Efficient Protection of Response Messages in DTLS-Based Secure Multicast Communication

Marco Tiloca
SICS Swedish ICT AB, Security Lab
Isafjordsgatan 22, Kista, Sweden
Phone: +46 070 6046501 Fax: +46 8 9751 7230
marco@sics.se

ABSTRACT

DTLS is a standardized security protocol designed to provide end-to-end secure communication among two peers, and particularly considered for the emerging Internet of Things. In order to protect group communication, the IETF is currently working on a method to secure multicast messages through the same DTLS security services. However, such an approach relies on traditional DTLS sessions to protect unicast responses to multicast messages. This increases the amount of security material stored by group members and can have a relevant impact on network performance. In this paper we propose an extension to the IETF approach which allows to efficiently protect group responses by reusing the same group key material. Our proposal does not require to establish additional DTLS sessions, thus preserving high communication performance within the group and limiting storage overhead on group members. Furthermore, we discuss a suitable key management policy to provision and renew group key material.

Categories and Subject Descriptors

C.2.0 [**Computer-Communication Networks**]: General—*Security and protection*

General Terms

SECURITY

Keywords

Security, DTLS, Multicast, Group communication

1. INTRODUCTION

The *Datagram Transport Layer Security* (*DTLS*) protocol [7] has been introduced by the IETF to provide secure communication between two peers, in the presence of unreliable datagram protocols such as UDP [10]. Although it has been standardized only recently, DTLS is already asserting itself as the *de-facto* security protocol to protect end-to-end communication in the emerging *Internet of Things* (*IoT*) [14][15][20]. DTLS has been designed to be as much similar as possible to the widely adopted *TLS* protocol [18], and provides equivalent security services, i.e. it allows client and server applications to communicate with one another preventing eavesdropping, tampering, and message forgery. In order to establish a DTLS session, two peers perform a preliminary message exchange known as *handshake*, so agreeing on a ciphersuite and establishing common security material.

This work was carried out during the tenure of an ERCIM "Alain Bensoussan" Fellowship Programme. The research leading to these results has received funding from the *European Union* Seventh Framework Programme (*FP7/2007-2013*) under *grant agreement* n° 246016.

More recently, the lighthweight *Constrained Application Protocol* (*CoAP*) [22] developed by the IETF working group *CoRE* [1] has specifically mandated the adoption of DTLS to provide secure communication in machine-to-machine applications and constrained networks. Furthermore, the *CoRE* working group has also considered application scenarios relying on *group communication*, and is currently defining how to use the *CoAP* protocol on top of IP multicast [4]. This is particularly desirable and convenient in the presence of *Low-Power* and *Lossy Networks* (*LLNs*), which are typically composed of low-power and resource constrained devices featuring scarce computing power and limited battery life.

Of course, such application scenarios may also require the presence of reliable and efficient security services, in order to assure secure multicast communication within a group. This is the reason why another IETF working group, namely *DICE* [2], is currently defining how to adapt the DTLS protocol to protect multicast messages sent by a *sender* node and received by multiple *listener* nodes in the same group [17]. Such an approach has the main goal of protecting multicast group communication through the usual DTLS security services. In addition, it relies on a single set of group key material shared among all group members, and does not require to perform the complex and costly DTLS handshake in order to establish a multicast secure session.

The approach described in [17] considers also the protection of *group responses* sent by *listener* nodes as unicast replies to multicast messages. That is, group responses are supposed to be protected through distinct unicast DTLS sessions, established between *sender* nodes and replying *listener* nodes in the group. This may require to perform a significant number of DTLS handshakes, and may be not reasonable in case unicast secure communication is required only to protect group responses. Also, such an approach is likely to be inefficient and even not feasible, especially in the presence of low-power constrained devices, due to the complex DTLS handshake execution, and the resulting computing and communication overhead. Finally, group members would be required to store a considerable amount of supplementary security material in order to maintain additional, possibly unnecessary, unicast DTLS sessions.

In this paper, we propose an extension to the DTLS-based multicast communication method under development by the IETF, and describe how to efficiently protect group responses according to the same group communication scheme and by reusing the same group key material. By doing so, it is not necessary to establish any unicast DTLS session, hence avoiding to perform additional and costly DTLS handshakes, and preserving high communication performance within the group. In addition, our extension does not require group members to store and maintain any additional security material, thus not affecting them in terms of storage overhead.

