, University of Münster, Münster, Germany*

Keywords: Capability-Based Access Control, Internet of Things, Network Security.

Abstract: Access control is critical for many applications of the Internet of Things (IoT) since the owner of an IoT device (and application) may only permit one user to access a subset of the resources of the device. To provide access control for an IoT network, recent work adopted the capability-based access control (CBAC) model, which allows an IoT device to decide on the authorization by itself based on a capability token. However, the existing approaches based on CBAC directly attach the capability token at the end of CoAP when sending a request message. For the receiver, it is not easy to retrieve the capability token from the request message if the CoAP payload is present, because CoAP does not have a length field to indicate the size of its payload. To counter this problem, we propose a CoAP option, Cap-Token, to encapsulate a capability token when sending request messages. Because a CoAP option is independent from other CoAP fields, a receiver can get the capability token from the Cap-Token option of the request message without ambiguity. We also provide a compression mechanism to reduce the size of the Cap-Token option. Our evaluation shows that the compression mechanism can save the size of the option by 60%. Adding a compressed Cap-Token option to a request message increases the IP datagram size by 45 bytes, which is only 41% of the increase when directly attaching the capability token at the end of CoAP.

1 INTRODUCTION

The Internet of Things (IoT) is a promising architecture for the next generation Internet where every device has the ability to communicate with each other. Various communication protocols, such as 6LoWPAN (Montenegro et al., 2007) and RPL (Winter et al., 2012) has been proposed to support realizing this concept in many fields, including home automation, smart building and industrial control (Seitz and Gerdes, 2015). With the growing use of IoT technologies, IoT security attracts more attention in academia and industry. IoT security includes confidentiality, integrity, authentication, data freshness, and access control (Alghamdi et al., 2013) (Granjal et al., 2015) (Nguyen et al., 2015) (Roman et al., 2013). Among these security requirements, access control is critical because an IoT device can provide multiple resources, however the device owner may only permit a user to access a subset of them. Hence, some research has been conducted to present different access control solutions for an IoT network.

Recent work adopted the capability-based access control (CBAC) model (Gusmeroli et al., 2013) for an IoT network. CBAC allows an IoT device to handle the authorization processes by itself based on a capability token sent with the request message. A capability token contains access permissions granted to a requester[1]. The requester sends a request message with the capability token, and the receiver decides on the authorization of the request based on the permissions described in that capability token. Such a design enables applying access control for an IoT network where a centralized authorization server is hard to deploy. It also helps to enforce the principle of least privilege, because different requesters can be granted with capability tokens containing different access permissions.

However, the existing approaches based on CBAC still have some issues.

- When applying CBAC for an IoT network which uses the Constrained Application Protocol (CoAP) (Shelby et al., 2014) as its communication protocol, every CoAP request message should be sent with a capability token. This is because CoAP was designed according to the Rep-

[1]We use "requester" instead of "user" in this paper, because in an IoT network the source of a request message can be a human or a network-connected device.

Chen, B., Güneş, M. and Huang, Y-L.
CoAP Option for Capability-Based Access Control for IoT-Applications.
DOI: 10.5220/0005950902660274
In *Proceedings of the International Conference on Internet of Things and Big Data (IoTBD 2016)*, pages 266-274
ISBN: 978-989-758-183-0

resentational State Transfer (REST) architecture, which requires the interaction between a requester and a receiver being stateless. Some CBAC approaches (Gusmeroli et al., 2013) (Hernández-Ramos et al., 2013) realized this requirement by directly attaching the capability token at the end of CoAP, i.e., behind the CoAP payload. However, CoAP does not have a length field to indicate the size of its payload. If the CoAP payload is present in a request message (e.g., "*UPDTAE*" or "*PUT*" messages), the receiver has no means to retrieve the capability token from the message, because the capability token is mixed up with the CoAP payload. We call this issue as the "payload ambiguity problem."

- A capability token contains not only access permissions but also data for identification, validity verification, and integrity checking. This causes a capability token is usually large in size. However, in a 6 LoWPAN network, the maximum frame size defined in IEEE 802.15.4 is much smaller than the minimum MTU for IPv6 (127 bytes vs. 1280 bytes). If a large size capability token is sent with the request message, the message might be fragmented into several data link frames. Things get worse when the 6 LoWPAN network uses RPL as the routing protocol. Because RPL is a hop-by-hop routing protocol, an intermediate router needs to collect all data link frames of a request message to determine the next forwarding target. Losing any data link frame fails the reassembling of the message, which means the entire message is lost. This decreases the success rate of delivering request messages over a 6 LoWPAN network.

