tls R. Lucas
Internet Draft Cisco International Limited
Intended status: Standards Track September 13, 2017
Expires: March 17, 2018

DTLS Multicast
draft-lucas-dtls-multicast-00.txt

Abstract

This proposal to provide a secure multicast 1-to-N or M-to-N device capability, with the same level of reliability as the underlying multicast network, also aims to be light-weight and supported by a very constrained device. Guaranteed reliability would be provided by an additional protocol working in co-operation with it.

The aim is to support end to end secure communications in the edge device world of IoT where the transport methods will vary or at least change once the IP realm is left. Hence there is no dependence on Ipv6 or IP or CoAP and no restrictions that might be introduced if too specific an end node application was implied. It is network independent, it just must be possible to transmit and receive frames in multicast.

This can be achieved with simply a minimal change to the DTLS behavior and using current DTLS libraries. DTLS headers are not changed, additional headers are used in the packets before the DTLS traffic.

DTLS Multicast keeps the layer concept pure and independent, hence it can be used for routing something that is not CoAP.

Contents

1. Introduction

This proposal provides a secure multicast M-to-N device (or 1-to-N) capability with the same level of reliability as the underlying multicast network. This proposal does not provide guaranteed reliability by itself, this would be provided by an additional protocol working in co-operation with it.

There is no assumption that the DTLS will be transported over IPv6 or IPv4 or even IP at all. This allows the method to be used both inside and outside the IP realm making it very useful for end to end security in the world of IoT and edge field devices that may use other transport methods.

Machines from the most powerful servers to very constrained edge devices may be connecting via several different indirect methods hence all communication must take place inside the multicast domain and a DTLS multicast group must handle additional messaging between the machines in that group. Importantly, to do this, the proposal maintains close compatibility with the existing DTLS protocol with the simple exception of using additional headers in the packets before the DTLS traffic.

The header structures and message definitions are defined herein followed by descriptions of how to use them. (There is no actual implementation code.)

This proposal provides solutions to assure the following and explains them in more detail in chapter five:-

- Establishment of a GSA (Group Security Association, see RFC 3740) where all group members use the same multicast data security

ciphersuite.
- Forward security. Ex-members can no longer decrypt group messages nor send new encrypted messages
- Backward confidentiality. A new members cannot decrypt messages sent before it became a member.
- Support for both 1-to-N and M-to-N topologies.
- Group size is limited only by RAM constraints.
- Multicast data confidentiality.
- Multicast data replays are protected. Can be detected and ignored.
- Multicast data group authentication. Only guarantee data integrity if all members are trusted.

Note that this proposal does not provide a solution for source authentication and data integrity. Authentication knows which group sent a message, not which member sent it.

2. Background

DTLS is currently a point-to-point communications protocol. In the past there have been draft proposals to expand this to be multicast and to deal with security. Notably, and listed under informative references, keoh-tls-multicast-security and keoh-dice-multicast-security which have both expired. They are in the author's opinion interesting proposals but they assume DTLS over IPv6 (see keoh-dice-multicast-security-08 sec 4.4) which may not always be true. DTLS cannot be assumed to be transported over IPv6 (or even IP for that matter). KEOH-DICE also breaks compatibility with the existing DTLS header.

This DTLS-Multicast draft proposal seeks to be network independent and to use existing headers.

3. Restrictions / Assumptions

Any machine may be connecting to a DTLS multicast group via a gateway (NAT, firewall or even protocol translator) so all communication must take place within the multicast domain only.

A DTLS multicast group must therefore handle any additional messaging to allow for the agreement of any keys, configuration data and behaviours between the machines in the group.

4. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

In this document, these words will appear with that interpretation

only when in ALL CAPS. Lower case uses of these words are not to be interpreted as carrying significance described in RFC 2119.

In this document, the characters ">>" preceding an indented line(s) indicates a statement using the key words listed above. This convention aids reviewers in quickly identifying or finding the portions of this RFC covered by these keywords.

This specification also uses the following terminology:

- Group Controller (or just Controller): The entity that is responsible for creating a multicast group, establishing security associations among authorized group members and renewing/updating the security associations.
  Note that a controller (or member capable of being a controller) must also be a sender.

- Sender: The Sender is an entity that sends data to the multicast group. In a 1-to-N multicast group only a single sender transmits data to the group. In an M-to-N multicast group (where M <= N), M group members are senders. All senders must also be listeners.

- Listener: A Listener is an entity that receives multicast messages from a multicast group. All listeners must be Members.

- Member: An entity that has joined the group.

- (SA) Security Association: A set of policy and cryptographic keys that provide security services to network traffic that matches that policy.
  Note that different types of traffic in the multicast group may be protected by different security associations.

- Group Security Association (GSA): A set of Security Associations (SAs) that together define how a group communicates securely [RFC3740].

- Keying material: Data that is specified as part of the SA which is needed to establish and maintain a cryptographic security association, such as keys, key pairs, and IVs [RFC4949].

5. DTLS Multicast Proposal

This proposal provides a secure multicast M-to-N device capability with the same level of reliability as the underlying multicast network.

This proposal does not provide guaranteed reliability by itself, this would be provided by an additional protocol working in co-operation with it.