To the best of our knowledge, this paper represents the first contribution aimed at improving efficiency of group response protection in DTLS-based group communication. Also, our approach does not break current standards, and thus is suitable for a possible integration in the current multicast scheme proposed by the IETF. As a further contribution, the paper describes a suitable key management policy to address provisioning and renewal of group key material, in the presence of DTLS-based multicast communication.

The rest of the paper is organized as follows. Section 2 briefly overviews the main features of DTLS, while Section 3 describes the DTLS-based multicast communication method proposed by the IETF. In Section 4, we present our extension for reusing group key material to protect group responses. Section 5 discusses a key management policy suitable to the considered communication scenario. Finally, in Section 6 we draw our conclusive remarks.

2. DTLS OVERVIEW

The *Datagram Transport Layer Security* (*DTLS*) protocol [7] has been designed to provide secure communication for applications based on unreliable datagram protocols, such as UDP [10]. DTLS is based on the *TLS* protocol [18], and provides equivalent security services, i.e. it allows client and server applications to communicate with one another preventing eavesdropping, tampering, and message forgery. In order to address the unreliable nature of datagram transport protocols, DTLS introduces a few minor changes with respect to TLS, such as making distinct messages *independent* from one another, and not forcing to terminate active sessions upon receiving invalid messages. Also, DTLS provides an optional stateless Cookie mechanism, in order to counteract possible *Denial of Service* (*DoS*) attacks against DTLS servers.

Type	Version	Epoch	Sequence Number	Length
1 Byte	2 Bytes	2 Bytes	6 Bytes	2 Bytes

Figure 1: DTLS record header.

Messages are transmitted as a sequence of DTLS *records*, each one of which includes the header shown in Figure 1. The *Type* and *Version* fields indicate the referred higher level protocol and the adopted version of DTLS, respectively, while the *Length* field represents the size in bytes of the actual application data conveyed in the record. Finally, with respect to TLS, two additional fields have been included, namely *Epoch* and *Sequence Number*. Specifically, the *Epoch* value is incremented upon changing the currently used security protocols and material, while the *Sequence Number* value is incremented for every new message transmitted by the same peer over the same DTLS session. The concatenation of the *Epoch* and *Sequence Number* fields is considered as a 64 bit fresh value, namely *explicit nonce*, and is used to compute a Message Authentication Code (MAC) to provide integrity of DTLS records.

For each secure session established, a DTLS peer maintains a *write connection state* and a *read connection state*. Practically, write (read) connection states are operating environments referred while processing outgoing (incoming) messages within a DTLS session. Among other information, connection states define a given DTLS peer as either *client* or *server*, and include current *Epoch* and *Sequence Number* values, as well as algorithms and security material used throughout the session.

A DTLS client and server establish a secure session through a specific *handshake* process. This is necessary in order to agree on a common set of security algorithms and derive the security material used to protect DTLS records. In particular, DTLS client and server generate and exchange two random values, i.e. *client random* and *server random*, and come to agree on a *premaster secret*. The latter is in turn composed of a 46 bytes *random* field, whose value is generated at random, and a 2 bytes *client version* field. Then, the *premaster secret* is used together with *client random* and *server random* by the two peers, in order to derive the actual security material for the DTLS session.

However, performing a DTLS handhshake is a complex, time consuming, and resource demanding process. In fact, it results in at least two message round trips, requires to perform a number of cryptographic operations, and assumes that DTLS client and server have been already provided with some preliminary security material. In particular, two main approaches to provide preinstalled cryptographic keys are admitted. The first one is based on *asymmetric key* pairs, while the second one relies on *symmetric keys* pre-shared between client and server and avoids sending/receiving public certificates and performing costly public key operations.

3. DTLS-BASED MULTICAST SECURITY

The IETF is currently working on readapting DTLS to secure *group communication*, with particular reference to *Low-Power* and *Lossy Networks* (*LLNs*) and the lightweight application protocol *CoAP* [4][22]. Specifically, the IETF working group *DICE* [2] is defining how to adapt the usual DTLS record security services in order to protect multicast messages sent by a *sender* node and received by multiple *listener* nodes in the same group [17]. This section summarizes the main aspects of such a proposal.

In the following, we consider a multicast group G associated to a unique IP address IP_G, which has been previously allocated for IP multicast communication. All group members trust one another, and are not supposed to tamper with multicast messages sent within the group. In particular, members of group G can belong to two possible categories, namely *senders* and *listeners*, and can have both roles depending on their actual network activity in the group. *Sender* nodes are responsible for transmitting multicast messages to group G, while *listener* nodes are explicitly interested in receiving multicast messages. In principle, any node can become a *listener*, by registering with a network routing device, and signaling its intent to receive messages sent to the IP_G address. Instead, *senders* nodes are not supposed to be aware of the group membership or get notified of new *listener* nodes.