To address these issues, we design a CoAP option, called Cap-Token, to encapsulate a capability token when sending a request message. Because a CoAP option is independent from other CoAP fields and the option knows its size, the capability token would not be mixed up with the contents of other CoAP fields. The receiver can hence retrieve the capability token from the Cap-Token option of a request message without ambiguity. Besides, to reduce the increase in the IP datagram size after adding the Cap-Token option to the request message, we provide a compression mechanism for the Cap-Token option. Since the capability token in the Cap-Token option may have duplicate contents in other CoAP fields or in the lower-layer protocols of a request message, the compression mechanism can elide those duplicates to minimize the size of the Cap-Token option. This results in smaller IP datagrams for request messages, reducing the chance of the occurrence of link-layer fragmentation in a 6 LoWPAN network. The main contributions of this paper are as follows.

- The Cap-Token option solves the "payload ambiguity problem" for a receiver when processing a CoAP request message containing both a capability token and a payload. The receiver can retrieve the Cap-Token option from any type of CoAP request message and get the capability token from the option.
- The compression mechanism saves 60% of the original size of the Cap-Token option. Benefited from the compression, we can reduce the increase in the IP datagram size when applying CABC for an IoT network. Our evaluation shows that adding a compressed Cap-Token option only increases the IP datagram size by 69%. In comparison to directly attaching a capability token at the end of CoAP, which increases the IP datagram size by 304%, our work significantly reduces the IP datagram size.
- The compression mechanism also helps reduce the chance of the occurrence of the link-layer fragmentation when transmitting CoAP request messages over a 6LoWPAN network. Because of the smaller IP datagram size, using a compressed Cap-Token option allows a request message to be transmitted in 1 packet. Comparatively, directly attaching a capability token would cause the IP datagram being fragmented into 3 data link frames, which downgrades the success rate of delivering request messages.

The paper is organized as follows. We first brief the design of our capability token in Section 2, and describe the design of the Cap-Token option and the compression mechanism in Section 3. Section 4 evaluates the benefits of the compression mechanism and Section 5 discusses how the compression mechanism impacts the request message transmission in a 6LoWPAN network. We review the related work in Section 6. Section 7 concludes the paper.

2 CAPABILITY TOKEN

A capability token contains access permissions granted to a requester as well as the necessary information to verify the ownership, integrity, and validity of the capability token.

In our design, a capability token includes the following data.

- **Issuer Identity** (`II`) is a 4-byte-long unsigned integer, which indicates the identity of the server that issues this capability token.

- **Subject Identity** (SI) makes reference to the owner of this capability token. SI is the IPv6 address of the owner's device.
- **Object Identity** (OI) makes reference to the target device which holds resources that the owner of this capability token is able to access. OI is the IPv6 address of the target device.
- **Issued Time** (IT) indicates the time at which the capability token was issued. IT is a 4-byte-long integer, representing how many seconds are elapsed since the Epoch time.
- **Not Before** (NB) indicates the time at which the capability token becomes valid. NB is a 4-byte-long integer, representing how many seconds are elapsed since the Epoch time.
- **Not After** (NA) indicates the time at which the capability token becomes expired. NA is a 4-byte-long integer, representing how many seconds are elapsed since the Epoch time.
- **Permission List** (PL) stores a set of access permissions granted to the owner of this capability token. Each access permission is composed of three elements, resource path, request methods, and additional conditions:
  - **Resource Path** (RP) is the path to a resource.
  - **Request Methods** (RM) indicates the types of request methods granted to the owner for accessing RP. The definition of request methods is aligned with the definition in RFC 7252 (Shelby et al., 2014).
  - **Additional Conditions** (AC) describes a set of conditions which should be fulfilled when the receiver decides on the authorization of an access request to RP. The contents of additional conditions could be related to requester's location information or receiver's battery status (Seitz et al., 2013) (Hernández et al., 2014).
- **Issuer Signature** (IS) stores the digital signature signed by the issuer to provide authenticity and integrity protection for the capability token.
- **Token Sub-Identity** (TI) is a 1-byte-long unsigned integer, which stores the data to support the issuer server in identifying this capability token.

