A

Axiom - DTLS-based secure IoT group communication

MARCO TILOCA, SICS Swedish ICT AB
KIRILL NIKITIN, SICS Swedish ICT AB
SHAHID RAZA, SICS Swedish ICT AB

This article presents Axiom, a DTLS-based approach to efficiently secure multicast group communication among IoT constrained devices. Axiom provides an adaptation of the DTLS record layer, relies on key material commonly shared among the group members, and does not require to perform any DTLS handshake. We made a proof of concept implementation of Axiom based on the tinyDTLS library for the Contiki OS, and used it to experimentally evaluate performance of our approach on real IoT hardware. Results show that Axiom is affordable on resource constrained platforms, and performs significantly better than related alternative approaches.

CCS Concepts:•**Security and privacy** → **Security protocols;**•**Computer systems organization** → *Embedded and cyber-physical systems;*

Additional Key Words and Phrases: Security, DTLS, multicast, group communication, Internet of Things

ACM Reference Format:
Marco Tiloca, Kirill Nikitin, and Shahid Raza, 2015. Axiom - DTLS-based secure IoT group communication. Special issue on Embedded Computing for Internet-of-Things (IoT), *ACM Trans. Embedd. Comput. Syst.* V, N, Article A (January YYYY), 25 pages.
DOI: 0000001.0000001

1. INTRODUCTION

We have been rapidly moving towards a pervasive networked society where all devices that can benefit from a connection will be connected with one another. This technology trend is commonly referred to as *Internet of Things* (*IoT*) [L. Atzori, A. Iera and G. Morabito 2010][G. Kortuem, F. Kawsar, D. Fitton and V. Sundramoorthy 2010], and aims at connecting the physical and cyber world by means of tiny resource constrained devices, embedded in everyday physical objects. To this end, different protocols have been standardized to enable interaction in the IoT. For instance, *6LoWPAN* [J. Hui and P. Thubert 2011] enables IP capabilities, *RPL* [T. Winter, P. Thubert, A. Brandt, J. Hui, R. Kelsey, P. Levis, K. Pister, R. Struik, JP. Vasseur and R. Alexander 2012] enables routing capabilities, and *CoAP* [Z. Shelby, K. Hartke and C. Bormann 2014] enables web capabilities.

Several IoT application scenarios such as smart lighting applications, collective building control, and emergency broadcast services can benefit from the adoption of a group communication model, regardless of the specific application level protocol. According to this communication model, a device becomes a member of a group by explicitly joining it as *sender* or *listener* node, or both. A sender node transmits multicast messages addressed to the group, and intended to all the listener nodes. This considerably reduces the number of transmitted messages, as a single network packet is delivered to multiple recipients at the same time. Also, it reduces the communication overhead and energy consumption on network devices, so improving network responsiveness and lifetime.

Author's addresses: M. Tiloca and S. Raza, SICS Swedish ICT AB, Isafjordsgatan 22, Kista (Sweden), email: {marco, shahid}@sics.se; K. Nikitin, School of Computer and Communication Sciences, EPFL, EDOC-IC INN 134 (Bâtiment INN) Station 14, Lausanne (Switzerland), email: kirill.nikitin@epfl.ch
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.

DOI: 0000001.0000001

Since CoAP has emerged as the de-facto IoT standard among application level protocols, and several IoT use cases benefit from group communication, the IETF has specifically standardized the usage of CoAP in a group communication context [A. Rahman and E. Dijk 2014]. In principle, a multicast request is transmitted as a CoAP message addressed to an IP multicast address, rather than to a CoAP endpoint. Therefore, a recipient listener node should be able to recognize that a CoAP request message has arrived via multicast. Also, CoAP group communication runs in the Any Source Multicast (ASM) mode of IP multicast operation [P. Savola 2008]. This means that there is no restriction on the source node that originates a CoAP request message and sends it to the IP multicast group. Furthermore, [A. Rahman and E. Dijk 2014] describes how to use CoAP on top of IP multicast, according to the pre-existing CoAP functionalities and newly defined features. In particular, it defines new rules for reuse of CoAP token values, request suppression, and proxy operations, as well as a new management interface for group membership configuration.

At the same time, several application scenarios require secure communication among IoT devices. To secure unicast communication between two peers, CoAP mandates the adoption of the *Datagram Transport Layer Security* (*DTLS*) protocol [E. Rescorla and N. Modadugu 2012], which has become the de-facto security protocol for the IoT. However, CoAP does not specify any mechanism to secure multicast communication. Since DTLS is the mandated choice for secure unicast communication, it makes particular sense to adapt it in order to secure also multicast communication. Although IPsec multicast [B. Weis, G. Gross and D. Ignjatic 2008] is indeed a possible alternative, it would require to adopt additional heavyweight security protocols, e.g. IPsec [S. Kent and K. Seo 2005][B. Briscoe 2010], and most likely even IKEv2 [C. Kaufman, P. Hoffman, Y. Nir, P. Eronen and T. Kivinen 2014]. Also, since unicast communication in the IoT is supposed to be protected through DTLS, a full adoption of solutions based on IPsec would be even less convenient and practical.

Following this rationale, the IETF has worked on how to adapt the DTLS protocol, in order to protect multicast messages sent to a group of IoT devices. That is, the approach proposed in [S. Keoh, S. Kumar, E. Dijk and A. Rahman 2014] protects multicast messages through the usual DTLS security services, and relies on the same group key material shared among all group members. However, this approach is prone to practical issues and can result in severe performance degradation, especially in large scale, dynamic, groups. In fact, the protection of unicast response messages is based on traditional unicast DTLS sessions, separately established between each pair of sender and listener nodes. This requires performing a significant number of DTLS handshakes, and may be particularly inconvenient in case unicast secure communication is required only for group response messages. Furthermore, it requires group members to store a considerable amount of supplementary security material, in order to maintain additional, possibly unnecessary, unicast DTLS sessions.

This paper presents Axiom, an alternative DTLS-based approach to secure multicast group communication. Axiom overcomes the limitations of the original IETF proposal and displays the following benefits. *First*, it relies on a single set of group key material to efficiently secure multicast messages transmitted by sender nodes. *Second*, unlike the original IETF proposal, each listener node relies on the common group key material also to efficiently and locally derive individual key material, used to secure unicast response messages. This is necessary only upon receiving the first multicast message from a given sender node, prevents a number of security attacks, and addresses the presence of multiple sender nodes. Similarly, each sender node relies on the common group key material also to efficiently and locally derive the same individual key material, only upon receiving the first unicast response message from a given listener node. *Third*, Axiom allows group members to avoid the execution of DTLS handshakes altogether. This limits the impact on performance, in terms of memory occupancy, computing and communication overhead, and energy consumption. Also, it preserves network responsiveness and service availability, especially in large scale dynamic groups where new listener nodes may join at any time. *Fourth*, Axiom keeps sender nodes agnostic of the listener nodes in the group, so considerably simplifying management operations.

In order to prove that our approach is feasible on constrained IoT devices and is preferable to the original IETF proposal, we made a proof of concept implementation of Axiom by extending the library *TinyDTLS* for the Contiki operating system [con 2015]. Then, we relied on our implementa-

tion to experimentally evaluate Axiom and compare it with two alternative secure approaches, i.e. a full unicast scheme and the original IETF proposal. Results show that Axiom is affordable on resource constrained platforms, and outperforms alternative approaches based on DTLS, in terms of memory occupancy, communication overhead and energy consumption. Our extended DTLS library is released as open source and under open license, and is available online at [tin 2015b]. To the best of our knowledge, it is currently the only publicly available implementation of DTLS for Contiki that also supports secure group communication. In order to encourage the adoption of our approach, we have also submitted a draft description to the IETF for possible standardization [M. Tiloca, S. Raza, K. Nikitin, S. Kumar 2015] Our draft focuses on the theoretical contribution and practical considerations, while it does not refer to any particular implementation or experimental evaluation.

The paper is organized as follows. Section 2 highlights relevant IoT use cases that benefit from group communication. Section 3 overviews DTLS, while Section 4 discusses multicast communication in the IoT and introduces the approach originally proposed by the IETF. Section 5 presents our new approach Axiom. Section 6 discusses the design choices behind our proposal and its impact on performance, while Section 7 provides a security analysis of Axiom. In Section 8, we present our experimental evaluation. Finally, in Section 9 we draw our conclusive remarks.

2. GROUP COMMUNICATION IN IOT USE CASES

Group communication is an important mechanism for several IoT application scenarios, especially for low-power and lossy networks where constrained devices often naturally operate in groups. Sending the same message to multiple recipients with a single network invocation considerably improves performance and reduces energy consumption on network devices. To this end, the specification [A. Rahman and E. Dijk 2014] defines how the CoAP protocol is used on top of IP multicast, to enable group communication in various IoT use cases. However, CoAP does not protect group communication, and instead suggests the adoption of security services at the application layer.

Securing communication in a multicast group is as important as in systems based on the unicast communication model. That is, sensitive or even mission-critical applications, e.g. health monitoring systems and alarm monitoring systems, require that (multicast) network messages are properly secured, and that only legitimate group members can take part in the group communication. In the following, we overview some IoT use cases that benefit from group communication and typically require the presence of security to protect message exchange within the group.

Smart lighting. Many buildings will be soon IoT-enabled and consisting of 6LoWPAN-connected lighting devices and switches. Devices can be organized into multicast groups according to their physical location, while switches can be used to send on/off or dimming multicast commands. In this environment, group communication enables synchronous operations , such as changing the light preset of several luminaries at the same perceived time. Devices may reply back to the switches, and report about the execution of the requested operation and their operational status. In Internet-connected lighting scenarios, it is necessary that only legitimate entities can use the system.

Collective building control. When enabling Building Automation and Control Systems (BACSs), group communication is desirable to control multiple air-conditioning, ventilation and heating units to default presets. Controlled units can be organized into multicast groups according to their physical position in the building, e.g. a multicast group for each room, and can acknowledge the execution of a requested operation and their operational status to the BACs issuing control commands. Such interactions with the critical assets in a building must be performed only by legitimate users.

Automatic software and firmware updates. In most real-world IoT deployments, updates often consist in large amounts of data, and can overload a constrained network that usually deals with small amounts of data, on a sporadic base. Rather than unicasting updates to each individual device, multicasting updates reduces the network load and decreases the overall latency for transmitting data to all devices. Even if the whole update process itself is secured, securing the individual messages is important, in case updates consist of relatively large amounts of data. In fact, checking individual received data piecemeal for tampering avoids devices from storing large amounts of par-

tially corrupted data and from detecting tampering only after all data have been received. Devices are expected to reply back, in order to provide a confirmation and their operational status.

Configuration and parameter update. Group communication enables efficient updates of multiple devices, with network load management and access control as relevant examples. Devices receiving parameter and configuration updates are expected to reply back and confirm their operational status.

Commissioning 6LoWPAN networks. A commissioning device probes a selected subset of network devices, to discover their presence and enquire their configuration, capabilities and operating conditions. Devices with similar features and capabilities, or in the same physical location, can be members of a single multicast group, and reply back to provide the requested information.