This proposal aims to be as light-weight as possible so that it can be supported (in its minimal form) by a very constrained device.

This proposal maintains compatibility with the existing DTLS protocol with the exception of additional headers in the packets before the DTLS traffic.

This proposal provides a solution for:

a. Establishment of a GSA: A secure mechanism to distribute keying materials, multicast security policies and security parameters to members of the multicast group.

b. Multicast data security ciphersuite: All group members use the same ciphersuite to protect the authenticity, integrity and confidentiality of multicast messages. The ciphersuite is part of the GSA.

c. Forward security: Devices that leave the group will not have access to any future GSAs. This ensures that a past member device cannot continue to decrypt confidential data that is sent in the group. It also ensures that this device cannot send encrypted and/or integrity protected data after it leaves the group.
The GSA update mechanism is part of the key management scheme. Note that the controller can weaken this as part of the key management policy for performance reasons and/or to reduce network traffic overhead.

d. Backward confidentiality: A new device joining the group will not have access to any old GSAs. This ensures that a new member device cannot decrypt data sent before it joins the group. The key management scheme should ensure that the GSA is updated to ensure backward confidentiality. Note that the controller can weaken this as part of the key management policy for performance reasons and/or to reduce network traffic overhead.

e. Multicast communication topology: Both 1-to-N (one sender with multiple listeners) and M-to-N (multiple senders with multiple listeners) communication topologies are supported

f. Multicast group size: The number of listeners is limited by the RAM on the group controller. The protocol allows for up to 2^128 unique listeners.
The number of senders is limited by the RAM on the listeners. The protocol allows for up to 2^31 unique senders.
Note that all Senders must also be Listeners.

g. Multicast data confidentiality: Multicast messages can be encrypted according to the cipher suite selected by the

controller.

h. Multicast data replay protection: Replayed messages can be detected and ignored.

i. Multicast data group authentication: Multicast messages can be authenticated according to the cipher suite selected by the Group Controller. The multicast group key (which is known to all group members) is used to provide authenticity to the multicast messages. It does not guarantee data integrity unless all group members are trusted.

This proposal does not provide a solution for:

a. Source authentication and data integrity: Messages can be authenticated to have come from a member of the group, but cannot be tracked to a specific member within the group.

6. Significant differences between DTLS Multicast and DTLS Unicast

Additional headers are added before the DTLS and other encrypted traffic to allow the traffic types to be identified.

The existing DTLS (unicast) standard allows either party to trigger a key rotate and associated epoch change. DTLS Multicast allows this on the CLIENT channel but key rotation and epoch change on all other channels are managed by the group controller.

Message acknowledgement and retry is not supported by any channel apart from the CLIENT channels. If the network is lossy then the listeners may not receive all the control or data messages. Any protocol using DTLS multicast must be designed to expect this and act appropriately.

7. Logical Traffic Types become Channels

To allow both control and data traffic to be carried on the same multicast group, this proposal inserts an additional header at the start of the data frame to allow the different logical traffic types to be separated into "channels".

This header is defined as:

```
struct {
    unit32 channel;
} DTLSMulticast;
```

Channels are assigned as follows:

Channel range	Name	Description
0	CLIENT	Client traffic channels

1	ELECTION	Controller election channel
2	CONTROL	Controller management channel
3 .. 0xFFFF_FFFF	SUBGROUP/SENDER	Data channels

7.1 CLIENT channel format

The CLIENT channel is used by members to connect to the multicast group and communicate with the group controller.

Messages on the CLIENT channel start with a DTLSMulticastClientPrefix header to allow individual client's connections to be identified.
This is defined as:

```
 struct {
     uint8  client[16];
 } DTLSMulticastClientPrefix;
```

The "client" value should be chosen randomly each time an entity attempts to join the multicast group.

Normal DTLS traffic follows the DTLSMulticastClientPrefix header.

The encrypted data frames are DTLSMulticastClient messages.

Apart from the additional DTLSMulticast and DTLSMulticastClientPrefix headers, the CLIENT channel is a normal DTLS connection.

7.2 ELECTION channel format

The ELECTION channel is used to handle election of controllers when the Active controller fails and controller election has been permitted for the group.

Messages on the ELECTION channel are sent as a DTLSCiphertext using The Election security association. The election security association is provided to the member in a DTLSMulticastAddSA message over the CLIENT, CONTROL or SUBGROUP channels.

The encrypted data frames are DTLSMulticastElection messages.

7.3 CONTROL channel format

The CONTROL channel is used for key distribution and other Management actions by the controller.

Messages on the CONTROL channel are sent as a DTLSCiphertext using the Group security association. The group security association is initially provided to the member in a DTLSMulticastAddSA message over the CLIENT channel. The controller may update the group

security association using a DTLSMulticastAddSA message over the CLIENT, CONTROL or SUBGROUP channels.

The encrypted data frames are DTLSMulticastControl messages.

7.4 SUBGROUP channel format

The SUBGROUP channels are used by the controller to restrict key distribution and other management actions to the members of the sub-group.