In order to manage the issued capability tokens, an issuer server also assigns each capability token with an identity (ID). ID is a combination of the subset of the contents of a capability token, which is SI||OI||IT||NB||NA||TI. TI is usually assigned with 0. Only when there exists two capability tokens having the same SI, OI, IT, NB, and NA, one of the capability tokens would be assigned with different value in TI to distinguish their identities. Besides, since ID depends on other contents of the capability token, when an issuer server generates a digital signature for the capability token, ID would not be included in the calculation. ID is also excluded when storing a capability token on the requester's device or sending a capability token with a request message in order to save the storage space and bandwidth.

```
"capability_token": {
    "II": 46804706,
    "SI": "2002::8c71:65",
    "OI": "2002::8c71:66",
    "IT": 1447066800000,
    "NB": 1447066800000,
    "NA": 1447066800000,
    "PL": [{"RP": "temperature",
            "RM": ["001"]}],
    "IS": "jBghEa2aO8v;urVIbafsQad_d1fn",
    "TI": 0
}
```

Figure 1: Example of a capability token.

Figure 1 shows an example of a capability token, which stores its contents in JSON format. The capability token is used to obtain temperature data from a sensor in a smart home environment. The owner's IPv6 address is "*2002::8c71:65*" and the sensor's IPv6 address is "*2002::8c71:66*". "001" in RM indicates the "*GET*" request method is granted to the owner of this capability token for accessing the "temperature" resource. NB equals to IT means this capability token is valid immediately after the capability token was issued, and NA equals to NB means the validity time of this capability token is permanent. The issuer server uses the Elliptic Curve Digital Signature Algorithm (ECDSA) with the *secp112r1* curve to generate SI, which is 28 bytes in length.

When the receiver obtains a capability token from the request message, it first examines NB and NA to check whether the capability token is still valid. Then, SI and OI are checked to insure the requester is the owner of the capability token and the receiver is the target device. Next, the receiver checks whether the requested resource and request method have a match in PL. If the matched access permission has additional conditions, the receiver also needs to insure these conditions are fulfilled. IS is verified at the last step, because it costs more computational power than the previous steps. If a request is rejected at the previous steps, the high-cost digital signature calculation can be avoided. Finally, if the digital signature is valid, the receiver authorizes the request and processes the request message.

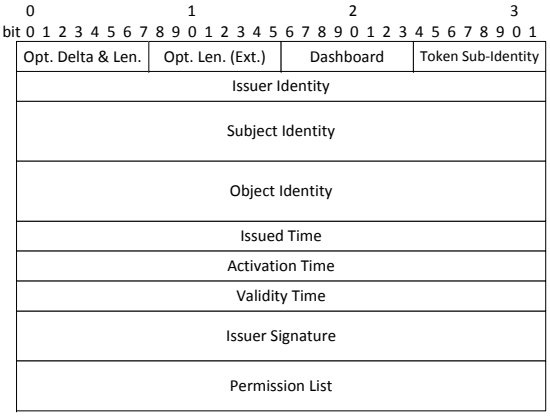

Figure 2: The option fields of the Cap-Token option.

3 CAP-TOKEN OPTION

The Cap-Token option is designed for solving the "payload ambiguity problem." When sending a request message, the requester encapsulates a capability token into the Cap-Token option and embeds the option into the message to be sent. The receiver extracts the Cap-Token option from the request message and obtains the capability token from the option.

Meanwhile, as a capability token is usually large in size, the Cap-Token Option is proposed with a compression mechanism to minimize the size of the option. The compression mechanism elides the option fields of which the contents have a duplicate in other CoAP fields or in the lower-layer protocols of the request message. The receiver relies on the duplicates in the request message to restore those elided option fields and "decompress" the Cap-Token option.

3.1 Option Fields

The Cap-Token option is composed of several option fields which store the contents of a capability token. It also has a length option field to indicate the size of the option.

Figure 2 shows the option fields of the Cap-Token option. The first byte indicates the difference between the option number of the Cap-Token option and that of the previous option. We define the option number of the Cap-Token option as 9, which implies this option is a critical option and is safe-to-forward for a proxy (Shelby et al., 2014). The second byte is an extension field indicating the size of the option. The third byte is the "Dashboard" option field, which encodes the format of the option fields in the Cap-Token option. The fourth byte stores the value of `TI`. The next four fields seriatim store the value of `II` (4 bytes), `SI` (16 bytes), `OI` (16 bytes), and `IT` (8 bytes) of the capability token. IP addresses are converted to the network order before being stored. After these four fields, the Activation Time field stores the time difference between `IT` and `NB` of the capability token. The Validity Time field indicates how long the capability token is valid, i.e., the time difference between `NA` and `NB`. Both fields are unsigned integers and maximum 4 bytes in length. After the Validity Time field, the issuer signature is stored. The Permission List field is placed at the end of the Cap-Token option, because the length of the permission list varies by its content.