Emergency broadcast. In emergency situations as natural disasters, a crisis announcement is generated and broadcast by a trusted emergency notifier to multiple devices. A recipient device may reply back to provide its feedback or local information about the ongoing emergency.

3. THE DTLS PROTOCOL

The *Datagram Transport Layer Security* (*DTLS*) protocol [E. Rescorla and N. Modadugu 2012] designed by the IETF provides secure communication for unreliable datagram protocols, such as UDP. Furthermore, DTLS has been selected as the default, must-implement, security protocol for securing communication relying on the *CoAP* protocol [Z. Shelby, K. Hartke and C. Bormann 2014]. Since CoAP has become the de-facto application protocol for the IoT, DTLS is practically the de-facto security protocol in the majority of IoT application scenarios. The IETF has also defined a DTLS profile that offers communication security services for IoT applications and is reasonably implementable on many constrained devices [H. Tschofenig and T. Fossati 2016].

DTLS is based on the *TLS* protocol [T. Dierks and E. Rescorla 2008], and provides equivalent security guarantees, so allowing client and server applications to communicate while preventing eavesdropping, tampering, and message forgery. However, DTLS introduces some changes to deal with unreliable datagram transport protocols. First, stream ciphers cannot be adopted, and an explicit *sequence number* is included in every DTLS message. Thus, messages are independent from one another, and can be correctly processed despite possible out-of-sequence deliveries. Also, packet loss is explicitly addressed though local timeouts and message retransmission. Finally, invalid received messages can be silently discarded, and the associated DTLS session may not be terminated.

Communication among two DTLS peers relies on secure *sessions*. Messages are transmitted as a series of *records*, whose structure is shown in Figure 1. The *Type* field indicates the higher level protocol used to process the enclosed data, while the *Version* field states the employed version of the protocol. The *Length* field indicates the size of the application data, conveyed in the *Fragment* field. Finally, with respect to TLS, two additional fields are present, i.e. *Epoch* and *Sequence Number*. The former is incremented in case security protocols and material are changed. The latter is incremented for every new record transmitted by the same peer over the same DTLS session. The concatenation of the *Epoch* and *Sequence Number* fields is considered as a single 64 bit fresh value used to compute a Message Authentication Code for assuring integrity of DTLS records.

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

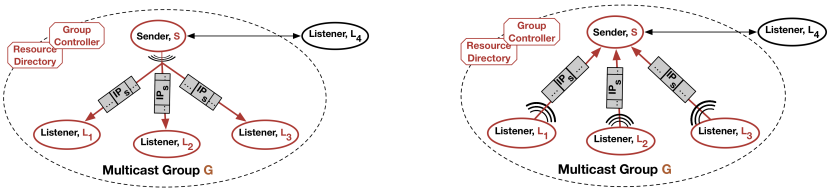
Fig. 1. Original DTLS record format.

A pair of DTLS *client* and *server* establishes a secure session through a specific message exchange, namely *handhshake*. The client typically starts a handshake, by sending a *ClientHello* message to the server. Before performing the handshake, DTLS assumes that involved peers have been previously provided with some security material. This consists in a set of preinstalled keys used during the handshake to agree on a *pre master secret*. This *pre master secret* is later used, together with random values generated by the client and server, to compute a *master secret* from which the final security material is derived. Two main approaches are considered to provide preinstalled keys.

The first approach relies on *asymmetric key* pairs. It considers *X.509* certificates [D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley and W. Polk 2008] or *raw public keys* [P. Wouters, H. Tschofenig, J. Gilmore, S. Weiler and T. Kivinen 2014], where key pairs come with no certificate and may be installed before deployment. A device relies on out-of-band means to validate raw public keys, and retains a list of peers it can communicate with. The second approach considers symmetric *pre-shared keys* [P. Eronen P. and H. Tschofenig 2005], provided to clients and each server they intend to communicate with. During the handshake, clients indicate which symmetric key must be considered to compute the *pre master secret*. This avoids the exchange of public certificates and costly public key operations. Also, it simplifies management operations, especially in environments where connections are manually configured, and certificates are not a preferable or feasible option.

4. TOWARDS SECURE GROUP COMMUNICATION

Multicast communication occurs in a pre-established IPv6 multicast group. In order to participate, a device must *join* the group as a new member, by registering with a routing device and signaling the intent to receive messages sent to the multicast group. Then, group members can have two possible roles, namely *sender* and *listener*. A sender node transmits multicast messages to the group, while a listener node receives them when listening to the multicast IPv6 address of the group. A listener node may reply, by sending a unicast response message to that sender node. A group member may have one or both roles, depending on the running application and its interactions with the communication stack. Also, it does not have to perform any access procedures or authentication to send or receive multicast messages [B. Weis, G. Gross and D. Ignjatic 2008]. Figure 2 shows an example of 1-to-N group communication. In order to send a multicast message to all the listener nodes, a sender node refers to a destination address that has been previously assigned to the group and allocated for IPv6 multicast. In principle, any device can listen to any IPv6 multicast address. Hence, applications do not know, and do not get notified, when new listener nodes join the group. Group communication for the CoAP protocol is also based on IPv6 multicast [A. Rahman and E. Dijk 2014].



(a) IP multicast message (b) IP unicast responses

Fig. 2. Group communication scenario. The multicast group G includes one sender node S, and three listener nodes L_1, L_2 and L_3. First, node S sends a single multicast message to the whole group (Figure 2(a)). Then, each listener node separately replies back to the sender node with a unicast response message (Figure 2(b)).

The IETF proposed an approach to enable secure group communication in the IoT, by adapting DTLS to protect multicast messages sent to the group [S. Keoh, S. Kumar, E. Dijk and A. Rahman 2014]. However, many application scenarios expect listener nodes to reply with individual response messages, that may require to be secured as well. The proposed approach considers this issue, and prescribes that unicast responses to multicast messages must be secured, by means of traditional DTLS sessions. That is, every sender node is required to perform a DTLS handshake with all the listener nodes in the group, i.e., in the worst case, with all the other group members. This is inconvenient especially if unicast secure communication is required only to protect responses from listener nodes. Also, it is likely to be inefficient and even not feasible for constrained IoT devices, due to the complexity of the DTLS handshake, and the resulting computing and communication overhead.

In fact, as a first option, all sender nodes establish a DTLS session with each listener node, before sending their first multicast message. This requires all sender nodes to be aware of the listener

nodes currently in the group. Also, all sender nodes establish further DTLS sessions in case new listener nodes join the group. Hence, sender nodes must be provided with otherwise unnecessary information, and kept up to date about changes in the group membership. However, the IETF proposal explicitly states that applications on group members do not know, and are not supposed to get notified, when new listener nodes join the group. Alternatively, every listener node can establish a DTLS session with a sender node, before replying with its own first response message. This may result in sender nodes flooded by a consistent number of DTLS handshakes to be performed. Both the alternatives discussed above introduce a non negligible delay before the group can become fully operative, due to the considerable number of required DTLS handshakes. The impact on devices' availability and performance would be even more severe in dynamic scenarios where devices may join the group at any time, or are allowed to be members of more than one group at the same time.

As an alternative to the IETF proposal, a preliminary version of the theoretical approach described in this paper was presented in [M. Tiloca 2014], where a smart use of the group key material makes it possible to secure also response messages sent by listener nodes, without performing any DTLS handshake. However, it protects response messages through the very same group key material shared by all listener nodes, hence resulting in a number of security vulnerabilities that make the approach unsecure in the presence of multiple sender nodes. Instead, the enhanced solution Axiom presented in this paper considerably improves the preliminary approach mentioned above, and successfully overcomes its shortcomings and limitations. In particular, it introduces the derivation of individual key material for each listener node, so preventing a number of security attacks in the presence of multiple sender nodes. Also, this paper provides an overview of relevant IoT use cases, a discussion about design choices and impact on performance, a security analysis, and an experimental evaluation based on our proof of concept implementation for the Contiki operating system.

5. TWO-WAY DTLS-BASED SECURE GROUP COMMUNICATION

This section describes our DTLS-based approach Axiom for two-way secure group communication. Hereafter, we assume the presence of the application protocol CoAP, as it has become the de-facto standard in the IoT. However, Axiom consists in adapting the DTLS record layer only. Therefore, just like DTLS, Axiom does not require, nor implies, the adoption of CoAP, and thus is immediately reusable with other application protocols. The rest of the paper considers the following terminology.

- *Key Material*: set of data which is necessary to establish and maintain a cryptographic security association. This includes, for instance, keys, key pairs, and IVs [R. Shirey 2007].
- *Security association (SA)*: set of policies and key material that provide security services to matching network traffic [T. Hardjono and B. Weis 2004]. It includes different attributes, such as: i) selectors, e.g. source and destination addresses; ii) properties, e.g. identities of involved entities; and iii) key material and cryptographic policies, e.g. algorithms, modes, key lifetimes, and key lengths.
- *Group controller (GC)*: entity responsible for creating a multicast group, establishing security associations among group members, and managing the joining of new group members. The GC is not required to be an actual member of the multicast group and to take part in the group communication.
- *Group security association (GSA)*: a security association defining how members of a multicast group communicate securely [T. Hardjono and B. Weis 2004]. Group members are provided with the GSA upon joining the group, by the GC or through alternative out-of-band means.
- *Group connection state*: a data structure including a pair {*Epoch*, *Truncated Sequence Number*} and possible derived key material. A group member can store multiple group connection states.
- *Group request*: multicast message sent by a sender node to all the listener nodes in the group.
- *Group response*: unicast message sent by a listener node as a response to a group request.

5.1. Overview of Axiom

In the following, we provide a high-level overview of our approach. Figure 3 shows two full message exchanges in the multicast group and the operations performed on the single group members, with

reference to one sender node S and two listener nodes L1 and L2. A more detailed description of the security data structures and message processing is provided later in Sections 5.3-5.6.

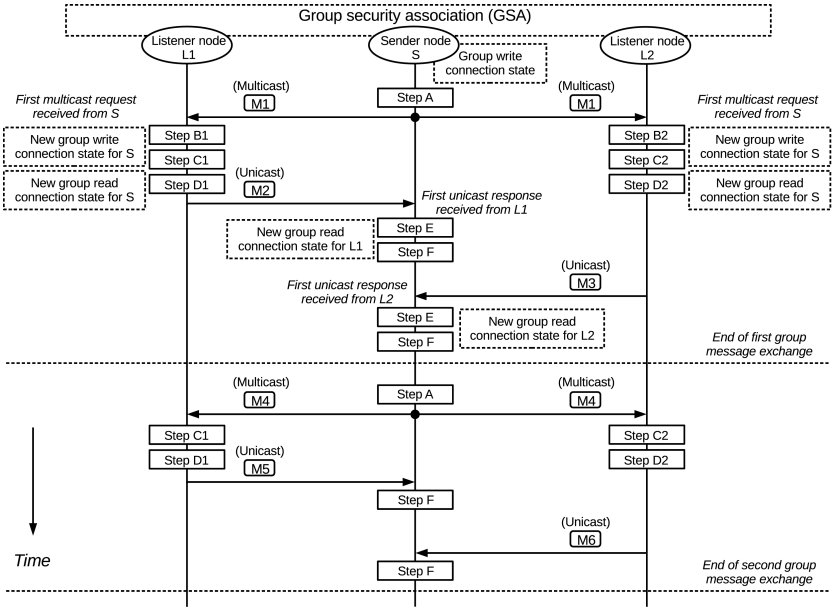


Fig. 3. Example of group message exchanges based on Axiom between one sender node S and two listener nodes L1 and L2. Upon receiving the first multicast group request from S, both the listener nodes perform additional operations (steps B1 and B2, respectively). Upon receiving the first unicast group response from L1 and L2, the sender node S performs additional operations (step E for each listener node). These operations are not performed for the subsequent message exchanges.

