CoRE Working Group L. Seitz
Internet-Draft SICS Swedish ICT
Intended Status: Standards Track G. Selander
Expires: April 24, 2014 Ericsson

October 21, 2013

Additional Security Modes for CoAP
draft-seitz-core-security-modes-00

Abstract

The CoAP draft defines how to use DTLS as security mechanism. In order to establish which nodes are trusted to initiate a DTLS session with a device, the following security modes are defined: NoSec, PreSharedKey, RawPublicKey, and Certificate. These modes require either to provision a list of keys of trusted clients, or to handle heavyweight certificates. This memo proposes two intermediate security modes involving a trusted third party that are very similar to PreSharedKey and RawPublicKey respectively, but which do not require out-of-band provisioning of client keys to the device.

Status of this Memo

Copyright and License Notice

Table of Contents

1. Introduction

The Constrained Application Protocol (CoAP) [I-D.ietf-core-coap] is a light-weight web transfer protocol suitable for applications in embedded devices used in services such as smart energy, smart home, building automation, remote patient monitoring etc. Due to the nature of the these use cases including critical, unattended infrastructure and the personal sphere, security and privacy are critical components.

CoAP message exchanges can be protected with different security protocols. The CoAP specification defines a DTLS [RFC6347] binding for CoAP, which provides communication security including authentication, encryption, integrity, and replay protection. In order to bootstrap trust relations, the CoAP specification defines four security modes that are the result of different provisioning procedures (see section 9 of [I-D.ietf-core-coap]):

- o NoSec
- o PreSharedKey (PSK)
- o RawPublicKey (RPK)
- o Certificate

The NoSec alternative assumes security measures at another protocol layer and provides no security at all. PSK and RPK modes rely on a pre-provisioned list of keys that the device can initiate a DTLS session with. Certificate mode requires provisioning of certain root trust anchor public keys (equivalent to CA certificates) that can be used to validate previously unknown X.509 certificates, before using them to establish a DTLS session.

Given a setting where security is required, and where at least some devices are too resource constrained to handle X.509 certificates, devices would have to use either the PSK or the RPK mode. If the set of nodes that a device would communicate with varies dynamically (e.g. a pay-per-use scenario) this would in turn require constant re-provisioning of lists of trusted clients to the individual devices.

Such an approach will obviously not scale well and make consistent management of security policies over a set of devices very difficult. Therefore we propose two additional security modes that take advantage of the low resource consumption of the PSK and the RPK modes, but also allow to manage dynamic trust relations without having to re-provision the individual nodes. The basic idea is to provision a symmetric key of a trust anchor to the devices. A node wishing to connect to the device can obtain either a derived secret key, or a Message Authentication Code (MAC) of its public key from one of the trust anchors, and the device can verify that this derived secret key, or MAC is generated by a trust anchor. The derived key or public key is then used by the device as in PSK or RPK mode, respectively.

1.1 Terminology

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

Certain security-related terms are to be understood in the sense defined in [RFC4949]. These terms include, but are not limited to, "authentication", "authorization", "confidentiality", "encryption", "data integrity", "message authentication code", and "verify".

Terminology for constrained environments is defined in [I-D.ietf-lwig-terminology] e.g. "constrained device".

Furthermore this memo refers to the following entities:

- o The constrained device providing resources is called the Resource Server (RS)
- o The node connecting to the Resource Server in order to request some resource is called Client (C)
- o The entity having an a-priori trust relation with RS is called the Trust Anchor (TA)

2. DerivedKey Mode

This mode addresses similar use cases as the PSK mode, but without the requirement for out-of-band provisioning of shared keys between C and RS. Instead each resource server is configured with secret, symmetric keys shared with its trust anchors. For simplicity of explanation we assume here, that each RS only has a single TA, and that they share the key K_RS-TA. A client wishing to establish a connection to a RS needs to obtain a symmetric key K_RS-C and a nonce from the TA, where K_RS-C is derived from K_RS-TA and that nonce. C transmits the nonce in the psk_identity field of the ClientKeyExchange message of the DTLS protocol. The RS then derives K_RS-C from the nonce and K_RS-TA, and then both proceed using K_RS-C as a pre-shared key [RFC4279]. Figure 1 illustrates this procedure.

```
         Client              Trust Anchor      Resource Server
```

```
          |   1. Request key      |                    |
          |---------------------->|                    |
          |                       | 2. Process         |
          |                       | authorization      |
          |   3. K_RS-C + nonce   |                    |
          |<----------------------|                    |
          |                       +                    |
          |                                            |
          |   4. DTLS handshake using PSK = K_RS-C     |
          |<------------------------------------------>|
          |      and psk_identity = nonce              |
```

Figure 1: The message flow for DerivedKey mode

How C authenticates with the TA, and how the TA authorizes the request for a key K_RS-C is out of scope for this memo.