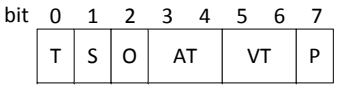

Figure 3: The bit layout of the Dashboard field.

3.2 Compression Mechanism

In the Cap-Token option, the compression mechanism elides the option fields of which the contents have a duplicate in other part of the request message. It leverages the Dashboard field to indicate whether an option field is compressed in the Cap-Token option. Figure 3 shows the bit layout of the Dashboard field.

- **Token Sub-Identity** (T) (1 bit) indicates whether the compression elides the Token Sub-Identity field in the Cap-Token option. The Token Sub-Identity field can be elided only when its value equals to 0. If $T = 1$, the Token Sub-Identity field is carried in-line. Otherwise, the Token Sub-Identity field is elided and the receiver restores this field by assigning its value to 0.
- **Subject Identity** (S) (1 bit) indicates whether the compression elides the Subject Identity field in the Cap-Token option. If $S = 1$, the Subject Identity field is carried in-line. Otherwise, the Subject Identity field is elided and the receiver uses the source IP address of the request message to restore this field.
- **Object Identity** (O) (1 bit) indicates whether the compression elides the Object Identity field in the Cap-Token option. If $O = 1$, the Object Identity field is carried in-line. Otherwise, the Object Identity field is elided and the receiver uses the destination IP address of the request message to restore this field.
- **Activation Time** (AT) (2 bits) indicates how many bytes (bit value plus 1) are used to present the Activation Time field. For example, if $AT = 1$,

two bytes are used to present the Activation Time field.

- **Validity Time** (*VT*) (2 bits) indicates how many bytes (bit value plus 1) are used to present the Validity Time field.
- **Permission List** (*P*) (1 bit) indicates whether the Permission List field is compressed. If $P = 0$, the field is compressed; otherwise, no compression is applied.

In the Permission List field, when $P = 0$, the compression mechanism elides the contents which have a duplicate in other CoAP fields. For a valid CoAP request message, the resource path and request method indicated by the CoAP header (more precisely, indicated by the *Uri-Path* options and the *Code* field of CoAP) must have a duplicate in the Permission List field. So, the compression mechanism elides the duplicated resource path and request method in the Permission List field, and lets the receiver to restore the permission list from the contents stored in the CoAP header.

Algorithm 1 shows the pseudo code for compressing a permission list. In the pseudo code, *rPath* and *rMethod* are the resource path and request method indicated by the CoAP header. *pList* is the permission list of a capability token, which stores a set of access permissions. The compression mechanism first searches *pList* to find the access permission which has the same resource path (`RP`) and request method (`RM`) as *rPath* and *rMethod*. If the permission is found, the matched resource path and request method in that permission are elided. Using the permission list shown in Figure 1 as an example, when sending a "*GET*" request message to obtain "temperature" data, we elide the indicated resource path and request method as well as elide the key part of a key-value pair in JSON if its associated value is elided. The permission list becomes "[{}]" after compression. The result is serialized and stored in the Permission List field.

Algorithm 1: Compressing a permission list.

```
 1: procedure CompressPermissionList
 2:     rPath ← path of the requested resource
 3:     rMethod ← the request method
 4:     pList ← permission list of a capability token
 5:
 6:     for each perm in pList do
 7:         if rPath matches the RP of perm then
 8:             if rMethod has a match in RM of perm then
 9:                 Elide the rPath from perm
10:                 Elide the rMethod from perm
11:                 break
12:             end if
13:         end if
14:     end for
15: end procedure
```

4 EVALUATION

In this section we evaluate the performance of the Cap-Token option and the compression mechanism. We first use an example to elaborate how many bytes could be saved when compressing the Cap-Token option. Next, we study the overhead in terms of the increase in the IP datagram size of a CoAP request message when applying CBAC to an IoT network.

4.1 Benefit of Compression

We use the capability token shown in Figure 1 as an example to elaborate how many bytes could be saved when compressing the Cap-Token option.