Messages on the SUBGROUP channel are sent as a DTLSCiphertext using the Specific sub-group security association. The sub-group security association is initially provided to the member in a DTLSMulticastAddSA message over the CLIENT channel. The controller may update the group security association using a DTLSMulticastAddSA message over the CLIENT, CONTROL or SUBGROUP channels.

The encrypted data frames are DTLSMulticastControl messages.

7.5 SENDER channel format

The SENDER channels are used to carry the application data. A member with transmit privileges will only be allowed to transmit on a singleSENDER channel and will be the only member allowed to send on that channel.

Messages on the SENDER channel are sent as a DTLSCiphertext using the group security association. The group security association is initially provided to the member in a DTLSMulticastAddSA message over the CLIENT channel. The controller may update the group security association using a DTLSMulticastAddSA message over the CLIENT, CONTROL or SUBGROUP channels.

Note: The encrypted data frames are application data - their format is NOT defined in this standard and varies according to the application.

8. Message definitions

```
enum {
    radius(0), joinReq(1), joinRsp(2), leave(3), acknack(4),
    addsa(5), dropsa(6), reconnect(7), heartbeatReq(8),
    heartbeatRsp(9), reqsa(10),
        (255)
} DTLSMulticastType;

struct {
    uint8  code;
    uint8  identifier;
    uint16 length;
```

```
        uint8  data[length - 4];
    } DTLSMulticastRADIUS;

    struct {
        uint32 sender_channel;  // Current sender channel (0 if
                                // listener only)
        uint32 max_subgroups;   // Maximum number of subgroups
                                // supported
        uint8  flags;           // Mode and capability flags
                                //   xxxx xxx0 "NE": Member cannot
                                //   take part in controller elections
                                //   xxxx xxx1 "EL": Member can take
                                //   part in controller elections
                                //   xxxx xx0x "RX": Member wishes to
                                //   listen only
                                //   xxxx xx1x "TX": Member wishes to
                                //   be a sender
                                //   0000 00xx "ZZ": For future
                                //   expansion
    } DTLSMulticastJoinReq;

    struct {
        uint32 firstSenderChannel;  // First sender channel (inclusive)
        uint32 lastSenderChannel;   // Last sender channel (inclusive)
    } DTLSMulticastJoinRsp;

    struct {
      // Empty structure
    } DTLSMulticastLeave;

    enum {
        noError (0),
        unknown (1),
        unauthorised (2),
        resourcesExceeded (3),
        notJoined (4),
        (255)
    } DTLSMulticastErrorCodes;

    struct {
        DTLSMulticastErrorCodes error;
        uint8 subcode;
    } DTLSMulticastAckNack;

    struct {
        uint16 epoch;           // DTLS multicast epoch for the key
        uint8  flags;           // Mode and capability flags
                                //   xxxx x000 "EL":  SA is for the
                                //   election channel
                                //   xxxx x001 "GP":  SA is the group
                                //   SA
```

```
                            //   xxxx x010 "SG":  SA is for a
                            //   subgroup channel
                            //   xxxx x011     :  Reserved for
                            //   future expansion
                            //         ...          ...
                            //   xxxx x111     :  Reserved for
                            //   future expansion
>>                          //   xxxx 0xxx "RX":  Member MUST NOT
                            //   send on the channel
>>                          //   xxxx 1xxx "TX":  Member MAY send
                            //   on the channel
                            //   0000 xxxx "ZZ":  For future
                            //   expansion, must be zero
        uint8  timeout;     // Channel timeout period (sec)
        uint32 channel;     // Channel number
        SecurityParameters key; // Encryption/MAC parameters, see
                            // RFC5246:A.6
    } DTLSMulticastAddSA;

    struct {
        uint32 channel;
    } DTLSMulticastDropSA;

    struct {
        uint32 controller_magic;
    } DTLSMulticastReconnect;

    struct {
      uint32 magic;             // Random number to match Req/Rsp
    } DTLSMulticastHeartbeatReq;

    struct {
      uint32 magic;             // Echo of
      DTLSMulticastHeartbeatReq.magic
    } DTLSMulticastHeartbeatRsp;

    struct {
      uint32 channel;
    } DTLSMulticastReqSA;


    //---------------------------------------------------------------

    enum {
        addsa, dropsa, reconnect, (255)
    } DTLSMulticastControlType;

    struct {
        DTLSMulticastControlType type;
        select (type) {
            case addsa:         DTLSMulticastAddSA;
```

```
            case dropsa:        DTLSMulticastDropSA;
            case reconnect:     DTLSMulticastReconnect;
        } content;
    } DTLSMulticastControl;

    //---------------------------------------------------------------

    enum {
        radius, addsa, dropsa, join, leave, heartbeatReq, heartbeatRsp,
        acknack, reqsa, (255)
    } DTLSMulticastClientType;

    struct {
        DTLSMulticastClientType type;
        select (type) {
            case radius:        DTLSMulticastRADIUS;
            case addsa:         DTLSMulticastAddSA;
            case dropsa:        DTLSMulticastDropSA;
            case join:          DTLSMulticastJoin;
            case leave:         DTLSMulticastLeave;
            case heartbeatReq:  DTLSMulticastHeartbeatReq;
            case heartbeatRsp:  DTLSMulticastHeartbeatRsp;
            case acknack:       DTLSMulticastAckNack;
            case reqsa:         DTLSMulticastReqSA;
        } content;
    } DTLSMulticastClient;

    // --------------------------------------------------------------

    struct {
        uint32  age;
        uint8   uuid[16];
    } DTLSMulticastElectionVote;

    enum {
        vote(0), (255)
    } MulticastElectionType;

    struct {
        MulticastElectionType type;
        select (type) {
            case vote:          DTLSMulticastElectionVote;
        } content;
    } DTLSMulticastElection;
```