As shown in Figure 3, the nodes S, L1 and L2 share the same GSA together with the common group key material. Also, the sender node S maintains a group write connection state. Then, S starts a first message exchange, i.e. it prepares a multicast group request M1 and performs *Step A*, that is: i) it secures message M1 by means of the group key material in the GSA; and ii) it updates the sequence number in the write group connection state. Then, S transmits the multicast message M1.

As M1 is the first group request received from S, the listener node L1 performs *Step B1*, that is L1: i) creates a read group connection state and a write group connection state, both associated to S; and ii) derives individual key material associated to S, and stores it in the same write group connection state. After that, L1 performs *Step C1*, that is L1: i) unsecures M1 by means of the group key material in the GSA; and ii) updates the sequence number in the read group connection state associated to S. Then, L1 prepares a unicast group response M2, and performs *Step D1*, that is L1: i) secures M2 by means of the individual key material in the write group connection state associated to S; and ii) updates the sequence number in the same write group connection state. Finally, L1 transmits message M2 to S. The same considerations hold for the listener node L2, that performs the equivalent *Step B2*, *Step C2* and *Step D2*, and transmits its unicast group response M3 to S.

As M2 is the first group response received from L1, the sender node S performs *Step E*, that is S: i) creates a read group connection state associated to L1; and ii) derives the *same* individual key material used by L1 to secure message M2, and stores it in the read group connection state. After that, S performs *Step F*, that is S: i) unsecures M2 by means of the individual key material in the read group connections state associated to L1; and ii) updates the sequence number in the same read group connection state. The same considerations hold for the reception of message M3, as to: i) the creation of a read group connection state associated to L2 and the derivation of the individual key material at *Step E*; and ii) the processing of message M3 at *Step F*.

Then, the sender node S starts a second message exchange, by performing *Step A* and transmitting a new multicast group request M4. As M4 is *not* the first group request received from S, the listener node L1 (L2) skips *Step B1* (*Step B2*) and immediately processes message M4 through *Step C1* (*Step C2*). Then, L1 (L2) prepares the unicast group response M5 (M6), processes it through *Step D1* (*Step D2*), and transmits it to S. As M5 (M6) is *not* the first group response received from L1 (L2), the sender node S skips *Step E* and immediately processes message M5 (M6) through *Step F*.

Requests and responses are bound to the specific multicast group by a *GroupID* value, included in the Axiom adapted DTLS header and representing an alias for the multicast IP address associated to the group. Besides, the group members generate the individual key material by considering: i) the common group key material in the GSA; ii) the IP address of the specific listener node involved in the message exchange; and iii) the *SenderID* value, conveyed in the multicast group request, i.e. a unique identifier associated to the sender node, and included in the Axiom adapted DTLS header.

5.2. Security requirements

Our approach Axiom is expected to fulfill the following security requirements.

• *Multicast communication topology*. Both the 1-to-N (one sender and multiple listeners) and the M-to-N (multiple senders and multiple listeners) communication topologies must be supported.
• *Multicast group size*. Different, possibly large, group sizes must be adequately supported. Group size is the combination of the number of sender nodes and listener nodes in a multicast group, with possible overlap (i.e. a sender node may also be configured as listener at the same time). In typical IoT use cases, the number of sender nodes is expected to be much smaller than the number of listener nodes, and the total number of group members is expected to be in the range of 2 to 100 devices. Groups larger than that should be divided into smaller independent multicast groups.
• *Data replay protection*. It must not be possible to replay messages sent within the group.
• *Group-level data confidentiality*. Messages sent within the multicast group must be encrypted. In fact, some control commands and/or associated responses could pose unforeseen security and privacy risks to the system users, when sent as plaintext. Axiom considers group-level data confidentiality since messages are encrypted at a group level, i.e. in such a way that any member of the multicast group can decrypt them, but an external adversary or other external entities cannot.
• *Data authentication*. Messages sent within the multicast group must be authenticated to ensure group-level data authentication, i.e. that a message is originated by a member of the group. Axiom provides group-level data authentication both for group requests and group responses. All group members are trusted to not tamper with the messages sent within the group.
• *Data integrity*. Messages sent within the multicast group must be integrity protected, to ensure that they have not been tampered with by an external adversary or other external entities. Data integrity is provided through the same means used to provide data authentication.

Furthermore, Axiom assumes the following security requirements to be already fulfilled.

• *Establishment of a GSA*. The Group Controller must establish the GSA associated to the multicast group, and securely provide it to new group members upon their joining.
• *Multicast data security ciphersuite*. All group members must agree on a ciphersuite specified in the GSA. As authenticity is more important than confidentiality in most IoT scenarios, at least ciphersuites with MAC only (NULL encryption) and AEAD [D. McGrew 2008] ciphersuites must be supported. Other ciphersuites for data record security used in DTLS should also be supported.
• *Backward and forward security*. A device that joins the group must not have access to previous GSAs or group messages exchanged before its joining. At the same time, devices that leave the group must not have access to future GSAs or group messages exchanged after their leaving. Also, they must not be able to send encrypted and/or integrity protected messages to the group after their leaving. The procedure to update the GSA and renew the group key material upon a group member's joining or leaving have to be defined as part of the adopted group key management scheme.

5.3. Security data structures

• *SecurityParameters*. Upon their joining, all members of the group receive from the GC the structure *SecurityParameters*, which reflects the definition in Section 6.1 of [T. Dierks and E. Rescorla 2008]. In particular, *entity* is set to *ConnectionEnd.server* for sender nodes and *ConnectionEnd.client* for listener nodes. All other parameters have the same value for all group members.

• *Group Security Association*. Every device owns a *Group Security Association* (*GSA*) for each multicast group it belongs to. The GSA is received upon joining the group, and includes the following elements: *GroupID*; *SenderID*; *CipherSuite*; *client write IV*; *client write encryption key*; *client write MAC key*; *server write IV*; *server write encryption key*; *server write MAC key*.

The *GroupID* has the same value for all the group members, and represents an alias for the multicast IP address associated to the group. We assume that *GroupIDs* are in the range between 0 and 255. The *SenderID* is relevant only for group members configured (also) as senders, and is determined by the GC upon a sender node's joining. A *SenderID* value must be unique within the group at any time. We assume that *SenderIDs* are in the range between 0 and 255. The GC provides a list of active sender nodes and their respective *SenderID* to all the listener nodes in the group.

The GC chooses a CipherSuite which is specified in the GSA and must be supported by all the group members. Also, all the group members derive the same group key material *client write IV*, *server write IV*, *client write encryption key*, *server write encryption key*, *client write MAC key* and *server write MAC key*. To this end, they consider the security parameters *master secret*$[48]$, *client random*$[32]$ and *server random*$[32]$ included in the *SecurityParameters* structure, and rely on the $PRF(\cdot)$ function defined in Section 6.3 of [T. Dierks and E. Rescorla 2008].

• *Group connection states*. Each group member instantiates different *group connection states*, depending on its role(s) in the group. The *Truncated Sequence Number* value is initialized to 0, while the *Epoch* value is fixed and provided by the GC to the group members upon their joining. A group member configured as sender node maintains: i) one *write group connection state*, instantiated during the node's initialization and referred when sending multicast group requests; and ii) one *read group connection state* for each listener node in the group, instantiated upon receiving a unicast group response from the associated listener node for the first time. Instead, a group member configured as listener node maintains: i) one *write group connection state* for each sender node in the group, instantiated upon sending a unicast group response to the associated sender node for the first time; and ii) one *read group connection state* for each sender node in the group, instantiated upon receiving a multicast group request from the associated sender node for the first time.

5.4. Secure multicast group requests

This section describes how multicast group requests are sent and received, considering the adapted DTLS record header depicted in Figure 4. The original 6-octet *Sequence number* field is split into a 1-octet *SenderID*, and a 5-octet *Truncated sequence number* increased by 1 each time the same sender node transmits a new DTLS record as part of a multicast group request.

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

Fig. 4. Adapted DTLS record header for multicast group requests. The original Sequence Number field has been split into two subfields, namely SenderID and Truncated Sequence Number.

A sender node transmits a DTLS record as part of a secure multicast group request as follows.

1) The sender node determines the right GSA and *write group connection state*, based on the multicast IP destination address and destination port number.
2) The sender node includes its own *SenderID* in the *SenderID* field of the adapted DTLS

header. Then, it considers the *Epoch* and *Truncated sequence number* values from the *write group connection state*, and uses them to fill the *Epoch* and *Truncated sequence number* fields of the adapted DTLS header. The first record sent to the group has *Truncated sequence number* 0.
3) The sender node processes the DTLS record according to the CipherSuite specified in the GSA. The *server write MAC key* and *server write IV* in the GSA are used for authentication, while the *server write encryption key* and *server write IV* in the GSA are used for the encryption operation.
4) The sender node passes the DTLS record to the lower layers for transmission on the multicast IP destination address and port number associated to the multicast group. Then, it updates the *Truncated sequence number* in its own *write group connection state*, incrementing its value by 1.

A listener node receives a DTLS record as part of a secure multicast group request as follows.

1) The listener node determines the right GSA, based on the multicast IP destination address and destination port number.
2) The listener node considers the multicast IP destination address and destination port number as well as the *SenderID* specified in the adapted DTLS header, in order to determine the right *read group connection state*. This is different from the standard DTLS logic, where the current *client read connection state* is bound to the IP source address and port number. If this is the first group request ever received from that sender node in the group, the listener node instantiates a *read group connection state* associated to that sender node.
3) The listener node checks the freshness of the received DTLS record, comparing the values conveyed in the *Epoch* and *Truncated sequence number* fields with the respective values stored in the selected *read group connection state*. The retrieved *Epoch* and *Truncated sequence number* should coincide with the respective stored values. Alternatively, a sliding window mechanism may be adopted to accept genuine out-of-order DTLS records.
4) The listener node processes the DTLS record according to the CipherSuite in the GSA. The *server write encryption key* and *server write IV* in the GSA are used to decrypt the DTLS record, while the *server write MAC key* and *server write IV* in the GSA are used for the authenticity check.
5) Once the freshness and authenticity of the DTLS record have been verified, the listener node passes the DTLS record to the higher level protocols. Then, it updates the value of the *Truncated sequence number* in the selected *read group connection state*, incrementing its value by 1.

5.5. Derivaton of individual key material

Before sending its first group response to a given sender node, a listener node computes individual key material and stores it in the *write group connection state* associated to that sender node. From then on, the listener node considers that individual key material to process all its group responses addressed to that sender node. Similarly, upon receiving the first group response from that listener node, the sender node computes the same individual key material and stores it in the *read group connection state* associated to that listener node. From then on, the sender node considers that individual key material to process all the group responses received from that listener node.