If compression is not applied, all option fields shown in Figure 2 are kept in the Cap-Token option. The uncompressed option requires 32 bytes for storing source and destination IP addresses, and 8 bytes for recording the activation time and validity time. The value of the permission list is serialized before being stored in the option field and the resulting string is 35 bytes in length. The overall size of the uncompressed Cap-Token option is 115 bytes.

When the compression mechanism is applied, both Subject Identity field and Object Identity field are elided. Since `TI` equals to 0, the Token Sub-Identity field can also be elided. The sizes of the Activation Time field and Validity Time field are reduced to 1 byte, because the difference between `IT` and `NB` and the difference between `NA` and `NB` are zero. Moreover, we assume the resource path and request method described in `PL` have a duplicate in other CoAP fields, so both are elided. The compression result of the permission list is "[{}]", which is 4 bytes in length after serialization. The final size of the compressed Cap-Token option is 45 bytes.

Table 1 shows the comparison in terms of the size between an uncompressed Cap-Token option and a compressed Cap-Token option. Our compression mechanism reduces 70 bytes when encapsulating a capability token into the option. 60% of the original size of the option is saved. Among those saved data, 31 bytes are reduced from compressing the Permission List field. The rest 39 bytes are saved from compressing other fields of the Cap-Token option. Such a result shows that our compression mechanism can minimize the size of the Cap-Token option, resulting in a smaller IP datagram size.

Table 1: Comparison of the size of the Cap-Token option before and after compression.

Option Field	Uncompressed Option	Compressed Option
Option Header	2	2
Token Dashboard	1	1
Token Sub-Identity	1	0
Issuer Identity	4	4
Subject Identity	16	0
Object Identity	16	0
Issued Time	4	4
Activation Time	4	1
Validity Time	4	1
Issuer Signature	28	28
Permission List	35	4
Total (bytes)	115	45

4.2 Access Control Overhead

We define the overhead of applying an access control model for an IoT network as the increase in the IP datagram size in percentage. Equation 1 shows how the overhead being calculated.

$$overhead = \frac{(new\_size - original\_size)}{original\_size} \times 100\% \quad (1)$$

Here, we compare the overheads in three different cases, which are: (1) directly attaching a capability token at the end of CoAP, (2) adding an uncompressed Cap-Token option, and (3) adding a compressed Cap-Token option. The first and third cases are used to compare the overhead of our Cap-Token option with the overhead of the capability token transmission approach proposed by previous work (Gusmeroli et al., 2013) (Hernández-Ramos et al., 2013). The second and third cases are used to demonstrate how the compression mechanism reduces the overhead.

We calculate the overheads based on sending a CoAP "GET" request message to obtain data from the "*/temperature*" path of a sensor device. The capability token shown in Figure 1 is used for access control and is sent with the request message. We also assume that the Datagram Transport Layer Security (DTLS) (Rescorla and Modadugu, 2012)) is adopted to secure the transmission of CoAP messages. The DTLS adopts AES-GCM (Dworkin, 2007) as the cryptography algorithm, which uses a 128-bit-long key for encryption and generates an 8-byte-long integrity check value (ICV).

Table 2 shows the IP datagram of the CoAP request message if no access control is applied. At the network layer, a LOWPAN_IPHC header (Hui and Thubert, 2011) is adopted to compress the IPv6 header, which results in 7 bytes in length. An 8-byte-long Hop-by-Hop Options extension header, which contains a RPL option (Hui and Vasseur, 2012), is also included to support storing-mode routing in a RPL network (Winter et al., 2012). At the transport layer, the UDP header is compressed to 4 bytes according to the definition in (Montenegro et al., 2007). The DTLS header contains a 13-byte-long record header and an 8-byte-long initialization vector. In the DTLS payload, the CoAP header occupies 5 bytes[2]. The *Uri-Host* and *Uri-Port* options are set to default, so both options are not present. The *Uri-Path* option occupies 12 bytes to specify the path to the resource. The contents of the DTLS payload are encrypted by AES-GCM and are followed by an 8-byte-long ICV. Summarily, the overall IP datagram size is 65 bytes when CBAC is not applied.

Table 2: The IP datagram of the CoAP request message without any access control protection.