9. How to use the DTLSMulticast structures

9.1 DTLSMulticastRADIUS

>> The multicast group MAY choose to authenticate clients before they are authorised to become members of the multicast group. If this is

required, the RADIUS protocol may be used with encapsulation in DTLSMulticastRADIUS messages. Once the RADIUS handshake has completed and the client has been authorised to join the group, the client proceeds with a DTLSMulticastJoin message.

>> If the group requires authorisation of clients, the controller MUST respond to all messages apart from DTLSMulticastRADIUS messages with an "unauthorised" DTLSMulticastAckNack message until the client is authorised.

If the group does not require authorisation of clients, the
>> controller MUST consider the client as authorised as soon as the DTLS CLIENT channel is established and the client can immediately send a DTLSMulticastJoin message.

Note that the controller is not required to use RADIUS to authenticate the client. The controller may instead use any information from the client DTLS connection to authenticate the client using some other protocol. Alternatively, the controller may simply allow all clients to join the group.

9.2 DTLSMulticastJoin

The DTLSMulticastJoin is sent by a client to the controller on CLIENT Channels to indicate a client's wish to become a member of the group. The controller must respond with a DTLSMulticastAckNack to indicate if the request was successful.

>> The controller MUST respond to all messages apart from DTLSMulticastRADIUS and DTLSMulticastJoin messages with a "notJoined" DTLSMulticastAckNack message until the client has successfully joined the group.

The parameters in the DTLSMulticastJoin message are described below.

9.2.1 sender_channel

If the sender is sending a join message as a result of receiving a DTLSMulticastReconnect, it should indicate its previously assigned sender channel in this field so that the same channel can be reassigned by the new controller (as the new controller may not have the channel assignment list from the previous controller).

This field should be set to 0 if the listener had no sender channel assigned.

9.2.2 max_subgroups

This is used to indicate the capabilities of the member. Constrained devices may have limited memory and may only be able to support a

small number of subgroups and possibly none at all. Devices with larger memory and processing capabilities may be able to support many sub-groups. A member should try to set this as large as possible so that the controller is not constrained in how it assigns and manages the subgroups.

9.2.3 Group controller election flags NE and EL

A group may or may not support group controller elections. If a group does not support elections, the sysadmin must ensure that an alternative solution is in place to ensure the availability of a group controller. Alternative solutions are beyond the scope of this document.

If any member wishes to take part in group controller elections, it should set the DTLSMulticastJoin.flags.EL otherwise it should set DTLSMulticastJoin.flags.NE.

9.2.4 Member send ability flags TX and RX.

Some members may only be listeners to the group. These members do Not need to be able to send and do not need a SENDER channel assigned. Theyshould set the DTLSMulticastJoin.flags.RX flag.

Some members may wish to take be able to send data to the group. They should set the DTLSMulticastJoin.flags.TX flag.

>> The group controller MUST track the assigned sender channels so that the same sender channel is never simultaneously assigned to multiple entities.

The controller should respond to the "join" with an "ack" or "nack". If the controller requires the client to authenticate before it is allowed to join the group, the controller should return an "unauthorised" error. The client should then begin a RADIUS authentication using DTLSMulticastRADIUS messages (which simply encapsulate standard RADIUS frames). If the RADIUS authentication is successful then the client should re-send its DTLSMulticastJoin message to join the group.

If the client join is successful, the controller will then send one or more DTLSMulticastAddSA messages to provide the appropriate security keys to the client.

9.3 DTLSMulticastAddSA

This message is sent by a controller to provide a security association to one or more members. This allows access to a channel and allows new security associations to be provided for a channel (e.g. during epoch change).

The group security association is sent with the GP flag set. The

same security association is used for the CONTROL channel and all SENDER channels. If the member is only allowed to listen then the RX flag is set. If the member is allowed to send data then the TX flag is set and the assigned SENDER channel is indicated in the "param" field.

The SUBGROUP channel security associations are sent with flags ZZ, RX and SG.

The SUBGROUP channel is indicated in the "channel" field. Only the active controller is allowed to send traffic on a SUBGROUP channel.

The ELECTION channel security association is sent with flags ZZ, EL and TX.

>> The controller SHOULD set the "channel" field to the unix timestamp
>> when the member joined the group. The member MUST store this value and use it as the "age" field in later ***JS

The timeout field is used to indicate the timeout/repeat value in seconds for that channel.

The DTLSMulticastAddSA message can be sent by the controller on a CLIENT channel, in which case it must be acknowledged with an "ack" Or Rejected with a "nack".