Also, members of group G maintain separate *group connection* *states* CS^W and CS^R, referred while processing outgoing or incoming multicast messages within the group, respectively. In particular, group connection states are configured in such a way that *sender* nodes act as *servers*, while *listener* nodes act as *clients*. Every *sender* node in the group initializes to 0 the *Sequence Number* value in its own *write group connection state* CS^W, and increments it for every multicast DTLS record sent to group G.

Finally, an additional entity named *Group Controller* (*GC*) is responsible for creating the group, managing the actual join process to add new group members, and providing them with the commonly shared group security material. The *GC* is not required to be also an actual *sender* within the group, and can be discovered by joining nodes through various methods, such as *DNS-SD* [16] and *Resource Directory* [21].

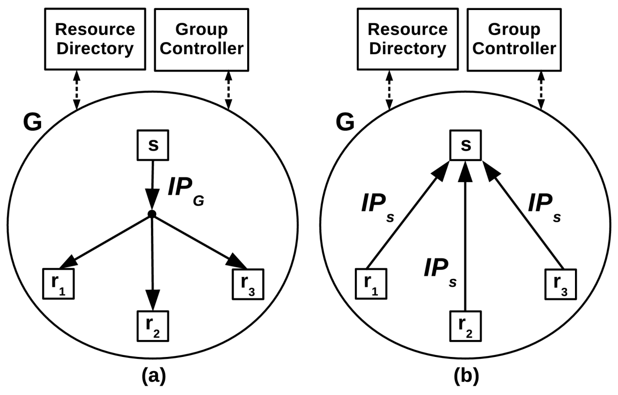

Figure 2: Group communication scenario: (a) single multicast message; (b) multiple unicast responses.

Figure 2 shows a group *G* composed of one *sender* node s and three *listener* nodes r_1, r_2, and r_3. Group messages are sent by the *sender* node s as a single multicast transmission (Figure 2:a), while possible responses by individual *listener* nodes are sent back as unicast messages (Figure 2:b). This is consistent with the CoAP group communication guidelines provided in [4].

3.1 Protection of multicast messages

As described in [17], a DTLS group session is established without performing a regular handshake process. Instead, the *GC* securely provides all group members with a common *Group Security Association* (*GSA*) [19], including the adopted ciphersuites and a set of preliminary security parameters. Specifically, all group members are provided by the *GC* with the same *GSA*, and use it to autonomously derive the same DTLS *SecurityParameters* structure as defined in [18]. Besides, the *GSA* includes also the DTLS *premaster secret*, the *client random* value and the *server random* value, usually generated while performing a DTLS handshake. Then, every group member relies on such three pieces of information in order to derive the actual group key material reported below. Note that, as an alternative, the *GC* may include the six key material elements directly in the *GSA*, upon providing it to group members.

```
client write MAC key
server write MAC key
client write encryption key
server write encryption key
client write IV
server write IV
```

Such group key material is thus associated to the *GSA*, and secretly shared between all group members. Then, *sender* nodes can use it to protect multicast messages by means of usual DTLS security services. Specifically, before transmitting a multicast message M, a *sender* node protects it by using the group *server write* parameters. Then, every *listener* node uses the same *server write* parameters to process message M upon its reception. The exact usage of the *server write* parameters depends on the specific ciphersuite adopted by group members [17]. On the other hand, at the moment it is not defined any possible usage of the *client write* parameters, either by *sender* nodes, or by *listener* nodes. Finally, note that, since all group members share the same group security material, it is not possible to assure *source authenticity* of multicast messages sent within the group.

As mentioned in Section 2, the message authenticity process relies on a 64 bit *explicit nonce*, i.e the concatenation of the *Epoch* field and the *Sequence Number* field in the DTLS record header. However, the presence of multiple *sender* nodes inevitably leads to reusing the same *explicit nonce* values, and makes it impossible to check that received multicast messages are actually fresh. As a consequence, multicast messages sent by different *sender* nodes may be considered as replayed messages by *listener* nodes, and get discarded. Furthermore, the commonly adopted ciphersuite MTS_WITH_AES_128_CCM_8 requires that the *explicit nonce* value is different for each distinct invocation of the encryption function using the same key material. Hence, DTLS sequence number values maintained by multiple *sender* nodes in their respective *write group connection state* CS^W would need to be synchronized, in order to avoid their reuse.