The individual key material is derived according to the same data expansion scheme used by DTLS and described in Section 6.3 of [T. Dierks and E. Rescorla 2008]. The invoked procedure considers as input the *client write key* and *server write key* originally stored in the GSA, the IP address IP_L of the listener node, and the *SenderID* associated to the sender node. With reference to the structure *SecurityParameters* introduced in Section 5.3, the following derivation is performed.

key_block = PRF(client write key + server write key, "key derivation", IP_L *+ SenderID)*

The procedure stops when an amount of bytes equal to *SecurityParameters.mac_key_length* plus *SecurityParameters.enc_key_length* has been generated. Then, the individual key material is extracted as follows. The first *SecurityParameters.mac_key_length* bytes of the *key_block* output are considered as the *client write MAC key* to be used for group responses sent by that listener node to

that sender node. The listener node stores it as the individual *client write MAC key* in the *write group connection state* associated to that sender node. Instead, the sender node stores it as the individual *client write MAC key* in the *read group connection state* associated to that listener node.

The following *SecurityParameters.enc_key_length* of the *key_block* output are considered as the *client write encryption key* to be used for group responses sent by that listener node to that sender node. The listener node stores it as the individual *client write encryption key* in the *write group connection state* associated to that sender node. Instead, the sender node stores it as the individual *client write encryption key* in the *read group connection state* associated to that listener node.

5.6. Secure unicast group responses

This section describes how unicast group responses are sent and received, considering the adapted DTLS header depicted in Figure 5. The original 6-octet *Sequence number* field is split into a 1-octet *GroupID*, and a 5-octet *Truncated sequence number* increased by 1 each time the same listener node transmits a new DTLS record as part of a group response addressed to the same sender node.

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

Fig. 5. Adapted DTLS record header for unicast group responses. The original Sequence Number field has been split into two subfields, namely GroupID and Truncated Sequence Number.

A listener node transmits a DTLS record as part of a secure unicast group response as follows. Note that, if it possibly has an open traditional DTLS session with the sender node, the listener node must instead refer to that DTLS session to reply to the sender node. In such a case, the listener node is not required to maintain a *write group connection state* associated to that sender node.

1) The listener node determines the right GSA, based on the multicast IP destination address and destination port number of the group request it has received and wants to reply to.
2) If this is its first group response ever sent to that sender node, the listener node instantiates a *write group connection state* associated to that sender node. Then, the listener node derives the necessary individual key material and stores it in the *write group connection state* (see Section 5.5). Otherwise, the listener node considers the IP source address and source port number from the group request, and determines the *write group connection state* associated to the recipient sender node.
3) The listener node retrieves the *GroupID* from the GSA, and includes it in the *GroupID* field of the adapted DTLS header. Then, it considers the *Epoch* and *Truncated sequence number* values from the selected *write group connection state*, and includes them in the respective fields of the adapted DTLS header. The first record sent to the sender node has *Truncated sequence number* 0.
4) The listener node processes the DTLS record according to the CipherSuite in the GSA. In particular, the individual *client write MAC key* stored in the selected *write group connection state* and the *client write IV* stored in the GSA are used for authentication, while the individual *client write encryption key* stored in the selected *write group connection state* and the *client write IV* stored in the GSA are used for the encryption operation.
5) The listener node passes the DTLS record to the lower layers for transmission on the unicast IP address and port number of the recipient sender node. The listener node updates the *Truncated sequence number* in the selected *write group connection state*, incrementing its value by 1.

A group member discards any received group response, in case it is not a sender node. Instead, a sender node processes a DTLS record as part of a secure unicast group response as follows.

1) The sender node checks if a traditional DTLS session is open with the listener node. In case of positive match, the sender node must process the received message according to the traditional DTLS protocol [E. Rescorla and N. Modadugu 2012], and is not required to maintain a *read group*

connection state associated to that listener node. Conversely, in case of negative match, the sender node does not yet consider the received message to be invalid, and moves to step 2.
2) The sender node parses the DTLS header according to the format in Figure 5, and determines the right GSA based on the retrieved *GroupID*. If no GSA associated to that *GroupID* is found, the sender node discards the received message. Otherwise, the sender node moves to step 3.
3) If this is the first group response ever received from that listener node, the sender node instantiates a *read group connection state* associated to that listener node. Then, the sender node derives the necessary individual key material and stores it in that *read group connection state* (see Section 5.5). Otherwise, the sender node considers the IP source address of the listener node from the group response, and determines the right *read group connection state*
4) The sender node checks the freshness of the received DTLS record, comparing the values of the *Epoch* and *Truncated sequence number* fields with the respective values stored in the selected *read group connection state*. The retrieved *Epoch* and *Truncated sequence number value* should coincide with the respective stored values. Alternatively, a sliding window mechanism may be adopted to accept genuine out-of-order DTLS records.
5) The sender node processes the DTLS record according to the CipherSuite in the GSA. The sender node decrypts the DTLS record using the individual *client write encryption key* in the selected *read group connection state* and the *client write IV* in the GSA. Also, the sender node checks the authenticity of the DTLS record using the individual *client write MAC key* in the selected *read group connection state* and the *client write IV* in the GSA.
6) Once the freshness and authenticity of the DTLS record have been verified, the sender node passes the DTLS record to the higher level protocols. Then, it updates the *Truncated sequence number* in the selected *read group connection state*, incrementing its value by 1.

6. DISCUSSION AND PERFORMANCE ANALYSIS

This section discusses the design choices considered when defining our approach, and the impact on performance in the presence of Axiom. We refer the reader to [E. Rescorla and N. Modadugu 2012][T. Dierks and E. Rescorla 2008] for more considerations strictly related to the DTLS protocol.

6.1. Same GroupID in different multicast groups

In general, every multicast group is managed by a different GC, which is responsible to determine the *GroupID* used within the group. Then, it is possible that a device is configured as sender in multiple groups at the same time, and such groups are identified by the same, ambiguous, *GroupID*. In such a case, upon receiving a unicast group response as a reply to a multicast group request, such sender nodes would not be able to determine to which group communication such group response refers to, i.e. which GSA and *read group connection state* has to be considered to process the received group response. In order to overcome this issue, sender nodes can identify the right GSA and *read group connection state* to be considered by means of the extended tuple $<$*GroupID*, *IP source address*, *destination port number*$>$ retrieved from the DTLS, UDP and IP headers of the group response. This relies on the practical assumption that, if a sender node belongs to different groups, it actually takes part in each group's communication through a different application or application instance, each one of which refers to a different port number.

6.2. Late joining nodes

Upon joining the group, a listener node is not aware of the current *Epoch* and *Truncated sequence number* used by the different sender nodes. This means that, when such listener node receives a group request message from a sender node for the first time, it is not able to verify if the message is fresh and has not been replayed. In order to address this issue, we can rely on techniques similar to *AERO* [D. McGrew and J. Foley 2014]. That is, upon receiving the first group request message from a particular sender node, late joining listener nodes initialize their last seen *Epoch* and *Truncated sequence number* in their *read group connection state* associated to that sender node. However, they drop such a message without delivering it to the application layer. This provides a reference

point to identify if future messages are fresher than the last one seen. Alternatively, the GC can be an additional member of the group, configured as listener. Then, it can maintain the *Epoch* and *Truncated sequence number* of each sender node. When late joiners send a request to the GC to join the group, the GC can provide them with the list of *Epoch* and *Truncated sequence number* values to be stored in the *read group connection states* associated to the appropriate sender nodes.

6.3. Reduced sequence number space

Since the *sequence number* in the DTLS header is truncated from 6 to 5 octets, we observe a reduction of the sequence number space. This should be taken into account to ensure that the *Epoch* value is incremented before the *Truncated sequence number* wraps around. A sender node, or the GC as an alternative, can send a DTLS *ChangeCipherSpec* message to the whole group whenever the *Truncated sequence number* has been exhausted, in order that the *Epoch* value is increased by all group members. This should be done as part of the adopted group key management scheme.

6.4. Overhead and impact on performance

Since no *handshake* is performed, Axiom operates only with DTLS record layer messages. Also, Axiom only slightly modifies the structure of the DTLS record header, while keeping the same overall size in bytes. Finally, Axiom relies on the very same security services provided by the original DTLS record layer, in order to secure/unsecure messages. Therefore, Axiom displays the same message complexity as DTLS. Alternative approaches for secure group communication based on DTLS require that every sender node in the group performs a DTLS handshake with each replying listener node in the group. This has a considerable impact on memory occupancy and network performance, which linearly grows with the number of listener (sender) nodes in the group, from each sender (listener) node point of view. Instead, Axiom never requires performing a DTLS handshake, regardless the number of group members and the possible joining of new members during the network lifetime.

As to *memory occupancy*, each sender node stores one read group connection state for each listener node, hence the Axiom memory overhead grows linearly with the number of listener nodes in the group. On the other hand, each listener node stores: i) one read group connection state for each sender node in the group; and ii) one write group connection state for each sender node in the group. Hence, also on the listener nodes' side, the Axiom memory overhead grows linearly with the number of sender nodes in the group. This is exactly the same memory overhead *trend* observed for original DTLS, when a sender node establishes a separate DTLS session with each listener node. However, despite the same memory overhead trend, original DTLS implies the storage of multiple full DTLS sessions. This means that, given N_S sender nodes and N_L listener nodes in the group, each sender node has to store N_L DTLS sessions, and each listener node has to store N_S DTLS sessions. Instead, Axiom requires each sender node to store only one GSA as DTLS group session, one write group connection state, and N_L read group connection states. On the other hand, each listener node has to store one GSA as DTLS group session, N_S read group connection states and N_S write group connection states. Since group connection states include only an epoch and sequence number value, as well as a reduced set of key material, they require considerably less memory than a full DTLS session. Thus, Axiom displays a reduced memory overhead with respect to alternative approaches.

In order to secure/unsecure group messages, Axiom displays the same *computing overhead* as DTLS, since it relies on the very same crypto suites. The only additional computing overhead introduced by Axiom is due to the generation of the individual key material between a sender node and a listener node, based on the same key derivation procedure considered by DTLS. We recall that: i) a listener node generates the individual key material associated to a sender node, only once, i.e. upon receiving the first group request message from that sender node; and ii) a sender node generates the individual key material associated to a listener node, only once, i.e. upon receiving the first group response message from that listener node. It follows that Axiom relies on one-time efficient key material generation, whose number of occurrences on the listener (sender) nodes linearly grows with the number of sender (listener) nodes in the group. Since this requires significantly less operations

and processing than performing a full DTLS handshake, it follows that Axiom displays a limited computing overhead and is highly convenient with respect to alternative approaches.