The DTLSMulticastAddSA message can also be sent by the controller on A CONTROL or SUBGROUP channel, in which case it is not acknowledged.

9.4 DTLSMulticastDropSA

This message is sent by a controller used to tell one or more listeners that a security association is no longer required. It should not be used as part of the group security because the controller has no guarantee that the listener has complied with the request. It is instead provided to allow the controller to help the listener maintain a minimal memory footprint.

The message can be sent by the controller on a CLIENT channel, in which case it must be acknowledged with an "ack" or rejected with a "nack".

The message can be sent by the controller on a CONTROL or SUBGROUP channel, in which case it is not acknowledged.

9.5 DTLSMulticastReconnect

This message is by a group controller used to tell listeners on the sub-channel to reconnect. This message is only sent by a new group controller after a group controller election has completed and is used to ensure that the listeners establish new CLIENT connections

to the new group controller.

Each time a controller wants clients to reconnect, it generates a new "controller_magic" value. The controller then sends as many reconnect messages as necessary on the CONTROL or SUBGROUP channels until it is confident that all listeners have reconnected. If a listener receives multiple "reconnect" messages (e.g. due to retransmission on a single channel or received on multiple channels), the listener can use the "controller_magic" field to determine if this is a new reconnect request or simply a repeat of an earlier reconnect message.

The message can be sent by the controller on the CONTROL or SUBGROUP channels and is not acknowledged.

When a listener receives a reconnect message with a new "controller_magic" value, it should discard its CLIENT connection and repeat the JOIN process.

The listener does not need to discard any other security association information so can continue to receive (and send) messages on the SENDER channels. This ensures that the multicast group continues to operate normally during the controller election and recovery process.

10. Joining a DTLS multicast group

This takes place after the standard IGMP Multicast JOIN for IPv4 and/or appropriate other actions for other protocols.

>> An entity MUST use the following sequence to join a DTLS multicast group and obtain the multicast key material.

When an entity wishes to JOIN a DTLS multicast group, it generates a new random 16 byte client ID then connects to the group controller on the CLIENT channel as defined above. Note that the CLIENT channel has DTLSMulticast and DTLSMulticastClientPrefix headers before the DTLS traffic.

The DTLS handshake takes place in the usual way and the CLIENT channel is established.

The joining entity has now become a member of the group. It then sends a DTLSMulticastJoin message on the CLIENT channel. The controller will reply with an "ack" or "nack". If the controller responds with a "nack" then the client will not be given the decryption keys for the group's traffic and the controller will close the DTLS connection.

If the controller responded with an "ack" then the controller follows up with one or more "addsa" messages on the CLIENT channel

to allow the member to become a listener (and possibly a sender) on the appropriate channels.

11. Notes on cipher suites

Equivalent cipher suites must be used for all security associations including the CLIENT channels. Clients may need to authenticate to the group controller using different identity keys, but must use equivalent encryption and hash modes.

For example, the following are considered equivalent cipher suites because all use AES-128 in CBC mode with a SHA-256 hash even though they use different identity keys.

```
TLS_RSA_WITH_AES_128_CBC_SHA256          = {0x00,0x3C};
TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256  = {0xC0,0x23};
TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA256   = {0xC0,0x25};
TLS_PSK_WITH_AES_128_CBC_SHA256          = {0x00,0xAE};
```

A cipher suite that used AES-256, a different mode (such as CTR or GCM) or a different hash (such as SHA-1 or SHA-384) would not be considered equivalent.

All security associations apart from the CLIENT channel have explicitly shared security parameters and should indicate the equivalent TLS_PSK_xxx cipher suite.

12. Receiving DTLS multicast data

A listener should apply the following rules in the order below when deciding whether to accept a message:

>> The active group controller MUST listen to all messages on the CLIENT channel so that it can communicate with all the members and handle join requests from new entities.

>> All standby group controllers MUST listen to messages on the CLIENT channel to detect a failure of the active group controller to respond to new join requests.

A listener

>> MUST ignore all messages on the CLIENT channel apart from those identified with its own client ID.

>> MUST ignore all messages on any SUBGROUP channels for which it does not have a security association.

>>- MUST ignore all messages that are not on the last known good epoch for that channel, the "current" epoch or the "next" epoch.

>> MUST ignore CONTROL, SUBGROUP and SENDER messages on the last

known good epoch for that channel and with an earlier or equal sequence number to the last known good sequence number for that channel.

A listener

>> SHOULD accept all remaining messages.

>> SHOULD attempt to decrypt and authenticate all accepted messages.
If the message is valid, the client should update its values for last known good epoch and sequence number for that channel then process the message content.

There is no retry mechanism for DTLS multicast traffic so the listener must take this into account when handling lost or fragmented packets. Fragmentation *is* supported for DTLS multicast but the listener must be able to discard incomplete fragmented packets.

A client should only ever need to store a maximum of two security associations for a channel (remember that all SENDER channels share the same group security association).

These are either:

"last" + "current"

This is the situation after a key rotation where not all senders have switched to the "current" epoch

or

"current" + "next"

This is the situation during a key rotation when no senders have switched to the "next" epoch.