Since synchronizing sequence number spaces within a group can definitely be a difficult task, [17] proposes to separate sequence number spaces of different *sender* nodes, and then embed a unique sender identifier in the original *Sequence Number* field of the DTLS record header, as suggested in [11]. Then, the *GC* is required to additionally assign a unique *SenderID* to each *sender* node in the group, and provide all group members with a list of active *SenderIDs*. Also, every *listener* node maintains a distinct *read group connection state* CS_s^R for every *sender* node s, identified by the associated *SenderID*. Finally, the original DTLS record header is adapted accordingly, as reported in Figure 3. In particular, the original 6 octet *Sequence Number* field in the DTLS record header has been redefined as a 1 octet *SenderID* field followed by a 5 octet *Truncated Sequence Number* field.

Type	Version	Epoch	SenderID	Truncated Sequence Number	Length
1 Byte	2 Bytes	2 Bytes	1 Byte	5 Bytes	2 Bytes

Figure 3: DTLS record header for multicast messages.

Upon receiving a multicast message M, every *listener* node retrieves i) the destination address IP_G as well as the destination port P_G; and ii) the *sender* identifier from the *SenderID* field of the DTLS record header. Then, it uses such pieces of information to retrieve the *GSA* associated to group *G* and the *read group connection state* CS_s^R referred to the *sender* node s. Finally, it retrieves the group *server write* parameters, and uses them to process message M. The message freshness is verified by checking the *Epoch* and *Truncated Sequence Number* field with the values stored in the *read group connection state* CS_s^R.

Note that *listener* nodes which lately join the group do not know the *Epoch* and *Truncated Sequence Number* currently used by different *sender* nodes. Hence, when such *listener* joining nodes receive a multicast message, they may be not able to verify if it is fresh and has not been replayed by an adversary. In order to overcome this issue, [17] suggests to adopt the following approach. Upon receiving a multicast message M from a given *sender* node s for the first time, a late joining *listener* node initializes the *Epoch* and *Truncated Sequence Number* in its own *read group connection state* CS_s^R associated to the *sender* node s. However, message M is discarded, i.e. it is not delivered to the application layer. This provides a reference point to identify if future multicast messages from the same *sender* node s are fresher than the last one received.

4. GROUP RESPONSE PROTECTION

As summarized in Section 3, the approach described in [17] extensively addresses the protection of multicast messages sent by *sender* nodes. However, in many application scenarios, *listener* nodes are supposed to reply to *sender* nodes, by sending back individual response messages. Besides, the latters may need to be secured as well, in order to assure confidentiality, integrity, authenticity, and replay protection. In fact, if such responses to multicast messages were not secured at all, an adversary could access their content as well as tamper with them by forging data and control information, so performing an attack against the whole group.

The current version of [17] takes this issue into account and prescribes that unicast responses to DTLS-based multicast messages *must* be secured. In particular, such unicast responses are supposed to be protected by means of traditional unicast DTLS sessions, established between *sender* nodes and *listener* nodes. However, this may require to perform a significant number of DTLS handshakes within the group, and may be not convenient in case unicast secure communication is required only to protect group responses. In this section, we present an extension to the secure communication scheme described in [17], and propose an approach to efficiently protect group responses, by relying on the same scheme described in Section 3 and reusing the same group security material.

4.1 Motivation

The straightforward approach described in [17] and based on traditional unicast DTLS sessions requires that every *sender* node in the group establishes a unicast DTLS session with each replying *listener* node. Hence, every *sender* node is required to perform a DTLS handshake with all replying *listener* nodes in the group, i.e., in the worst case, with all other group members.

However, this can be not reasonable and convenient, especially if unicast secure communication within the group is required only by *listener* nodes to send group responses to multicast messages. Moreover, such an approach is likely to be inefficient and even not feasible, especially in the presence of constrained low-power devices, due to the complexity displayed by the DTLS handshake execution, and the resulting computing and communication overhead.

In fact, as a first option, every *sender* node should establish a unicast DTLS session with each replying *listener* node *before* sending its first multicast message to the group. This would require every *sender* node to be aware of the current group membership, i.e. to know what *listener* nodes are currently present in the group. Also, every *sender* node should establish further unicast DTLS sessions in case new *listener* nodes join the group. Hence, this would force to provide *sender* nodes with otherwise unnecessary information, as well as to keep them up to date about possible changes in the group membership, upon new nodes' joining. However, [17] explicitly states that applications on group nodes do not know, and do not get notified, when new *listener* nodes join the group.