Layer	Header or Payload		Size (byte)
Network	Compressed IPv6 header		7
	Hop-by-Hop Options		8
Transport	Compressed UDP header		4
	DTLS header	Record header	13
		Initialization vector	8
	DTLS payload	CoAP header	5
		Uri-Path option	12
	DTLS ICV		8
Total size			65

When applying CBAC for an IoT network, it requires every request message being transmitted with a capability token. If we directly attache the capability token at the end of the CoAP, the content of the capability token is serialized and the result is attached behind the *Uri-Path* option. Because the *Uri-Path* option has its own length field, unlike attaching behind CoAP payload, the receiver can still retrieve the capability token from the message. In such a case, the serialized capability token is 198 bytes in length. The overhead of directly attaching a capability token at the end of CoAP is 304%.

On the other hand, encapsulating a capability token into the Cap-Token option can reduce the increase in the IP datagram size, because most special characters used by JSON (such as comma, colon, and quotation marks) are removed during the encapsulation process. After embedding an uncompressed Cap-Token option into the CoAP request message, the in-

[2]We assume the CoAP header uses 1 byte for *Version* and *Type*, 1 byte for *Code* field, 2 bytes for *Message ID* field, and 1 byte for *Token* field. The last field is used for matching a response message with a request message.

Table 3: Comparison of the IP datagram sizes and overheads in difference cases.

	IP datagram Size	Overhead
Case 0	65 bytes	-
Case 1	263 bytes	304 %
Case 2	180 bytes	176 %
Case 3	110 bytes	69 %

* Case 0: no access control protection.
* Case 1: attaching a capability token.
* Case 2: uncompressed Cap-Token option.
* Case 3: compressed Cap-Token option.

crease in the IP datagram size reduces to 115 bytes. The overhead decreases to 176%. Compressing the Cap-Token option further reduces the increase in the IP datagram size. As shown in our previous analysis, the compressed Cap-Token option is 45 bytes in length. The overhead becomes 69% after compressing the Cap-Token option.

Table 3 compares the IP datagram sizes and overheads in different cases. The results show that attaching a capability token at the end of CoAP not only suffers the payload ambiguity problem, it also generates the largest IP datagram, which is 263 bytes. Comparatively, using the Cap-Token option to encapsulate a capability token reduces the IP datagram size by 83 bytes. When the compression mechanism is applied, the IP datagram size further reduces to 110 bytes, which is only 41% of the IP datagram size when attaching a capability token at the end of CoAP. The overhead is reduced by 235%. This demonstrates that our Cap-Token option and the compression mechanism not only solve the payload ambiguity problem, they also reduce the IP datagram size.

5 DISCUSSION

A benefit of using the compressed Cap-Token option is that it results in fewer data link frames when transmitting the CoAP request message over a 6LoWPAN network. Because the maximum frame size defined in IEEE 802.15.4 is 127 bytes, a large IP datagram will be fragmented into several data link frames before transmission. Since an intermediate router needs to collect all data link frames to determine the next forwarding target. Losing any data link frame means the entire message is lost. Hence, generating fewer data link frames for a request message in a 6LoWPAN network would result in a better success rate on delivering request messages.

Based on our results in the previous section as an example, directly attaching a capability token at the end of CoAP would generates 3 data link frames. Although using an uncompressed Cap-Token option reduces the IP datagram size, the size is still larger than 127 bytes. So, the request message is fragmented into 2 data link frames. Only when using a compressed Cap-Token option, the IP datagram can be fitted in one IEEE 802.15.4 frame. The generated data link frames of the above three cases are shown in Table A1, A2, and A3 in Appendix, respectively. Thus we conclude that using the Cap-Token option and compression mechanism can reduce the chance of the occurrence of the link-layer fragmentation when sending CoAP request messages. It also improves the success rate of delivering request messages over a 6LoWPAN network.

6 RELATED WORK

Many researchers have adopted the CBAC model for an IoT network. Mahalle et al. (Mahalle et al., 2012a), (Mahalle et al., 2012b) treated a capability token as a permission for setting up a connection with an IoT device. However, their design doesn't send every CoAP request message with a capability token, breaking the stateless rule of REST. Besides, no fine-grained access permission is defined in the capability token to control the access of resources hosted on the target device. Gusmeroli et al. (Gusmeroli et al., 2013) proposed a design for capability tokens to support the principle of least privilege. However, their design relies on a centralized authorization server to decide on the authorization. This makes their design hard to be deployed in an IoT network where a centralized authorization server is hard to deploy. In the work done by Hernández-Ramos et al. (Hernández-Ramos et al., 2013), (Hernández-Ramos et al., 2014), an IoT device can decide on the authorization by itself without the intervention of a centralized authorization server. However, when sending a CoAP request message, their design directly attaches a capability token behind the CoAP payload. This causes a receiver hard to retrieve the capability token from the request message, because CoAP does not define a length field to indicate its payload size. To prevent a capability token from being mixed up with the contents of CoAP, both Seitz et al. (Seitz et al., 2013) and Pereira et al. (Pereira et al., 2014) proposed to use a CoAP option to encapsulate a capability token. But, no detail design of the option is mentioned in their paper. Hence, we can conclude that none of existing work has provided a good solution for the "payload ambiguity problem" and considered how to reduce the IP datagram size after applying CBAC for an IoT network.