>> A client SHOULD track the following information for all channels that it is listening on:

channel_id	4 bytes
epoch	2 bytes
sequence_number	6 bytes

where "epoch" and "sequence_number" are the epoch and sequence_number from the last good packet received on that channel.

If a key rotation takes place and the client is still tracking senders on the "last" epoch, the client can discard its security association for the "last" epoch. Note that a client may choose to keep the "last" security association for a short period of time (a few seconds) to allow for any delayed packets on the "last" security association to be received and decrypted.

When the listener receives a message on the "next" epoch security association but has no security association for that epoch, it should send a "reqsa" message to the controller to request the

necessary security association data. The controller will respond with an "ack" or "nack". If the controller responds with an "ack", it will then send the listener an "addsa" message with the security association which the listener should "ack" in the usual way. If the controller responded with a "nack" then the the listener is not permitted to access data in the new epoch.

13. Sending DTLS multicast data

>> A sender MUST NOT send messages on a channel unless it has been granted the right to transmit on that channel by the controller in the security association.

>> A sender MUST send messages in the normal DTLS manner with incrementing sequence numbers starting from zero.

When sending encrypted packets, the sender MUST ensure that the packets do not have duplicate IV with any other packets sent with the same security association including packets sent by other devices. Two possible ways to ensure this are:

1) Ensure that the packet's IV is truly random, so a collision is not likely given the size of the IV compared to the maximum number of packets sent on that security association

or

2) Use the channel ID for the top 32 bits of the IV and use any other suitable method to derive the remaining bits. For some cipher modes, this may allow a sequential number assignment (which is lower overhead than random number generation). For other cipher modes, a pseudo-random number with suitable entropy gathering may be sufficient.

>> A sender MUST use the security association of the "current" multicast epoch.

If this would cause the sequence number would wrap, the sender MUST NOT send any more messages on its channel. The sender must instead LEAVE then JOIN the TLS multicast group indicating a sender_channel value of zero so that it obtains a new channel and can safely restart its sequence number from zero.

Note that the group controller should have performed a key rotation on the channel before any sequence counters were due to wrap, so this scenario indicates either a fault in the group controller or a network with very high packet loss.

All changes of epoch are co-ordinated by the group controller as part of the key rotation.

When the sender receives a valid message on the "next" epoch security association, it should:

discard the security association information for the "last" epoch

copy the security association information from the "current" epoch to the "last" epoch

copy the security association information from the "next" epoch to the "current" epoch

clear the security association information for the "next" epoch

set the "next" epoch to the "current" epoch + 1

reset its sender sequence number for the "current" epoch to zero

14. Leaving a DTLS multicast group

>> All members SHOULD indicate their wish to leave a DTLS multicast group

The member sends a DTLSMulticastLeave message to the controller on its CLIENT channel. The controller will acknowledge this message.

The listener has now left the group.

>> When a listener leaves the group, the controller SHOULD perform a key rotation on all channels that the listener had access to so that the listener can no longer send or receive messages on the multicast group.

A controller can consider the listener as having left the group if its CLIENT channel connection is lost (e.g. explicit close or no response to heartbeats from the controller).

15. DTLS multicast group key rotation

The group controller must maintain the integrity and correct operation of the group.
This requires the group controller to:

- Update the necessary security associations if listeners leave the group
- Update a security association before any sender's sequence number wraps

The group controller may also choose to update security associations at other times depending on policy that is outside the scope of this document.

Group key rotation takes place for a security association as follows:

- The controller generates a new random key
- The controller calculates the "next" epoch by adding 1 to the "current" epoch
- The controller sends DTLSMulticastSA messages containing the new security association to the appropriate listeners using the SUBGROUP and/or CLIENT channels
- The controller sends one or more an empty messages on the CONTROL channel using the new security association and a sequence number of zero.

If no multicast messages are lost, all senders will see the empty message on the CONTROL channel and all subsequent messages will be sent using the "next" epoch.

If multicast messages are unreliable, some senders may not see the empty message on the MASTER channel and may continue to send messages on what they think is the "current" epoch. As soon as they see a message on the "next" epoch from one of the other senders, however, they will switch to the "next" epoch.

If a controller sees multiple messages from a sender on what the controller considers to be the "last" epoch (and what the sender
>> considers to be the "current" epoch), the controller MAY repeat one or more empty messages on the CONTROL channel so that the sender sees them and switches to the "next" epoch.

16. Use of CONTROL, SUBGROUP and CLIENT channels for key rotation

For groups with few members, it is feasible to use the CLIENT channels for key rotation as this is a reliable mechanism. The group controller can simply update each listener's security association individually.

The group controller can therefore easily ensure that:

1) The new security association is only delivered to the appropriate listeners

and

2) The listener has received the new security association

This is a simple approach, but does not scale well as the number of members of the group increases. Although the approach is still technically possible, the time required to provide the key to each listener is an O(N) problem and will therefore cause an increasingly large delay between when the key rotation starts and when the key is available for use by the group. This approach also generates an O(N)amount of traffic on the network.

To allow for faster key distribution, the controller may choose to distribute the key using the CONTROL or SUBGROUP channels. This is not a reliable distribution mechanism as these messages are not

acknowledged, but it does take advantage of the multicast nature of the group. The controller may choose to send the same "addsa" message multiple times to reduce the impact of message loss before considering the key rotation complete.