As a different approach, every *listener* node should establish a unicast DTLS session with a given *sender* node in the group *before* sending back its own first group response message. However, this would result in *sender* nodes possibly *flooded* with a potentially consistent number of DTLS handshakes to be performed.

Note that both the approaches described above also introduce a not negligible delay before the group can become fully operative, due to the execution of a considerable number of DTLS handshakes. Moreover, the observed impact on group members' availability and performance would be even more severe, in case nodes are allowed to be members of more than one multicast group at the same time.

4.2 Protection of response messages

In the following, we describe how *listener* nodes can protect their individual group responses, by reusing the same group security material described in Section 3.1.

Basically, in order to adopt our approach, *sender* nodes must be able to recognize whether a received unicast message is actually a group response sent by a *listener* node, i.e. a reply to a previously sent multicast message. Hence, upon creating the group *G*, the GC provides group nodes also with a unique 1 byte *GroupID* associated to G, namely ID_G. After that, possible new nodes must be provided with the same identifier ID_G upon joining the group. So doing, every group member is able to univocally identify the *GSA* associated to *G*, by means of the associated multicast address IP_G and port P_G, as well as by the *GroupID* ID_G.

In addition, every *sender* node maintains a separate *read group connection state* CS_r^R identified by the unicast address IP_r, for every *listener* node r that has sent at least one group response. Similarly, every *listener* node maintains a separate *write group connection state* CS_s^W, for every *sender* node s in the group, identified by the unique associated *SenderID* ID_s. Each one of such CS_s^W includes a *Sequence Number* value initialized to 0 and incremented for every unicast group response sent to *sender* node s. Finally, we introduce an additional format for the DTLS record header, whose structure is depicted in Figure 4.

Type	Version	Epoch	GroupID	Truncated Sequence Number	Length
1 Byte	2 Bytes	2 Bytes	1 Byte	5 Bytes	2 Bytes

Figure 4: DTLS record header for group response messages.

***Listener* nodes.** Upon receiving a multicast message M sent to the multicast address IP_G by a *sender* node s, a *listener* node r can process its group response R as follows. First, it prepares a DTLS record specifying the actual response data as payload, and referring to the record header structure depicted in Figure 4. In particular, the *listener* node r: i) fills the 5 octet *Truncated Sequence Number* field with the sequence number value stored in the *write group connection state* CS_s^W associated to *sender* node s, before incrementing it; and ii) fills the 1 octet *GroupID* field with the group identifier ID_G. Finally, the *listener* node r protects the group response R by means of the group *client write* parameters, before sending it to the *sender* node s as a unicast message. Note that, in case a *sender* node s and a *listener* node r have previously established a traditional unicast DTLS session, the latter can still be used to send back the response message R in a secure way. Hence, in such a case, the *listener* node r is not required to maintain a *write group connection state* CS_s^W associated to the *sender* node s.

***Sender* nodes.** Upon receiving the group response R, the *sender* node s proceeds as follows. Firstly, it checks if any unicast DTLS session has been previously established with the *listener* node r, which is identified by the unicast source address IP_r retrieved from message R. In case of positive match, the response message R is processed according to the traditional DTLS protocol [7]. Note that, in such a case, the *sender* node s does not need to maintain a *read group connection state* CS_r^R associated to the *listener* node r. Conversely, in case of negative match, the *sender* node s does not yet consider message R to be invalid, and proceeds as follows.

First, the *sender* node s parses the DTLS record header of message R according to the format shown in Figure 4, and retrieves the value reported in the *GroupID* field, i.e. ID_G. Then, s verifies whether it maintains a *GSA* associated to the retrieved ID_G, namely GSA_G. In case of negative match, the *sender* node *s* discards message R. Otherwise, it processes message R by means of the group *client write* parameters associated to GSA_G. Then, it verifies the freshness of message R, by comparing the *Epoch* and *Truncated Sequence Number* field with the values stored in the *read group communication state* CS_r^R associated to the *listener* node r. In case the *read group communication state* CS_r^R cannot be found, the *sender* node s creates it, and initializes *Epoch* and *Truncated Sequence Number* to the same values retrieved from message R. Note that, in case the response message R has been received by a group member which is not configured as a *sender* node, the latter can simply consider R to be invalid, and discard it.