7 CONCLUSION

In this paper, we propose the Cap-Token option to solve the payload ambiguity problem when applying the capability-based access control model for an IoT network. Because the Cap-Token option is independent form other CoAP fields and the option knows its size, we can prevent a capability token from being mixed up with the contents of other CoAP fields. Besides, we also propose a compression mechanism to reduce the size of the Cap-Token option. Our evaluation shows that the compression mechanism can save the size of the Cap-Token option by 60%. This also helps decrease the overhead (the increase in the IP datagram size) after applying CBAC for an IoT network. Our results show that the overhead of adding a compressed Cap-Token option to a request message is only 69%, while adding an uncompressed Cap-Token option is 176% and directly attaching a capability to the end of CoAP is 304%. The smaller IP datagram size also helps generate fewer data link frame when sending a CoAP request message over a 6LoWPAN network.

Our future work focuses on solving the storage consumption problem on the requester's device. Because the current design only allows a capability token to be used to access a particular device, a requester has to acquire at least N capability tokens if attempting to access the resources hosted on N IoT devices. This may consume a large amount of storage space in a large IoT use case, such as smart city. Hence, we plan to design a new type of capability token to tackle this problem.

REFERENCES

Alghamdi, T., Lasebae, A., and Aiash, M. (2013). Security Analysis of the Constrained Application Protocol in the Internet of Things. In *Second International Conference on Future Generation Communication Technology*, pages 163–168.

Dworkin, M. (2007). Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC. Technical report, National Institute of Standards and Technology.

Granjal, J., Monteiro, E., and Sa Silva, J. (2015). Security for the Internet of Things: A Survey of Existing Protocols and Open Research Issues. *IEEE Communications Surveys Tutorials*, 17(3):1294–1312.

Gusmeroli, S., Piccione, S., and Rotondi, D. (2013). A Capability-based Security Approach to Manage Access Control in the Internet of Things. *Mathematical and Computer Modelling*, 58:1189–1205.

Hernández, J. L., Moreno, M. V., Jara, A. J., and Skarmeta, A. F. (2014). A Soft Computing Based Location-aware Access Control for Smart Buildings. *Soft Computing*, 18(9):1659–1674.

Hernández-Ramos, J. L., Jara, A. J., Marín, L., and Gómez, A. F. S. (2014). DCapBAC: Embedding Authorization Logic into Smart Things through ECC Optimizations. *International Journal of Computer Mathematics*, 0:1–22.

Hernández-Ramos, J. L., Jara, A. J., Marín, L., and Skarmeta, A. F. (2013). Distributed Capability-based Access Control for the Internet of Things. *Journal of Internet Services and Information Security*, 3:1–16.

Hui, J. W. and Thubert, P. (2011). Compression Format for IPv6 Datagrams over IEEE 802.15.4-Based Networks. RFC 6282.

Hui, J. W. and Vasseur, J. (2012). The Routing Protocol for Low-Power and Lossy Networks (RPL) Option for Carrying RPL Information in Data-Plane Datagrams. RFC 6553.

Mahalle, P., Anggorojati, B., Prasad, N., and Prasad, R. (2012a). Identity Driven Capability based Access Control (ICAC) Scheme for the Internet of Things. In *2012 IEEE International Conference on Advanced Networks and Telecommunications Systems*, pages 49–54.

Mahalle, P., Anggorojati, B., Prasad, N., and Prasad, R. (2012b). Identity Establishment and Capability based Access Control (IECAC) Scheme for Internet of Things. In *15th International Symposium on Wireless Personal Multimedia Communications (WPMC)*, pages 187–191.

Montenegro, G., Kushalnagar, N., Hui, J. W., and Culler, D. E. (2007). Transmission of IPv6 Packets over IEEE 802.15.4 Networks. RFC 4944.