2.1 Generating The Nonce

Upon request, the trust anchor verifies if C is authorized to connect to the resource server. How this is done is out of scope for this memo. If the verification succeeds, the TA generates the nonce as follows:

```
nonce = 'DK.' + TA_id + '.' C_id + '.' + sequence_number

where '+' indicates concatenation,
'TA_id' is an identifier that the RS can use to select the
   correct K_RS-TA,
'C_id' is an identifier of C, and
'sequence_number' is a sequence number maintained by the RS and
   the TA.
```

The TA then generates the shared key K_RS-C as described in section 2.2 and transfers the nonce and K_RS-C to C via a secure channel.

2.2 Calculating The Derived Key

K_RS-C is derived from K_RS-TA by the trust anchor and the resource server through a data expansion step, as defined in [RFC5246]:

```
P_hash(secret, seed) = HMAC_hash(secret, A(1) + seed) +
                       HMAC_hash(secret, A(2) + seed) +
                       HMAC_hash(secret, A(3) + seed) + ...

where '+' indicates concatenation and A() is defined as:
```

```
A(0) = seed
A(i) = HMAC_hash(secret, A(i-1))
```

In the present case:

- o hash is SHA-256,
- o 'secret' is the shared key K_RS-TA, and
- o 'seed' is the nonce.

The nonce, associated to the connection request, is generated by the trust anchor (see 2.1). A nonce SHALL NOT be reused with the same shared key K_RS-TA.

With one iteration of the P_SHA256, the output data D of 256 bits can be used to define K_RS-C either as one 256 bit key, or as one 128 bit key using the first 128 bits of D and discarding the rest.

2.3 Generating PSK_IDENTITY

The nonce is used as the psk_identity field of the DTLS ClientKeyExchange message. Upon receiving a psk_identity in the ClientKeyExchange message, an RS can determine by the 'DK' prefix that C wants to use the DerivedKey security mode, and select the corresponding key K_RS-TA by using the nonce in order to calculate the key K_RS-C as specified in 2.2.

2.4 Key Expiration, Anti-replay, And Revocation

The key K_RS-C enables the client to open a DTLS connection to the resource server, but in many cases one does not want this key to be valid forever. Furthermore an attacker can reuse a stolen key to gain access to the RS. Therefore the sequence_number part of the nonce can be used to expire the key K_RS-C (i.e. make it invalid for setting up new DTLS sessions) and protect against reuse of a {key, nonce} pair in a DTLS handshake.

The sequence number is a 32-bit number that is specific to a TA and an RS.

The TA keeps a list of sequence numbers per RS it is responsible for. A RS's sequence number is incremented by 1 for each new shared key K_RS-C generated for this RS.

For each TA an RS has (typically only one), it keeps a window of most recently verified sequence numbers. Sequence number verification SHOULD be performed using the following sliding window procedure, borrowed from Section 3.4.3 of [RFC2402] (see also [RFC6347] section 4.1.2.6).

The sequence number MUST be initialized to zero when an association between a TA and an RS is established. For each received DTLS handshake using the DerivedKey Mode, the RS MUST verify that the nonce contains a fresh sequence number. This SHOULD be the first check applied to a nonce after it has been received in the ClientKeyExchange message, to speed rejection of duplicate or old records.

Freshness is checked through the use of a sliding receive window. (How the window is implemented is a local matter, but the following text describes the functionality that the implementation must exhibit.) A minimum window size of 32 MUST be supported, but a window size of 64 is preferred and SHOULD be employed as the default. Another window size (larger than the minimum) MAY be chosen by the RS.

The "right" edge of the window represents the highest validated Sequence Number value received on this RS. DTLS handshakes, using this security mode, that contain Sequence Numbers lower than the "left" edge of the window are rejected. Handshakes falling within the window are checked against a list of received handshakes with sequence numbers within the window. An efficient means for performing this check, based on the use of a bit mask, is described in Appendix C of [RFC2401].

If the sequence number falls within the window and is new, or if the sequence number is to the right of the window the RS proceeds to generate the shared key K_RS-C. If the handshake succeeds the RS updates the window.

On some occasions one may want to explicitly revoke a key K_RS-C before its expiration. In these cases the trust anchor has to send a message to the RS specifying the sequence number of the key K_RS-C it wants to revoke. The RS can then update the receive window to mark this key as used.