As to the impact on *network performance*, Axiom introduces a communication overhead which is as large as the one introduced by the original DTLS record layer. In particular, no handshakes are performed between sender nodes and listener nodes, unlike other approaches previously proposed. Therefore, the communication overhead actually due to Axiom itself is not affected at all by the number of sender nodes and listener nodes in the group, even in large scale multicast groups. To fix ideas, let us consider a full message exchange between one sender node and N_L listener nodes. Then, we have that each listener node receives one multicast group request and replies with one unicast group response, regardless the value of N_L. That is, the impact on communication overhead due to the multicast request messages remains constant, regardless the number of sender nodes and listener nodes in the group. On the other hand, the sender node obviously receives N_L distinct response messages from the listener nodes. Hence, the impact on communication overhead due to the unicast response messages grows linearly with the number of listener nodes in the group. Of course, this is *not* due to Axiom itself, but it is instead related to the adoption of multicast IP communication in the first place. Thus, the communication overhead actually introduced by Axiom is limited to the adapted DTLS header, whose size is the same for all Axiom messages and the same as in the original DTLS header. Similar considerations hold for energy consumption, as it is directly related to the computing overhead and communication overhead.

The *leaving* of a group member does not result in Axiom affecting network performance. That is, if a sender node leaves, then listener nodes would simply stop receiving multicast group requests from that sender node. Conversely, if a listener node leaves, sender nodes would simply stop receiving unicast group responses from that listener node. To optimize memory consumption, one may introduce a timeout mechanism on the sender (listener) nodes, such that they deallocate any group communication state associated to a listener (sender) node, if they have not received any incoming messages from that node over a pre-defined time window. Similarly, the *joining* of a group member does not result in Axiom affecting network performance. That is, if we consider a full message exchange between a sender node and the listener nodes in the group, nothing changes if a new sender node joins. On the other hand, if a new listener node joins, a given sender node would obviously receive one more unicast group response. As already discussed above, this is of course unavoidable, and actually due to the adoption of multicast communication in the first place. Of course, the joining of additional group members has a (predictable) impact on memory consumption. That is, if a new sender node joins, then listener nodes create and store one read group connection state and one write group connection state, upon receiving the first multicast group request from that sender node. Instead, if a new listener node joins, a sender node would create and store one read group connection state, upon receiving the first unicast group response from that listener node.

7. SECURITY ANALYSIS

In this section, we provide a security analysis of Axiom, and argue how it effectively deals with some potential security issues. We refer the reader to [E. Rescorla and N. Modadugu 2012][T. Dierks and E. Rescorla 2008] for more considerations strictly related to the DTLS protocol.

7.1. Security of group messages

Axiom relies on the same DTLS record layer and its security services, while re-adapting it to handle multicast group request messages, and binding the latter with the related unicast response messages. Note that Axiom does not propose any alternative way to actually secure/unsecure group messages. That is, Axiom refers to the same ciphers, hash functions and cryptographic primitives considered by DTLS, whose building blocks have been widely proven to be reliable and secure. Hence, Axiom processes group messages in a way which is as secure as the original DTLS protocol.

All group members keep their key material secret, i.e. they do not share it out of the multicast group, and trust each other according to the typical group communication trust model. Key material has a size that makes attacks based on exhaustive analysis practically infeasible for an adversary

external to the multicast group. Also, as discussed more in detail in the rest of this section: i) including a unique senderID in the group requests avoids that *Truncated Sequence Numbers* from different sender nodes overlap, so assuring the correctness of nonces for message authentication; ii) the generation and usage of individual key material associated to a single listener node prevents possible attacks based on identity impersonation and message injection performed by an external adversary; iii) the adoption of group key management policies and protocols to renew the group key material upon nodes' joining or leaving makes it possible to preserve the security properties mentioned above, together with forward security and backward security of group communication.

7.2. Handling secure group requests

The DTLS record layer admits AEAD ciphers like AES-CCM [D. McGrew and D. Bailey 2012] and AES-GCM [J. Salowey, A. Choudhury and D. McGrew 2008]. According to the AES-CCM specification for TLS [D. McGrew and D. Bailey 2012], the CCMNonce is a combination of a *salt* and a *nonce explicit* value. While the salt is the *client write IV* or the *server write IV*, depending on the communication direction, the *nonce explicit* value must be distinct for each invocation of the CCM encrypt function for any fixed key [D. McGrew and D. Bailey 2012]. Since traditional DTLS sessions are established among two peers only and rely on unique pairwise key material, the original *nonce explicit* consists in a 64-bit sequence number, composed of the 16-bit *Epoch* value concatenated with the 48-bit *Sequence number* value included in the DTLS header.

However, all sender nodes in Axiom refer to the same group key material to secure multicast group requests. Hence, if the original DTLS record header in Figure 1 is considered, the possible alignment of *Sequence number* values considered by different sender nodes would also result in nonce reuse for AEAD cipher suites, so completely breaking their security. In order to prevent that a same CCMNonce is reused, either all sender nodes in the group keep their sequence numbers synchronized, or separate non-overlapping sequence number spaces must be created for each sender node. Synchronization between sender nodes is particularly difficult to achieve, especially among constrained IoT devices. Thus, Axiom considers the second approach and separates the sequence number spaces, by embedding a unique sender identifier, i.e. the *SenderID*, in the original sequence number, as suggested in [J. Salowey, A. Choudhury and D. McGrew 2008]. In particular, sender nodes transmit multicast group requests considering the adapted DTLS header in Figure 4, which includes the *SenderID* and *Truncated sequence number* fields. It follows that the GC is also required to assign a unique *SenderID* to each device in the group configured as sender. Also, the GC provides a list of active sender nodes and their *SenderID* to all listener nodes in the group.

7.3. Handling secure group responses

Just like DTLS, our approach Axiom randomizes the encryption and authentication operations by means of a nonce value. In particular, a listener node securing a group response composes the nonce as the sequence of: i) the *client write IV* from the GSA; and ii) the concatenation of *Epoch*, *GroupID* and *Truncated sequence number*, namely *explicit nonce*, that are included in the adapted header in Figure 5. While the *client write IV* and the *GroupID* are the same for all the group members, it is also possible that the *Truncated sequence numbers* considered by different listener nodes become aligned with each other. This means that different listener nodes would use the same explicit nonce values when processing their unicast group responses.

Axiom preserves secure communication even if explicit nonce are reused, thanks to the derivation of individual key material associated to single listener nodes (see Sections 5.5 and 5.6). That is, two listener nodes that accidentally consider the same explicit nonce actually use *different* key material. This makes it possible to comply with AEAD ciphers like AES-CCM [D. McGrew and D. Bailey 2012] and AES-GCM [J. Salowey, A. Choudhury and D. McGrew 2008], where each value of the explicit nonce must be distinct for each invocation of the encrypt function for any fixed key. Also, it prevents an adversary from solving underlying keyed hash so making subsequent forgeries trivial, and from choosing *IV*s to produce colliding counters. Thus, any degradation of the provided security level due to a possible nonce reuse is prevented. Finally, this also prevents an external adversary from

injecting fake group responses, by spoofing the IP source address of a given listener node. Also, we recall that every listener node derives an individual key material separately for each sender node in the group. This prevents an external adversary from intercepting a group response sent to a given sender node, and then replaying it to a different sender node by spoofing the IP destination address.

7.4. Group-level security

Axiom relies on group key material commonly shared among the group members. This requires that all group members are trusted, i.e. that they do not forge messages to make them appear as sent by different entities. In many use cases, devices in the same group belong to a common authority and are configured by a commissioner. Of course, the risk that group members get compromised should also be considered. As a first thing, such a risk is reduced when multicast groups are deployed in physically secured locations, like lighting inside office buildings. Secondly, the adoption of group key management schemes should be considered, as further discussed in Section 7.6.

In case group members can not be trusted to this extent, it becomes necessary to provide source authenticity, i.e. to assure that a given message has been sent by the alleged source. This can be achieved through digital signatures, by extending the approach described in Section 5 as follows. Every listener node stores the public key of each sender node in the group, together with the respective *SenderID*. Listener nodes can be provided with the senders' public keys upon joining the group. As an alternative, a listener node can ask a trusted Certification Authority (CA) for a sender's public key, upon receiving a group request message from that sender node for the first time. Similarly, every sender node stores the public key of each listener node in the group, together with the respective IP address. Sender nodes can ask a trusted CA for a listener's public key, upon receiving a group response message from that listener node for the first time. The adoption of either source authenticity or group authenticity can be specified by means of an additional parameter in the GSA.

7.5. Uniqueness of SenderIDs

SenderIDs must be unique within a multicast group, in order to preserve the freshness of nonce values and the security of DTLS records. If multiple sender nodes were configured with the same *SenderID*, it would be necessary to explicitly prevent nonce reuse and the related security weakness. A possible mechanism consists in configuring all sender nodes also as listener nodes. This allows a sender node S1 that receives a message with the same *SenderID* in the DTLS header from a different sender node S2 to become aware of the *SenderID* conflict. Then, S1 can notify the GC, which can provide a different *SenderID* to either one of the two sender nodes or both of them.

7.6. Management of group key material

A real deployed system requires additional services for distributing and renewing key material, security parameters, and security policies in the multicast group. Also, the GC is intended as the primary responsible for providing the GSA to the group members, and assisting them upon joining and leaving the group. As described in [T. Dierks and E. Rescorla 2008], the *master secret* included in the *SecurityParameters* structure is computed using a *pre master secret*, together with the security parameters *client random* and *server random*. Therefore, the group key material can be renewed by providing the group members with a new securely generated *pre master secret* value. Then, the group members can use it to update the group security material stored in their GSA, as well as the individual key material stored in the *write group connection states* (listener nodes) and in the *read group connection states* (sender nodes). The establishment of a GSA will be part of a dedicated IETF activity, preferably based on [T. Hardjono and B. Weis 2004][M. Baugher, R. Canetti, L. Dondeti and F. Lindholm 2005][H. Harney, U. Meth, A. Colegrove and G. Gross 2006]. However, in the following we suggest a possible approach to distribute the current GSA to new group members, and distribute renewed security material to the present group members (*rekeying*). To this end, we assume that the GC is an actual group member and is configured as a sender node.

Periodic key renewal. The group key material should be periodically renewed, in order to discourage exhaustive key search or traffic analysis. By broadcasting a single DTLS multicast message

protected with the current server write parameters in the GSA, the GC can periodically distribute a new *pre master secret* to the group members. Upon receiving it, all group members can use it to update the *master secret* and the group security material stored in their GSA, as well as the individual key material stored in the respective group connection states. Periodical key renewal assumes that no group members are under an adversary's control. Therefore, the amount of time between two consecutive occurrences of periodic rekeying should be appropriately defined, by taking into account the anticipated level of threat that the group is exposed to and the specific application requirements. At the same time, the frequency according to which periodic rekeying is performed should remained affordable for the limited capabilities and resources of the IoT devices.

Backward security. In order to ensure backward security, a joining node must not be able to access the group communication that took place prior to its joining. Therefore, upon a new node's joining, first the present group members are provided with new security material, and only after that the GC provides the new updated GSA to the joining node through a secure communication channel. For instance, the joining node could retrieve a public certificate associated to the GC, use the retrieved public key 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.