Nguyen, K. T., Laurent, M., and Oualha, N. (2015). Survey on Secure Communication Protocols for the Internet of Things. *Ad Hoc Networks*, 32:17–31.

Pereira, P., Eliasson, J., and Delsing, J. (2014). An Authentication and Access Control Framework for CoAP-based Internet of Things. In *40th Annual Conference of the IEEE Industrial Electronics Society*, pages 5293–5299.

Rescorla, E. and Modadugu, N. (2012). Datagram Transport Layer Security Version 1.2. RFC 6347.

Roman, R., Zhou, J., and Lopez, J. (2013). On the Features and Challenges of Security and Privacy in Distributed Internet of Things. *Computer Networks*, 57(10):2266–2279.

Seitz, L. and Gerdes, S. (2015). Use Cases for Authentication and Authorization in Constrained Environments. IETF Draft.

Seitz, L., Selander, G., and Gehrmann, C. (2013). Authorization Framework for the Internet-of-Things. In *14th IEEE International Symposium and Workshops on a World of Wireless, Mobile and Multimedia Networks*, pages 1–6.

Shelby, Z., Hartke, K., and Bormann, C. (2014). The Constrained Application Protocol (CoAP). RFC 7252.

Winter, T., Thubert, P., Brandt, A., Hui, J. W., Kelsey, R., Levis, P., Pister, K., Struik, R., Vasseur, J., and

Alexander, R. K. (2012). RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks. RFC 6550.

APPENDIX

Here we show how many data link frames are generated when sending a CoAP request message with a capability token and (1) directly attaching the capability token at the end of CoAP (Table A1), (2) encapsulating the capability token into the Cap-Token option (Table A2), and (3) encapsulating the capability token into the Cap-Token option and compressing the option (Table A3). We assume that the link-layer header occupies 9 bytes, which includes the Frame Check Sequence. PAN ID is elided and short MAC addresses are used. If the link-layer fragmentation occurs, additional fragmentation header would be added to the adaptation layer, which occupies 4 bytes in the first frame and occupies 5 bytes in the succeeding frames.

Table A1: The data link frames of the CoAP request message when attaching a capability token at the end of CoAP.

1^{st} data link frame			
Layer	Header or Payload		Size (byte)
Data Link	IEEE 802.15.4 header		9
Adaptation	Fragmentation header		4
Network	Compressed IPv6 header		7
	Hop-by-Hop Options		8
Transport	Compressed UDP header		4
	DTLS header	Record header	13
		Initialization vector	8
	DTLS payload*		74
Total size			127

2^{nd} data link frame		
Layer	Header or Payload	Size (byte)
Data Link	IEEE 802.15.4 header	9
Adaptation	Fragmentation header	5
Transport	DTLS payload*	113
Total size		127

3^{rd} data link frame		
Layer	Header or Payload	Size (byte)
Data Link	IEEE 802.15.4 header	9
Adaptation	Fragmentation header	5
Transport	DTLS payload*	28
	DTLS ICV	8
Total size		50

* The DTLS payload is composed of the CoAP header, the *Uri-Path* option, and the serialized capability token.

Table A2: The data link frames of the CoAP request message when using an uncompressed Cap-Token option.

1^{st} data link frame			
Layer	Header or Payload		Size (byte)
Data Link	IEEE 802.15.4 header		9
Adaptation	Fragmentation header		4
Network	Compressed IPv6 header		7
	Hop-by-Hop Options		8
Transport	Compressed UDP header		4
	DTLS header	Record header	13
		Initialization vector	8
	DTLS payload*		74
Total size			127

2^{nd} data link frame		
Layer	Header or Payload	Size (byte)
Data Link	IEEE 802.15.4 header	9
Adaptation	Fragmentation header	5
Transport	DTLS payload*	58
	DTLS ICV	8
Total size		80

* The DTLS payload is composed of the CoAP header, the *Uri-Path* option, and the uncompressed Cap-Token option.

Table A3: The data link frame of the CoAP request message when using a compressed Cap-Token option.

1^{st} data link frame			
Layer	Header or Payload		Size (byte)
Data Link	IEEE 802.15.4 header		9
Network	Compressed IPv6 header		7
	Hop-by-Hop Options		8
Transport	Compressed UDP header		4
	DTLS header	Record header	13
		Initialization vector	8
	DTLS payload	CoAP header	5
		Uri-Path option	12
		Cap-Token option	45
	DTLS ICV		8
Total size			119