In principle, it is possible that a *sender* node belongs to two or more different multicast groups whose respective *GC* has assigned the same $GroupID$ value. However, the *sender* node is supposed to communicate within each one of such different groups by means of distinct applications. In turn, each one of them would refer to a different port number to handle incoming messages. Hence, upon receiving a protected group response R, a *sender* node can still univocally identify the right *GSA*, by using the $GroupID$ value retrieved from the DTLS header, together with the port number conveyed in the transport protocol header.

Finally, in the presence of our approach, the method proposed to deal with late *listener* joiners described in Section 3 should be used also for late *sender* joiners. In particular, upon receiving a group response R from a given *listener* node r for the first time, a late joining *sender* node creates and initializes a *read group communication state* CS_r^R as described above. However, message R is discarded, i.e. it is not delivered to the application layer. This provides a reference point to identify if future unicast group responses from the same *listener* node *r* are fresher than the last one received.

4.3 Discussion

The main advantage of our approach consists in reusing the same group security material shared by all group members. This makes it possible for *listener* nodes in the group to securely reply to multicast messages without establishing new DTLS sessions. As a consequence, group members are not required to store additional security material, thus considerably limiting their storage overhead. More important, unless unicast secure sessions are specifically required, group members do not need to perform any DTLS handshake. This particularly benefits *sender* nodes, which are not required to establish and maintain a unicast DTLS session with each replying *listener* node in the group. As a consequence, the group can become fully operative with no particular delays, and new joining nodes can instantly participate in multicast group communication. In addition, a considerable processing overhead is avoided on group members, and high communication performance is preserved within the group. To the best of our knowledge, there are currently no publicly available implementations of DTLS adaptations for multicast scenarios. As a future work, we will implement multicast communication support for DTLS, and experimentally evaluate performance in the presence and in the absence of the extension described in Section 4.2.

Furthermore, our approach requires that only a 1 byte group identifier is additionally provided to group members, upon joining the group. Also, it does not require *sender* nodes to be aware of the current group membership, i.e. to know what *listener* nodes are currently present in the group. Therefore, in dynamic application scenarios, it is not necessary to provide group members with the identity of new *listener* nodes when they join the group. Finally, the DTLS record header we propose for group responses (see Figure 4) is only 13 bytes in size, and has the very same structure of the DTLS record header proposed for multicast messages (see Figure 3). Hence, our approach is particularly easy to be integrated with the secure multicast scheme proposed by the IETF [17].

On the other hand, just like the approach presented in [17], our proposal does not provide data source authentication, since group responses to multicast messages are protected by means of the commonly shared group security material. Also, since all group members are potentially able to access and alter such response messages, confidentiality is assured only against an adversary external to the group. However, as discussed in [17], both limitations are practically acceptable in many application scenarios, since members of a multicast group are reasonably assumed to be trusted, and are not prone to tamper with messages from other group members. Furthermore, in many *LLNs* use cases, group members even belong to a common authority and are configured by a trusted commissioner.

In the presence of applications that inevitably require data source authentication, or assume that group members are not supposed to trust one another, application layer solutions such as *digital signatures* can be explicitly adopted. However, digital signatures are well known to be quite honerous from a computation standpoint, not to mention that size of signed messages increases due to the signature appending [3]. For instance, RSA-1024 requires the presence of 128 additional bytes per message, so resulting in a considerable communication overhead. Performance of digital signatures can be ameliorated by relying on *Elliptic Curve Criptography* (*ECC*) [6]. For instance, ECC-160 is roughly an order of magnitude faster than RSA, and results in 40 additional bytes per message, although providing the same level of security. Of course, as a final alternative approach, unicast DTLS sessions can still be adopted to provide source authenticity by protecting group responses altogether.

5. KEY MANAGEMENT

As stated in [17], DTLS-based multicast communication should rely on a secure mechanism aimed at distributing keying material, multicast security policies, and security parameters to the multicast group. Also, the *GC* is indicated as primarily responsible for providing the *GSA* to group members. However, the actual establishment of a *GSA* is not addressed in [17], and will be part of a future IETF activity dedicated to the design of a generic key management scheme, preferably based on requirements and recommendations defined in [9][13][19]. In this section, we propose a possible key management policy to renew the group security material and provide the current *GSA* to new group members upon their joining. In particular, we refer to the IETF multicast scheme described in [17] and the extension we have presented in Section 4.

We recall that the actual group key material is computed from the *SecurityParameters* derived from the *GSA*, with particular reference to the *premaster secret*, the *client random* value and the *server random* value. Hence, group key renewal can be easily performed by securely providing the group members with a new *premaster secret*. After that, the latter can be used to renew the group key material, while all other settings and information in the *GSA*, e.g. the adopted cryptosuite, can remain unchanged. In the following, we consider the *GC* to be an additional member of the multicast group, and to be configured as a *sender* node.