Forward security. Forward secrecy must be ensured, in case a group member voluntarily leaves the group, or if it is forced to leave once found to be compromised or suspected so. That is, the leaving node must not be able to access group communication which takes place after its departure from the group, and it must be prevented from taking part in the rekeying process itself. An easy but resource exhaustive approach consists in transmitting the new *pre master secret* value separately to each remaining group member. To this end, the GC can establish a different pairwise symmetric key with each member of the group upon its joining, and then using such keys to distribute the new *pre master secret* in a one-to-one fashion. As an alternative, the GC can distribute the new *pre master secret* value by means of specific group key management schemes [C. K. Wong, M. Gouda and S. S. Lam 2000][G. Dini and M. Tiloca 2013][M. Tiloca and G. Dini 2016], which result to be more efficient and display high scalability with the number of nodes in the group.

Finally, source authenticity of rekeying messages sent by the GC must be assured. An efficient approach, suitable for constrained IoT devices, consists in determining the random part of the new *pre master secret* as the next element of a pre-computed reversed hash chain, a mechanism derived from Lamport's one-time password [L. Lamport 1981]. The main benefit is that the most recently released element in the chain can be efficiently authenticated by computing its hash and verifying that it is equal to the previously released element in the chain. Upon creating the multicast group, it is sufficient that the GC provides all group 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. After that, the group members can automatically and efficiently authenticate all other chain elements.

8. EXPERIMENTAL PERFORMANCE EVALUATION

In order to prove that our approach is feasible on constrained IoT devices and is preferable to the original IETF proposal in [S. Keoh, S. Kumar, E. Dijk and A. Rahman 2014], we made a proof of concept implementation of Axiom and performed an experimental evaluation of performance.

In particular, we considered the *Contiki* OS [con 2015], the *Erbium* CoAP library for Contiki [erb 2015], and the library *TinyDTLS* 0.5 [tin 2015a], a lightweight DTLS implementation for constrained environments. We have extended *TinyDTLS* in order to support DTLS-based secure group communication, according to either the original IETF proposal [S. Keoh, S. Kumar, E. Dijk and A. Rahman 2014] or our approach described in Section 5. Our extended library is available online at [tin 2015b]. To the best of our knowledge, it is currently the only publicly available implementation of DTLS for Contiki that also supports secure group communication. Finally, we considered resource constrained IoT devices based on the CC2538 platform [Texas Instruments 2014], featuring a 32 MHz CPU, and provided with 512 KB of ROM flash and 32 KB of RAM.

8.1. Experimental scenario

For our experimental tests, we considered one multicast group, including four IoT devices that have successfully joined the group. In particular, the group members formed an IEEE 802.15.4 network [Institute of Electrical and Electronics Engineers, Inc. 2006], organized according to a star topology and relying on IPv6 and 6LoWPAN. As shown in Figure 6, one device was configured as *sender* node, while the remaining three devices were configured as *listener* nodes.

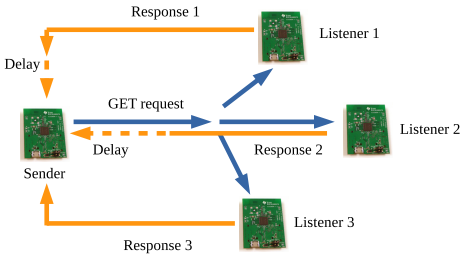


Fig. 6. The considered experimental scenario consists in a multicast group, including one sender node and three listener nodes. If the multicast communication scheme is adopted, the sender node transmits a single multicast request message to the whole group. Upon receiving it, each of the three listener nodes replies back to the sender node, by transmitting a unicast response message. In order to minimize the risk of collisions, each listener node schedules the transmission of its response message according to a delay value randomly generated within a fixed range.

In principle, the IoT devices perform the following message exchange. First, the sender node contacts the three listener nodes by means of CoAP request messages. More in detail, the sender node transmits either one different request per listener node, or a single group request, in case a unicast or multicast communication scheme is considered, respectively. In case three different requests are transmitted, the sender node waits for a response from the intended recipient, before proceeding with the next listener node. Instead, upon receiving a CoAP request from the sender node, each listener node replies with a unicast CoAP response message.

In case multicast requests are sent, each listener node observes a random delay before transmitting its response, in order to reduce the risk of collision among different listener nodes, as recommended in Section 8.2 of [Z. Shelby, K. Hartke and C. Bormann 2014]. To this end, we considered such delay as ranging from 0 to 0.25 seconds. After preliminary tests, we chose such range as still able to assure both efficient communication and absence of collisions among the three sender nodes.

In the following, we describe the five experimental configurations considered hereafter. In particular, we refer to *group transaction* as the process involving a full information exchange between the sender node and all the listener nodes in the group. In case a DTLS handshake is performed, we considered the key provisioning based on pre-shared keys (see Section 3).

• *CoAP Unicast (CoAP Uni)*. The sender node transmits an unsecure unicast request message to each listener node, and waits for an unsecure response from each listener node. The group transaction begins when the sender node starts preparing the CoAP request message for the first listener node, and ends when it finishes to process the response message from the third listener node.
• *CoAP Multicast (CoAP Mul)*. The sender node transmits one unsecure multicast group request message. Then, each listener node replies with an unsecure unicast response message. The group transaction begins when the sender node starts preparing the CoAP multicast request message, and ends when it finishes to process the third received group response message. Since the listener nodes observe a random delay before replying, the three response messages can be received by the sender node in any order.
• *DTLS Unicast Request, Unicast Response (DTLS Uni-Uni)*. The sender node first separately performs a DTLS handshake with each listener node. After all the three DTLS sessions have been established, the sender node individually transmits three secure unicast request messages to each

listener node. Then, each listener node relies on the DTLS session associated to the sender node, and replies with a secure unicast response message. The group transaction begins when the sender node starts a handshake with the first listener node, and ends when it finishes processing the response message from the third listener node.

• *DTLS Multicast Request, Unicast Response (DTLS Mul-Uni).* The sender node first separately performs a DTLS handshake with each listener node. After all the three DTLS sessions have been established, the sender node transmits one secure multicast group request message, protected according to the IETF approach described in [S. Keoh, S. Kumar, E. Dijk and A. Rahman 2014]. Then, each listener node relies on the DTLS session established with the sender node, and replies with a secure unicast response message. The group transaction begins when the sender node starts a handshake with the first listener node, and ends when it finishes processing the third received group response message.

• *DTLS Multicast Request, Multicast Response (DTLS Mul-Mul).* The sender node transmits one secure multicast group request message, protected according to our approach Axiom as described in Section 5.4. Then, each listener node replies with a unicast secure response message, protected according to our approach Axiom as described in Section 5.6. The group transaction begins when the sender node starts preparing the CoAP multicast request message, and ends when it finishes processing the third received group response message.

8.2. Experimental results

In the following, we show and discuss our results. During our experiments, we first measured the memory occupancy for every experimental configuration. Secondly, we measured the time required to complete a group transaction. Finally, we evaluated the energy consumption.

8.2.1. Memory footprint. As to the memory occupancy, we relied on the size command provided by Contiki, and measured: i) the consumption of Random Access Memory (RAM); and ii) the consumption of Read-Only Memory (ROM), i.e. the size of the overall binary image loaded on the devices. Results refer to the full binary image, including the Contiki OS and the whole communication stack. The ROM and RAM footprints are reported below in Figure 7 and Figure 8, respectively.

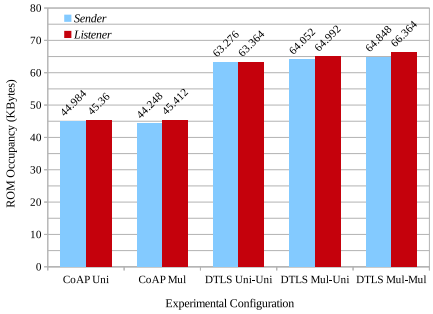


Fig. 7. The ROM occupancy becomes higher if security services are included. The three rightmost secure configurations result in a ROM occupancy higher than the one in the two leftmost unsecure configurations. Our approach Axiom in *DTLS Mul-Mul* is comparable with the original IETF approach in *DTLS Mul-Uni*.

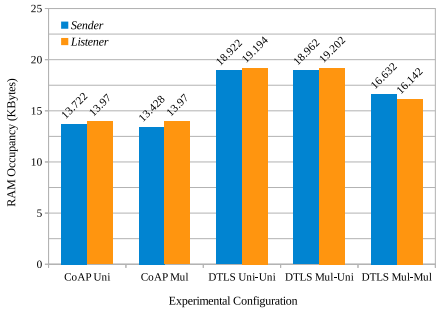


Fig. 8. The three rightmost secure configurations display a RAM occupancy higher than the one in the two unsecure configurations. Also, Axiom in *DTLS Mul-Mul* requires less RAM than the other two secure approaches, as it does not allocate additional full DTLS sessions to process group response messages.

With reference to Figure 7, the increasing trend of the ROM occupancy is due to the additional security features included in the different experimental configurations, which results in an increasing code size. In particular, the *CoAP-Uni* and *CoAP Mul* configurations do not include any security

services, and hence display a reduced ROM occupancy. Conversely, the three rightmost configurations *DTLS Uni-Uni*, *DTLS Mul-Uni* and *DTLS Mul-Mul* require all a greater amount of ROM, mostly due to the presence of DTLS. Besides, they all result in a comparable memory occupancy, and our approach Axiom in *DTLS Mul-Mul* requires 0.796 KB and 1.372 KB more than the original IETF proposal in *DTLS Mul-Uni*, on the sender and listener side, respectively.

However, it is even more important to limit the RAM occupancy, due to its very scarce availability on IoT devices. As shown in Figure 8, we can observe that the two unsecure configurations *CoAP Uni* and *CoAP Mul* result in a reduced memory occupancy, due to the absence of security services, and thus a smaller amount of data structures allocated at runtime. Conversely, the three secure configurations *DTLS Uni-Uni*, *DTLS Mul-Uni* and *DTLS Mul-Mul* result all in a greater RAM occupancy, mostly due to the allocation of security contexts and data structures. However, note that our approach Axiom in *DTLS Mul-Mul* is the one displaying the smallest RAM occupancy among the three secure configurations. In particular, it requires 2.33 KB and 3.06 KB of RAM less than the original IETF proposal in *DTLS Mul-Uni*, on the sender and listener side, respectively. This is mainly due to the fact that Axiom does not require to allocate any additional full DTLS sessions to support secure unicast communication between the sender node and the listener nodes.

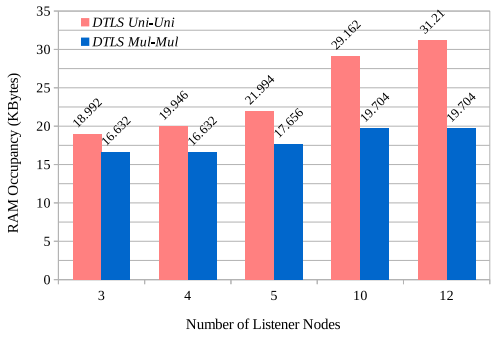


Fig. 9. The RAM occupancy in the sender node increases with the number of listener nodes in the group. The increasing trend is much slower for our approach Axiom in *DTLS Mul-Mul* than in the full unicast approach in *DTLS Uni-Uni*. This makes Axiom highly scalable with the group size, and more convenient in the presence of several listener nodes.