The CONTROL group uses the Group Security Association so all listeners can receive messages on this channel. If a listener "leaves" the group, if it does not discard its group security association information, it can continue to receive and decrypt messages on this channel. If the CONTROL channel is used to provide the new security association, such a listener could decrypt the information for the new security association and therefore continue to decrypt the messages on the channel.

This does not provide "Forward security" so an alternative solution must be available for the controller to rotate keys after a listener has left without forcing the controller to update each remaining listener individually on its CLIENT channel.

When the controller accepts a member into the group, the controller
>> MAY add the member to one or more SUBGROUP channels. Each SUBGROUP Channel has its their own security association so any traffic to a SUBGROUP cannot be decrypted by listeners that are not members of the sub-group.

When a key rotation is required and a listener must be excluded from receiving the new security association, the controller can use the SUBGROUP channels to send the "addsa" messages to the sub-groups that the listener is *not* a member of.

The controller may add listeners to multiple SUBGROUPs (e.g. using a binary tree) so that the number of messages required to update the security association can be significantly reduced.

To see the advantage of the subgroup approach, consider the situation when a member leaves a group:

For a group with 10 remaining listeners, a simple key rotation using the CLIENT channels would require a minimum of {listeners * (addsa + ack)} = 20 messages to distribute the new security association.

For a larger group with 1,000 remaining listeners, the minimum number of messages required is now {listeners * (addsa + ack)}= 2,000 messages to distribute the new security association over the CLIENT channels.

If, however, the controller had organised the listeners into a binary tree and assigned a SUBGROUP channel to each branch, the original 1,000 listeners would require a binary tree with a depth of 10. Each branch on the tree would be assigned a subgroup to allow

multicast communication with all members below that branch.

After a member leaves the group, the controller can update the group security association for all the other members by sending an "addsa" message to the SUBGROUP channels on the branches that do not include the member that has just left, recursing down the branch as necessary to ensure all remaining members have received the message. This would require 9 "addsa" messages to be sent to SUBGROUP channels and one "addsa" + "ack" handshake on the remaining listener, a total of 11 messages.

The controller would also need to perform a key rotation on the subgroups that the departing member was part of so that future messages to that subgroup could not be eavesdropped. The member would have been part of 9 groups so following the same approach, the number of additional messages required would be:

Group branch depth	# members	# messages
1	512	8 + 2
2	256	7 + 2
3	128	6 + 2
4	64	5 + 2
5	32	4 + 2
6	16	3 + 2
7	8	2 + 2
8	4	1 + 2
9	2	0 + 2

Total messages:		54

With a simple binary tree approach, the number of messages required to rotate the keys after a listener leaves is reduced from 2,000 to a minimum of 65. The tree would require 1024 SUBGROUP channels.

With a larger number of branches at each level of the tree, the depth of the tree could be reduced. This would in turn reduce both the number of SUBGROUP channels required and the number of messages required for key rotation after a member leaves the group.

Note that constrained listeners that are not able to join a sufficient number of subgroups may still need to be individually updated by the controller in some circumstances.

If a listener has not received the new security association but
>> receives messages on the new epoch, the listener SHOULD request the information using the "reqsa" message on its CLIENT channel. If the listener is unable to buffer the messages on the new security association whilst waiting for the controller to provide the security association details, the listener will be forced to drop

the packets and data loss will occur.

17. DTLS multicast controller failure detection and election protocol

>> All DTLS multicast groups MUST have a controller. Some groups may allow controller election whereas others may have a fixed controller (which should be highly available if the group is to be reliable).

>> Groups that allow controller election MUST implement the DTLS multicast controller election protocol.

>> A member MUST NOT take part in the election unless it has been provided with the security association for the ELECTION channel and has been granted permission to send on that channel.

>> A member SHOULD NOT attempt to monitor the controller for failure unless it has been allowed to take part in the election.

>> Members allowed to take part in the election SHOULD monitor the controller for failure.

>> A member monitoring the controller MUST consider it to have failed if the following occurs:

- The member has lost its CLIENT channel connection AND the controller has not responded to at least 3 subsequent JOIN messages from the member

OR

- No group key rotation has occurred but the sequence number on any SENDER channel has both MSBs set

OR

- No subgroup key rotation has occurred but the sequence number on a valid packet on a SUBGROUP channel has both MSBs set

>> A member MAY consider the controller to have failed if the following occurs:

- At least 2 JOIN messages have been seen from a 3rd party on the JOIN channel but no controller response has been seen.

>> A member MUST trigger an election if it considers the controller to Have failed.

The election uses the DTLSMulticastElection messages. These are always sent as a DTLSCiphertext content on the ELECTION channel and protected by the election security association.

If a member wishes to trigger an election or join an active election, it first generates a random UUID if it does not have one already. It then sends a DTLSMulticastElectionVote message on the

ELECTION channel with the "current" epoch and sequence number zero, setting the "age" field to the "channel" value from the DTLSMulticastAddSA it received for the ELECTION channel and "uuid" field with its UUID.