If a server is in use for a long period of time and able to process DTLS handshakes rapidly, the sequence number range may get exhausted within the lifetime of the server. In that case a new shared key K_RS-TA must be provisioned to the server and the TA, and the sequence number counters must be reset.

Note: If we make the very optimistic assumption that a DTLS handshake takes very roughly 1 second for a constrained device, a 32-bit sequence number can last roughly 136 years, before it needs to be reset (60*60*24*365 = max 31,536,000 handshakes per year, 2^32/31,536,000 > 136).

3. AuthorizedPublicKey Mode

This security mode addresses similar use cases as the RPK mode, but without the need for out-of-band validation of public keys. As in the DerivedKey mode, we assume that the resource servers are configured with a symmetric key K_RS-TA for each of their trust anchors. In order to run this mode, the client needs to get its public key authorized for DTLS with the RS by one of the TA. The TA does this by creating an authorization certificate protected by a message authentication code (MAC) using the key K_RS-TA. The TA also provides C with the public key of RS for use in DTLS. The client then performs the DTLS handshake in RPK mode, but replaces the RawPublicKey ClientCertificate with the authorization certificate. The RS verifies the certificate, and if it is valid, proceeds with the DTLS handshake as if the client public key had been provisioned out of band. Furthermore the RS sends an empty certificate_list in the ServerCertificate message, since the key has already been provided to C by the TA.

The authorization certificate is essentially the RawPublicKey certificate of [I-D.ietf-tls-oob-pubkey] with an additional MAC. As with the RPK mode, this security mode benefits from a significantly smaller size of the client’s Certificate message in the DTLS handshake. The verification of a MAC is also less resource consuming than verifying a digital signature. A considerable reduction in message size compared with the RPK mode, is that the RS does not have to send any certificate.

Figure 2 illustrates the message flow of this mode.

```
Client                Trust Anchor      Resource Server
  | 1. Req. authz cert for |                      |
  |----------------------->|                      |
  |  public key PubK_C     | 2. Process           |
  |                        | authorization        |
  |     3. authz cert +    |                      |
  |<-----------------------|                      |
  |   public key PubK_RS   +                      |
  |                                               |
  |    4. DTLS handshake using authz cert         |
  |<--------------------------------------------->|
  |                                               |
```

Figure 2: The message flow for AuthorizedPublicKey mode

How C authenticates with the TA, and how the TA authorizes the request for an authorization certificate out of scope for this memo.

3.1 Structure of the Authorization Certificate Extension

This section outlines the changes to the DTLS handshake message contents for the AuthorizedPublicKey mode. The procedure is analogous to the one in [I-D.ietf-tls-oob-pubkey], using the new certificate_type 'AuthzCert' and the new structure 'MacCert'.

```
struct {
   select(certificate_type) {
      // certificate type defined in this document
      case AuthzCert:
         MacCert certificate;

      // certificate type defined in [I-D.ietf-tls-oob-pubkey]
      case RawPublicKey:
         opaque ASN.1_subjectPublicKeyInfo<1..2^24-1>;

      // X.509 certificate defined in RFC 5246
      case X.509:
         ASN.1Cert certificate_list<0..2^24-1>;

      // Additional certificate type based on TLS
      // Certificate Type Registry
   };
} Certificate;
```

The MacCert structure is defined as follows:

```
struct {
  opaque ASN.1_subjectPublicKeyInfo<1..2^24-1>
  opaque trust_anchor_id;
  uint32 sequence_number;
  MACAlgorithm mac_algorithm;
  uint8 mac_length;
  opaque MAC[mac_length];
} MacCert;
```

Where ASN.1_subjectPublicKeyInfo is defined in section 3 of [I-D.ietf-tls-oob-pubkey], and the MACAlgorithm type is defined in [RFC5246]. The 'mac_algorithm' parameter specifies a function MAC = M(key, message), where K_RS-TA is used as the key, and the certificate with an empty MAC value as the message.

Note that the size of the MacCert structure is only marginally larger than the RawPublicKey certificate used in RPK mode.

The extended client_hello and extended server_hello defined in section 3 of [I-D.ietf-tls-oob-pubkey] are also used here, with the new certificate_type 'AuthzCert'.