Finally, we show that the RAM occupancy due to our approach Axiom is highly scalable with the number of listener nodes in group. We consider how this affects RAM occupancy only on the sender side, since listener nodes are not affected at all by the group size. Figure 9 depicts the amount of RAM required on the sender node, while increasing the number of listener nodes in the group. In particular, we compare the RAM occupancy in case either the full unicast secure scheme *DTLS Uni-Uni* or our approach Axiom in *DTLS Mul-Mul* is adopted. Note that the original IETF approach in the *DTLS Mul-Uni* configuration does not perform better than *DTLS Uni-Uni*, due to the allocation of additional security contexts and data structures to process the multicast request messages.

As shown in Figure 9, the configuration *DTLS Mul-Mul* always results in a reduced RAM occupancy. Also, our approach Axiom becomes more and more convenient than the full unicast secure scheme in *DTLS Uni-Uni*, as the number of listener nodes increases. This proves that Axiom is highly scalable with the group size, and is particularly preferable in the presence of several listener nodes. In addition, the *DTLS Uni-Uni* configuration practically requires the whole available RAM on the considered devices, i.e. 32 KB, in the presence of only 12 listener nodes. Instead, the *DTLS Mul-Mul* configuration considering Axiom requires a comparable amount of RAM, i.e. 31.992 KB, in the presence of 40 listener nodes. This means that, given the same hardware platform, Axiom makes it possible to deploy a group including a greater number of nodes.

8.2.2. Communication performance. In order to compare the communication performance of the different configurations, we measured the time spent on the sender node side to complete a first initial group transaction with the three listener nodes. For each configuration, we ran 20 independent experiments, and computed the average of collected results and the related standard deviation.

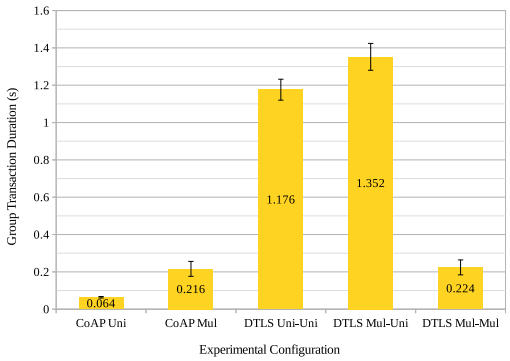


Fig. 10. Time experienced by the sender node to complete a full initial interaction with the three listener nodes. With respect to the unsecure configurations *CoAP Uni* and *CoAP Mul*, the two secure configurations *DTLS Uni-Uni* and *DTLS Mul-Uni* result in a significantly longer group transaction duration, mostly because of the three DTLS handshakes. Conversely, our secure approach Axiom in *DTLS Mul-Mul* is even comparable with the unsecure configuration *CoAP Mul*, and thus outperforms the alternative secure approaches in *DTLS Uni-Uni* and *DTLS Mul-Uni*.

Figure 10 shows the group transaction durations for each experimental configuration. First of all, we observe that the *CoAP Mul* configuration results in a group transaction duration which is three times bigger than in the *CoAP Uni* configuration. This is mainly due to the listener nodes relying on random delays for transmitting response messages in the *CoAP Mul* configuration. In principle, it is possible to achieve better performance and reduce the group transaction duration by properly tuning such transmission delays. The adoption of improvement techniques aimed at optimizing the transmission of unicast responses is out of the scope of this paper.

The two secure configurations *DTLS Uni-Uni* and *DTLS Mul-Uni* result in a much longer group transaction duration. This is mainly due to the performance of the DTLS handshakes between the sender node and the three listener nodes. Of course, the group transaction duration would become more and more long, in the presence of more listener nodes. Also, it is worth noting that the original IETF approach considered in *DTLS Mul-Uni* results in a longer group transaction length than the full unicast approach considered in *DTLS Uni-Uni*. As discussed above for the two unsecure configurations, this is mainly due to the adoption of random delays for transmitting response messages.

Conversely, our approach Axiom considered in the *DTLS Mul-Mul* configuration outperforms the alternative secure approaches. That is, it displays a group transaction duration which is five times shorter than in *DTLS Uni-Uni* considering the full unicast approach, and about six times shorter than in *DTLS Mul-Uni* considering the original IETF approach. Furthermore, while providing secure communication, Axiom displays a group transaction time which is even comparable with the one of the equivalent unsecure configuration *CoAP Mul-Mul*.

Table I reports additional detailed results, referred to the three secure configurations *DTLS Uni-Uni*, *DTLS Mul-Uni*, and *DTLS Mul-Mul*. In particular, we show i) the time spent on the sender node to complete the DTLS handshake, separately with each listener node; and ii) the overall amount of time spent on the sender node to complete the actual exchange of request and response messages with the three listener nodes. As we can observe, performing the handshake with the first listener node takes about twice the time than with the other two listener nodes. This is due to the fact that the sender node has to initialize a number of security contexts and data structures, in order to fully enable the DTLS layer and performs the first handshake. Furthermore, the actual exchange

Table I. Time required to perform the DTLS handshakes and the CoAP message exchange between the sender node and the listener nodes. In *DTLS Uni-Uni* and *DTLS Mul-Uni*, the first handhshake takes a longer time, due to the initialization of security context and data structures. In *DTLS Mul-Uni* and *DTLS Mul-Mul*, the CoAP message exchange is completed in a comparable amount of time. This means that the derivation of individual key material performed by Axiom in *DTLS Mul-Mul* does not significantly impact on the message exchange duration.

	DTLS handshake with Listener 1	**DTLS handshake with Listener 2**	**DTLS handshake with Listener 3**	**CoAP message exchange**
DTLS Uni-Uni	0.552 s $\pm$ 0.032 s	0.264 s $\pm$ 0.032 s	0.280 s $\pm$ 0.040 s	0.080 s $\pm$ 0.004 s
DTLS Mul-Uni	0.560 s $\pm$ 0.024 s	0.280 s $\pm$ 0.032 s	0.296 s $\pm$ 0.032 s	0.208 s $\pm$ 0.040 s
DTLS Mul-Mul	-	-	-	0.224 s $\pm$ 0.040 s

of request and response messages in the *DTLS Uni-Uni* configuration requires only 0.016 seconds more than the whole group transaction duration in the *CoAP Uni* configuration, i.e. 0.064 seconds. This indicates that secure communication results in a reasonable 25% communication overhead. Finally, the exchange of request and response messages requires a comparable amount of time for the configurations *DTLS Mul-Uni* and *DTLS Mul-Mul*. This means that the derivation of individual key material considered in Axiom does not result in any considerable impact on performance.

We would like to point out that results in Figure 10 and Table I refer to the *first*, initial, group transaction that the sender node performs with the three listener nodes. That is, starting from the following group transaction, the same sender node does not need to perform any further handshake with the same listener nodes. Hence, the two secure multicast configurations *DTLS Mul-Uni* and *DTLS Mul-Mul* display comparable performance from then on. However, the same performance trend and benefits of our approach Axiom can be observed every time new nodes join the multicast group and start to exchange secure messages. This makes Axiom particularly convenient to adopt in the presence of large-scale multicast groups, especially if with highly dynamic membership.

With reference to the secure configurations *DTLS Uni-Uni* and *DTLS Mul-Uni*, a new listener node would be required to perform a DTLS handshake by all sender nodes in the group. Even worse, a new sender node would perform a DTLS handshake with all currently present listener nodes, which are supposed to be considerably more in number. In both cases, this may significantly reduce the overall network responsiveness and availability, and would force the adoption of additional mechanisms to keep sender nodes aware of new listener nodes. Instead, our approach Axiom considered in the *DTLS Mul-Mul* configuration does not require performing any DTLS handshake. Hence, it has no particular impact on the overall network performance and availability, and allows sender nodes to remain agnostic of the current listener nodes in the group.

8.2.3. Energy consumption. We measured energy consumption during a group transaction as the sum of three different contributions: i) the energy spent by the CPU, i.e. E_{CPU}; ii) the energy spent by the radio interface in transmission mode, i.e. E_{TX}; and iii) the energy spent by the radio interface in reception mode, i.e. E_{RX}. We computed each single contribution as the product between the related power consumption reported in [Texas Instruments 2014] and the related active time collected by means of the *Energest* framework provided by Contiki. Energest has been proven to accurately estimate energy consumption, while increasing the computing time only of the 0.7% [A. Dunkels, F. Österlind, N. Tsiftes and Z. He 2007]. Furthermore, we considered the worst case when devices do not rely on the *Radio Duty Cycling* mechanism provided by Contiki [rdc 2015] for switching off the radio interface when not in use. Note that, despite obvious advantages in terms of energy saving, such mechanisms result also in a non negligible packet loss, which can significantly worse communication performance. The proper tuning of energy saving mechanisms is actually related to network performance optimization, and thus is out of the scope of this paper.

Figure 11 and Figure 12 show the overall energy consumption in a group transaction on the sender node and on one of the listener nodes, respectively. We did not observe any relevant difference between the different listener nodes. Error bars are too small to be appreciated and thus are not included. Furthermore, Table II reports the different energy contributions in detail. We refer to *Key material operations* as the set of actions including key retrieval, key derivation, and DTLS

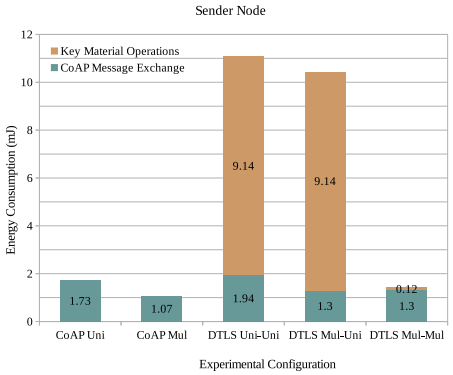


Fig. 11. Energy consumed by the sender node during an initial group transaction. The higher consumption in *DTLS Uni-Uni* and *DTLS Mul-Uni* is mostly due to the DTLS handshake with the listener nodes. Our approach Axiom in *DTLS Mul-Mul* displays an energy consumption which is 7 times lower than the one of the other two secure approaches in *DTLS Uni-Uni* and *DTLS Mul-Uni*, and even comparable with the one displayed by the unsecure multicast approach adopted in *CoAP Mul*.

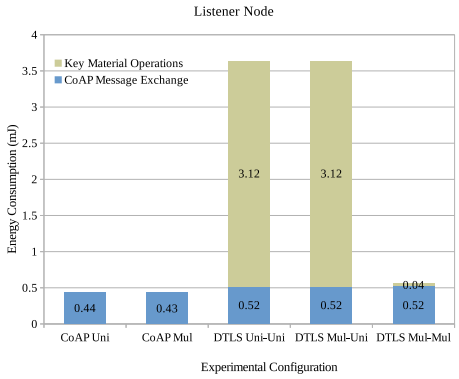


Fig. 12. Energy consumed by a listener node during an initial group transaction. The higher consumption in *DTLS Uni-Uni* and *DTLS Mul-Uni* is mostly due to the DTLS handshake with the sender nodes. Our approach Axiom in *DTLS Mul-Mul* displays an energy consumption which is 6 times lower than the one of the other two secure approaches in *DTLS Uni-Uni* and *DTLS Mul-Uni*, and even comparable with the one displayed by the unsecure multicast approach adopted in *CoAP Mul*.