Periodical rekeying. Renewing the group key material in a periodical fashion is recommended in order to discourage an external adversary from performing exhaustive key search or traffic analysis. This assumes that the group has not been compromised, i.e. a possible adversary has not taken any group member under her control. The amount of time between two consecutive occurrences of periodical rekeying should be properly defined considering the application requirements, and the perceived level of threat in the group. Of course, the more frequent the periodical rekeying, the less the damage due to possibly compromised group key material.

In order to perform a periodical rekeying, the *GC* can broadcast a new securely generated *premaster secret* to group members. This can be done by broadcasting a single DTLS multicast message, protected by means of the current *server write* parameters as any other multicast message transmitted to the group. Once it has been received, the new *premaster secret* value is stored in the *GSA* by all group members, and used to renew the current group key material.

Such an approach relies in turn on the commonly shared group key material, and thus, as discussed in Section 4.3, cannot assure source authenticity of rekeying messages. That is, group members cannot be sure that a rekeying message has been actually sent by the *GC*. In order to address this issue, values for the *random* field of new *premaster secrets* can be generated by the *GC* as elements of a *reversed hash chain*, an authentication mechanism derived from Lamport's *one-time password* [12]. The advantage of such an approach is that the most recently released element in the chain can be efficiently authenticated by computing its hash, and verifying that the result is equal to the previously released element in the chain. Therefore, upon creating the multicast group, it is sufficient that the *GC* provides all members with the head element of the hash chain in an authenticated way, e.g. off-line or through a predefined point-to-point authenticated channel. Then, all other chain elements can be automatically and efficiently authenticated by group members.

Nodes' joining. In the presence of a new node joining the group, the rekeying procedure is required to assure *backward security*, whose importance has been explicitly stressed in [13]. Specifically, the joining node must not be able to access any group communication which took place before its joining. As a first step, currently present group members can be rekeyed according to the same periodical rekeying procedure described above. After that, the *GC* can provide the updated *GSA* containing the new *premaster secret* value to the joining node, which can then complete the join process.

In particular, the *GC* should provide the joining node with the *GSA* through a secure communication channel. As a possible approach, upon contacting the *Resource Directory* service to gain knowledge of the *GC* address, the joining node could retrieve a public certificate associated to the *GC*, use the retrieved public key K^{+}_{GC} to establish a secure channel with the *GC*, and securely obtain the *GSA*. As an alternative approach, the joining node can establish a DTLS session with the *GC*, and receive the *GSA* as a sequence of protected DTLS records.

Nodes' leaving. A group member can decide to leave the multicast group, for instance when its mission is concluded or its membership is expired. Also, it can be forced to leave, in case it has been found to be compromised or it is suspected so. Then, remaining group members must be securely provided with updated group key material. This is vital in order to assure *forward security*, whose importance has been explicitly stressed in [13]. Specifically, the leaving node must not be able to access group communication which takes place after its departure from the group. However, the most important and difficult aspect of such a rekeying process is to prevent the leaving node from taking part in the rekeying process itself, i.e. from accessing the new security material during its distribution. In particular, since the leaving node is aware of the current group key material, the latter can not be used to securely distribute a new *premaster secret* to the remaining nodes.

Security considerations in [17] state that the *GC* is assumed to share a different pairwise symmetric key with every member of the group. Although more details are not provided, it is reasonable to assume that such a key is established between the two entities during the join process. Then, upon a node's leaving, the *GC* could rely on such pairwise keys to distribute a new *premaster secret* to all remaining nodes, in a one to one fashion. However, such a unicast approach lacks of efficiency, since it requires a number of rekeying messages which linearly grows with the number of nodes in the group, hence not scaling well with the group size. As an alternative, it is possible to rely on application level schemes for group rekeying, which result to be more efficient and display high scalability with the number of nodes in the group [5][8].

6. CONCLUSION

In this paper, we have considered the DTLS-based multicast communication method recently proposed by the IETF. In particular, we have highlighted that it displays inefficiencies in protecting unicast group responses to multicast messages. Then, we presented an extension to protect group responses according to the same group communication scheme and by reusing the same group security material. Our proposal does not break current standards, and does not require to establish additional unicast DTLS sessions among group members, thus preserving high communication performance within the group and limiting storage overhead on group members. Furthermore, we have described a suitable key management policy to perform provisioning and renewal of group key material. Future work will consist in implementing multicast communication support for the DTLS protocol, and experimentally evaluating performance in the presence and in the absence of our extension.