>> If the current controller has not failed, it MUST join the election and send a DTLSMulticastElectionVote with age of 0 and its UUID.

If any member of the election sees a DTLSMulticastElectionVote message with a LOWER age than its own, it should withdraw from the election.

If any member of the election sees a DTLSMulticastElectionVote message with the SAME age as its own AND with a lower UUID, it should withdraw from the election.

All members of the election should retransmit their DTLSMulticastElectionVote message after the ELECTION timeout period (as defined in the DTLSMulticastSA message for the ELECTION channel).

A member of the election can consider itself the elected master if it is still a member of the election after two timeout periods have expired with no DTLSMulticastElection messages seen from other members.

The above election process is not intended to be fair. It is instead deliberately designed to prioritise the election as follows:

```
    HIGHEST
        The existing controller (if it is still running)
        The oldest member in the election group
        ...
        The newest member in the election group
    LOWEST
```

If the controller has changed as a result of the election, the new controller must trigger all members of the group to re-connect to it so it can reconstruct the details of the group and hence be able to handle JOIN, LEAVE and key rotation as necessary.

The controller triggers all members to reconnect as follows:

1) It generates a random 32-bit number as the "controller_magic"
2) It sets its next sequence number to 0x8000_0000 for the CONTROL channel.
3) It sends a DTLSMulticastReconnect on the CONTROL channel.
4) It waits for DTLSMulticastSA[CONTROL].timeout seconds.
5) It repeats DTLSMulticastReconnect on the CONTROL channel.
6) It waits for DTLSMulticastSA[CONTROL].timeout seconds.
7) It repeats DTLSMulticastReconnect on the CONTROL channel.

8) It waits for DTLSMulticastSA[CONTROL].timeout seconds.
9) It continues to wait if any clients are still joining the group.
10) It performs a key rotation for the CONTROL channel in the usual way.

All listeners will receive the DTLSMulticastReconnect messages on the CONTROL channel in the usual way. Each listener inspects the "controller_magic" value and if it is new, drops its current MEMBER connection (which is no longer valid) and reconnects to the new group in the usual way.

>> A listener MUST ignore DTLSMulticastReconnect messages if they have the same "controller_magic" value as a previous DTLSMulticastReconnect message and the listener has already reconnected (or is in the process of reconnecting) to the group as a result of the earlier message.

>> The new controller SHOULD attempt to preserve the SENDER channel assigned to each sender when it reconnects to the group unless there is a conflict.

Even if a controller fails, normal application traffic can continue to flow on the SENDER channels while the detection, election, reconnect and key rotation actions are taking place. If these actions all complete before any of the sender sequence numbers would need to wrap, there will be no impact on the flow of application data.

New clients will not be able to connect to the group after a controller has failed until the election process has completed.

18. Acknowledgements

Parts of this document are a byproduct of the "aSSURE" project, partially funded by Innovate UK. It is provided "as is" and without any express or implied warranties,including, without limitation, the implied warranties of fitness for a particular purpose. The views and conclusion contained herein are those of the author and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the aSSURE project or Innovate UK.

19. Security Considerations

DTLS Multicast is all about security. Minor changes, for example adding headers in the frame before the traffic data helps establishes secure channels between 1 to N or M to N end nodes.

20. IANA Considerations

21. References

21.1 Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC4279] Eronen, P., Ed., and H. Tschofenig, Ed., "Pre-Shared Key Ciphersuites for Transport Layer Security (TLS)", RFC 4279, DOI 10.17487/RFC4279, December 2005, <http://www.rfc-editor.org/info/rfc4279>.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <http://www.rfc-editor.org/info/rfc5246>.

[RFC5847] Badra, M., "Pre-Shared Key Cipher Suites for TLS with SHA-256/384 and AES Galois Counter Mode", RFC 5487, DOI 10.17487/RFC5487, March 2009, <http://www.rfc-editor.org/info/rfc5487>.

21.2 Informative References

[RFC3740] Hardjono, T. and B. Weis, "The Multicast Group Security Architecture", RFC 3740, DOI 10.17487/RFC3740, March 2004, <http://www.rfc-editor.org/info/rfc3740>.

[I-D. keoh-dice-multicast-security] Keoh, S., Kumar, S., Garcia-Morchon, O., Dijk, E., and A. Rahman, "DTLS-based Multicast Security in Constrained Environments", expired, draft-keoh-dice-multicast-security-08, July 2014, https://datatracker.ietf.org/doc/draft-keoh-dice-multicast-security/ https://www.ietf.org/archive/id/draft-keoh-dice-multicast-security-08.txt

[I-D. keoh-tls-multicast-security] Keoh, S., Kumar, S., Garcia-Morchon, O., Dijk, E., "DTLS-based Multicast Security for Low-Power and Lossy Networks (LLNs), expired, draft-keoh-tls-multicast-security-00, October 2012, https://tools.ietf.org/html/draft-keoh-tls-multicast-00, https://www.ietf.org/proceedings/85/slides/slides-85-tls-.pdf

Author's Address

Roger Lucas
c/o Cisco International Limited
10, New Square Park
Bedfont Lakes
Feltham
TW14 8HA
United Kingdom

Email: iot@hiddenengine.co.uk