Table II. Detailed energy consumption on the sender node and the listener nodes.

Experimental configuration	**Energy Consumption (mJ)**											
	Sender node						**Listener node**					
	Key material operations			**CoAP Message exchange**			**Key material operations**			**CoAP Message exchange**		
	E_{CPU}	E_{TX}	E_{RX}	E_{CPU}	E_{TX}	E_{RX}	E_{CPU}	E_{TX}	E_{RX}	E_{CPU}	E_{TX}	E_{RX}
CoAP Uni	-	-	-	0.65	0.67	0.40	-	-	-	0.10	0.15	0.19
CoAP Mul	-	-	-	0.46	0.21	0.40	-	-	-	0.10	0.15	0.19
DTLS Uni-Uni	2.09	3.91	3.14	0.77	0.61	0.56	0.78	1.26	1.08	0.13	0.22	0.17
DTLS Mul-Uni	2.09	3.91	3.14	0.54	0.20	0.56	0.78	1.26	1.08	0.14	0.22	0.16
DTLS Mul-Mul	0.12	-	-	0.54	0.20	0.56	0.04	-	-	0.14	0.22	0.16

handshake (if relevant). Also, we refer to *CoAP message exchange* as the set of actions related to the transmitted/received CoAP messages, including their security processing (if relevant).

With reference to Figure 11 and the two unsecure approaches, we observe that the *CoAP Uni* configuration displays an energy consumption which is roughly 50% higher than the one in the *CoAP Mul* configuration. This is due to the transmission of three different unicast request messages, rather than a single multicast message. Of course, this difference would grow as the number of listener nodes increases. The two secure configurations *DTLS Uni-Uni* and *DTLS Mul-Uni* result in a considerable higher energy consumption. This is mainly due to the performance of the DTLS handshake with the three listener nodes, as well as to the additional processing and transmission/reception of the CoAP messages as DTLS records. Also in this case, the *DTLS Mul-Uni* configuration displays an energy consumption which is slightly lower than the one for the *DTLS Uni-Uni* configuration, because of the transmission of a single multicast request message, rather than three unicast request messages. However, our approach Axiom adopted in the *DTLS Mul-Mul* configuration does not require to perform any handshake with the listener nodes. Thus, the impact of security is limited to the transmission/reception of expanded CoAP messages as DTLS records, and the execution of lightweight key material operations. It follows that Axiom displays an energy consumption which is 7 times lower than the one of the other two secure approaches, and even comparable with the one displayed by the unsecure multicast approach adopted in *CoAP Mul*.

Now, let us consider the energy consumption on the listener node side. As shown in Figure 12, the two unsecure approaches in *CoAP Uni* and *CoAP Mul* display the same energy consumption. This is consistent with the fact that every listener node simply replies with a response message, regardless

the previous reception of a unicast or multicast request message. For the same reason, the three secure approaches in *DTLS Uni-Uni*, *DTLS Mul-Uni* and *DTLS Mul-Mul* display the same energy consumption as to the actual CoAP message exchange. Also, the two configurations *DTLS Uni-Uni* and *DTLS Mul-Uni* display the same energy consumption as to the key material operations, due to the performance of the DTLS handshake with the sender node. However, our approach Axiom adopted in the *DTLS Mul-Mul* configuration does not require to perform any handshake with the sender node. Thus, the impact of security is limited to the transmission/reception of the expanded CoAP response message as DTLS records, and the execution of lightweight operations to generate key material. It follows that Axiom displays an energy consumption which is 6 times lower than the one of the other two secure approaches, and even comparable with the one displayed by the two unsecure approaches adopted in *CoAP Uni* and *CoAP Mul*.

As discussed also in Section 8.2.2, results in Figures 11 and 12 refer to the *first*, initial, group transaction that the sender node performs with the three listener nodes. That is, starting from the following group transaction, the same sender node does not need to perform any further handshake with the same listener nodes. As a consequence, the two secure multicast configurations *DTLS Mul-Uni* and *DTLS Mul-Mul* display comparable performance from then on. However, the same performance trend and benefits of our approach can be observed every time new nodes join the multicast group and start to exchange secure messages. This makes Axiom particularly convenient to adopt in the presence of large-scale multicast groups, especially if with highly dynamic membership.

9. CONCLUSION

We have presented Axiom, our DTLS-based approach to secure multicast communication in groups of resource constrained IoT devices. Axiom provides an adaptation of the DTLS record layer, exploits common key material shared among the group members to efficiently protect multicast messages and related unicast reply messages, and does not require performing any DTLS handshake. We have made a proof of concept implementation of Axiom for the Contiki OS and experimentally evaluated our approach on real IoT hardware platforms. Our results show that Axiom is affordable on resource constrained IoT platforms and outperforms previously proposed approaches, in terms of memory occupancy, communication overhead and energy consumption.

Acknowledgment

The authors sincerely thank the anonymous referees and the editor for their insightful comments and suggestions that helped to considerably improve the technical quality of the paper. This project has received funding from the EU's FP7 programme for research, technological development and demonstration under grant agreement no. 607109, the EU H2020 project NobelGrid under grant no. 646184, VINNOVA, the EIT Digital HII project ACTIVE, and a Swedish Institute scholarship.

REFERENCES

2015. Contiki: The Open Source Operating System for the Internet of Things. (2015). http://www.contiki-os.org/

2015. Erbium (Er) REST Engine - C CoAP Implementation. (2015). http://people.inf.ethz.ch/mkovatsc/erbium.php

2015. Radio duty cycling - Contiki Wiki. (2015). https://github.com/contiki-os/contiki/wiki/Radio-duty-cycling

2015a. tinyDTLS. (2015). http://sourceforge.net/projects/tinydtls/

2015b. tinygroupdtls. (2015). https://github.com/nikirill/tinygroupdtls

A. Dunkels, F. Österlind, N. Tsiftes and Z. He. 2007. Software-based On-line Energy Estimation for Sensor Nodes. In *Proceedings of the 4th Workshop on Embedded Networked Sensors (EmNets '07)*. ACM, New York, NY, USA, 28–32.

A. Rahman and E. Dijk. 2014. *RFC 7390 - Group Communication for the Constrained Application Protocol (CoAP)*. Internet Engineering Task Force.

B. Briscoe. 2010. *RFC 6040 - Tunnelling of Explicit Congestion Notification*. Internet Engineering Task Force.

B. Weis, G. Gross and D. Ignjatic. 2008. *RFC 5374 - Multicast Extensions to the Security Architecture for the Internet Protocol*. Internet Engineering Task Force.

C. K. Wong, M. Gouda and S. S. Lam. 2000. Secure group communications using key graphs. *IEEE/ACM Transactions on Networking* 8, 1 (2000), 16–30.

C. Kaufman, P. Hoffman, Y. Nir, P. Eronen and T. Kivinen. 2014. *RFC 7296 - Internet Key Exchange Protocol Version 2 (IKEv2)*. Internet Engineering Task Force.

D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley and W. Polk. 2008. *RFC 5280 - Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*. Internet Engineering Task Force.

D. McGrew. 2008. *RFC 5116 - An Interface and Algorithms for Authenticated Encryption*. Internet Engineering Task Force.

D. McGrew and D. Bailey. 2012. *RFC 6655 - AES-CCM Cipher Suites for Transport Layer Security (TLS)*. Internet Engineering Task Force.

D. McGrew and J. Foley. 2014. *Authenticated Encryption with Replay prOtection (AERO), draft-mcgrew-aero-01 (work in progress)*. Internet Engineering Task Force.

E. Rescorla and N. Modadugu. 2012. *RFC 6347 - Datagram Transport Layer Security Version 1.2*. Internet Engineering Task Force.

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

G. Kortuem, F. Kawsar, D. Fitton and V. Sundramoorthy. 2010. Smart objects as building blocks for the Internet of things. *IEEE Internet Computing* 14, 1 (2010), 44–51.

H. Harney, U. Meth, A. Colegrove and G. Gross. 2006. *RFC 4535 - GSAKMP: Group Secure Association Key Management Protocol*. Internet Engineering Task Force.

H. Tschofenig and T. Fossati. 2016. *RFC 7925 - Transport Layer Security (TLS) / Datagram Transport Layer Security (DTLS) Profiles for the Internet of Things*. Internet Engineering Task Force.

Institute of Electrical and Electronics Engineers, Inc. 2006. *IEEE Std. 802.15.4-2006, IEEE Standard for Information technology*. Institute of Electrical and Electronics Engineers, Inc., New York, NY, USA.

J. Hui and P. Thubert. 2011. *RFC 6282 - Compression Format for IPv6 Datagrams over IEEE 802.15.4-Based Networks*. Internet Engineering Task Force.

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

L. Atzori, A. Iera and G. Morabito. 2010. The Internet of Things: A survey. *Computer Networks* 54, 15 (2010), 2787–2805.

L. Lamport. 1981. Password authentication with insecure communication. *Communications of the ACM* 24, 11 (1981), 770–772.

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

M. Tiloca. 2014. Efficient Protection of Response Messages in DTLS-Based Secure Multicast Communication. In *Proceedings of the 7th International Conference on Security of Information and Networks (SIN 2014)*. ACM, New York, NY, USA, 466–472.

M. Tiloca and G. Dini. 2016. GREP: a Group REkeying Protocol Based on Member Join History. In *Proceedings of the twenty-first IEEE Symposium on Computers and Communications (ISCC 2016)*. IEEE, New York, NY, USA, 326–333.

M. Tiloca, S. Raza, K. Nikitin, S. Kumar. 2015. *Secure Two-Way DTLS-Based Group Communication in the IoT, draft-tiloca-dice-secure-groupcomm-00 (work in progress)*. Internet Engineering Task Force.

P. Eronen P. and H. Tschofenig. 2005. *RFC 4279 - Pre-Shared Key Ciphersuites for Transport Layer Security (TLS)*. Internet Engineering Task Force.

P. Savola. 2008. *RFC 5110 - Overview of the Internet Multicast Routing Architecture*. Internet Engineering Task Force.

P. Wouters, H. Tschofenig, J. Gilmore, S. Weiler and T. Kivinen. 2014. *RFC 7250 - Using Raw Public Keys in Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)*. Internet Engineering Task Force.

R. Shirey. 2007. *RFC 4949 - Internet Security Glossary, Version 2*. Internet Engineering Task Force.

S. Kent and K. Seo. 2005. *RFC 4301 - Security Architecture for the Internet Protocol*. Internet Engineering Task Force.

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

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

T. Hardjono and B. Weis. 2004. *RFC 3740 - The Multicast Group Security Architecture*. Internet Engineering Task Force.

T. Winter, P. Thubert, A. Brandt, J. Hui, R. Kelsey, P. Levis, K. Pister, R. Struik, JP. Vasseur and R. Alexander. 2012. *RFC 6550 - RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks*. Internet Engineering Task Force.

Texas Instruments. 2014. *CC2538 Powerful System-On-Chip for 2.4-GHz IEEE 802.15.4, 6LoWPAN and ZigBee Applications*. Texas Instruments Inc. http://www.ti.com/lit/gpn/cc2538

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