Acknowledgment
The author would like to thank Christian Gehrmann and Ludwig Seitz for their comments and suggestions. A special thank goes to Sandeep Kumar and Akbar Rahman for the useful discussions.

7. REFERENCES

[1] Constrained RESTful Environments (CoRE). https://datatracker.ietf.org/wg/core/. Last accessed: 16 July 2014.

[2] DTLS In Constrained Environments (DICE). http://datatracker.ietf.org/wg/dice/. Last accessed: 16 July 2014.

[3] A. J. Menezes, P. C. van Oorschot and S. A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, Boca Raton, FL, USA, 2001.

[4] A. Rahman and E. Dijk. *Group Communication for CoAP, draft-ietf-core-groupcomm-21 (Work in progress)*. Internet Engineering Task Force, July 2014.

[5] C. K. Wong, M. Gouda and S. S. Lam. Secure Group Communications Using Key Graphs. *IEEE/ACM Transactions on Networking*, 8(1):16–30, February 2000.

[6] Certicom Research. *Standards for Efficient Cryptography - SEC 1: Elliptic Curve Cryptography*. Certicom Research, May 2009.

[7] E. Rescorla and N. Modadugu. *RFC 6347, Datagram Transport Layer Security Version 1.2*. Internet Engineering Task Force, January 2012.

[8] G. Dini and M. Tiloca. HISS: A HIghly Scalable Scheme for Group Rekeying. *The Computer Journal*, 56(4):508–525, April 2013.

[9] H. Harney, A. Colegrove and G. Gross. *RFC 4535, GSAKMP: Group Secure Association Key Management Protocol*. Internet Engineering Task Force, June 2006.

[10] J. Postel. *RFC 768, User Datagram Protocol*. Internet Engineering Task Force, August 1980.

[11] J. Salowey, A. Choudhury and D. McGrew. *RFC 5288, AES Galois Counter Mode (GCM) Cipher Suites for TLS*. Internet Engineering Task Force, August 2008.

[12] L. Lamport. Password Authentication with Insecure Communication. *Communumications of the ACM*, 24(11):770–772, November 1981.

[13] M. Baugher, R. Canetti, L. Dondeti and F. Lindholm. *RFC 4046, Multicast Security (MSEC) Group Key Management Architecture*. Internet Engineering Task Force, April 2005.

[14] O. Garcia-Morchon, S. L. Keoh, S. Kumar, P. Moreno-Sanchez, F. Vidal-Meca and J.H. Ziegeldorf. Securing the IP-based Internet of Things with HIP and DTLS. In *Sixth ACM Conference on Security and Privacy in Wireless and Mobile Networks*, WiSec '13, pages 119–124, New York, NY, USA, 2013. ACM.

[15] R. Hummen, H. Wirtz, J.H. Ziegeldorf, J. Hiller and K. Wehrle. Tailoring end-to-end IP security protocols to the Internet of Things. In *21st IEEE International Conference on Network Protocols (ICNP)*, pages 1–10, October 2013.

[16] S. Cheshire and M. Krochmal. *RFC 6763, DNS-Based Service Discovery*. Internet Engineering Task Force, February 2013.

[17] S. Keoh, S. Kumar, E. Dijk and A. Rahman. *DTLS-based Multicast Security for Low-Power and Lossy Networks (LLNs), draft-keoh-dice-multicast-security-08 (Work in progress)*. Internet Engineering Task Force, July 2014.

[18] T. Dierks and E. Rescorla. *RFC 5246, The Transport Layer Security (TLS) Protocol Version 1.2*. Internet Engineering Task Force, August 2008.

[19] T. Hardjono and B. Weis. *RFC 3740, The Multicast Group Security Architecture*. Internet Engineering Task Force, March 2004.

[20] T. Kothmayr, C. Schmitt, W. Hu, M. Brünig and G. Carle. DTLS based security and two-way authentication for the Internet of Things. *Ad Hoc Networks*, 11(8):2710–2723, 2013.

[21] Z. Shelby, C. Bormann and S. Krco. *CoRE Resource Directory draft-ietf-core-resource-directory-01 (Work in progress)*. Internet Engineering Task Force, December 2013.

[22] Z. Shelby, K. Hartke and C. Bormann. *RFC 7252, Constrained Application Protocol (CoAP)*. Internet Engineering Task Force, June 2014.