3.2 Client and Server Handshake Behavior

Section 4 of [I-D.ietf-tls-oob-pubkey] shows the use of the client and server certificate types in TLS. The AuthorizedPublicKey mode uses a variant of the handshake exemplified in section 5.3 of [I-D.ietf-tls-oob-pubkey] as illustrated by figure 4.

```
client_hello,
client_certificate_type=(AuthzCert) //(1)
                ->
                <- server_hello,
                   server_certificate_type=(X.509) //(2)
                   certificate, //(3)
                   client_certificate_type=(AuthzCert) //(4)
                   certificate_request, //(5)
                   server_key_exchange,
                   server_hello_done
certificate, // (6)
client_key_exchange,
change_cipher_spec,
finished        ->
                <- change_cipher_spec,
                   finished

Application Data <-------> Application Data
```

Figure 4: Example of a DTLS handshake with Authorization Certificate provided by the Client

This handshake starts with the client indicating its ability to use AuthorizedPublicKey mode (1). Since the client has already received the server's public key from the TA, the server sends an empty certificate_list in the certificate message (3), using the indication for X.509 certificates in (2). This indication is only used, because it allows to send an empty certificate_list. For client authentication the server indicates in (4) that it selected the AuthorizedPublicKey mode and requests a certificate from the client in (5). The client provides a MacCert structure (6) after receiving and processing the server hello message.

3.3 Payload Verification Procedure

After negotiating client_certificate_type="AuthzCert" in the

ClientHello/ServerHello steps of the DTLS protocol, and receiving the ClientCertificate message, the RS proceeds to verify the C’s public key using the following steps:

- o Check if the trust_anchor_id identifies a trust anchor
- o Check if the sequence_number is valid
- o Check that the ASN.1_subjectPublicKeyInfo contains a valid SubjectPublicKeyInfo structure
- o Check the mac with the shared key K_RS-TA.

If any of these checks fail, the DTLS handshake is aborted and the RS MUST send a bad_certificate alert.

3.4 Key Expiration, Anti-replay, And Revocation

The rationale and procedures for handling sequence numbers are the same as described in section 2.4.

4. Access Control Lists

The CoAP specification uses Access Control Lists to keep track of pre-shared symmetric keys, raw public keys, and root trust anchors for X.509 certificates, used in the corresponding security modes (see section 9 and especially 9.1.3.2.1 of [I-D.ietf-core-coap]). An implementation supporting one or both of the security modes specified above MUST be extended to support storing lists of identifiers and secret keys of the trust anchors.

5. Security Considerations

All security consideration from [RFC6347] and [RFC4279] also apply to this approach. Furthermore the trust anchors used for authorizing the use of keys in the two proposed security modes are valuable targets for attacks since they potentially allow access to many devices. They should be protected accordingly.

The nonce used to generate the shared key for the DK mode is static except for the sequence number, an attacker could exploit this for a dictionary attack. If such an attack is considered feasible, an additional random seed should be added to the nonce to increase the variable part. The client identifier and other information in the nonce cannot be trusted until the client is authenticated using the key derived from the nonce.

The sequence number mechanism for expiration can potentially lead to keys being valid for a longer time than expected. This will be the case if the number of requests for a device drops significantly, and it therefore takes longer to fill the sliding window. Trust anchors can monitor this and explicitly revoke keys if the frequency of requests drops significantly. It is also possible to use timers in the device to implement complementing expiry mechanisms.

An attacker can induce a server to perform the DTLS handshake up to Flight 4, without having any legitimate key material from a trust anchor. This could be used for denial of service attacks against the server. However these problems are also present with any of the standard CoAP security modes and respective DTLS handshakes.

6. IANA Considerations

IANA is asked to register a new value in the "TLS Certificate Types" registry of Transport Layer Security (TLS) Extensions [TLS-Certificate-Types-Registry], as follows:

```
   Value: 3   Description: Authorized Public Key Certificate
(AuthzCert)   Reference: [[THIS RFC]]
```

7. Acknowledgements

The authors would like to thank Stefanie Gerdes, Mats Naeslund, and John Mattsson for contributions and helpful comments.

8. References

8.1 Normative References

[I-D.ietf-core-coap]
Shelby, Z., Hartke, K., Bormann, C., and B. Frank, "Constrained Application Protocol (CoAP)", draft-ietf-core-coap-18 (work in progress), June 2013.

[I-D.ietf-tls-oob-pubkey]
Wouters, P., Tschofenig, H., Gilmore, J., Weiler, S., and T. Kivinen, "Out-of-Band Public Key Validation for Transport Layer Security (TLS)", draft-ietf-tls-oob-pubkey-10 (work in progress), October 2013.

[TLS-Certificate-Types-Registry]
"TLS Certificate Types Registry", February 2013, <http://www.iana.org/assignments/tls-extensiontype-values#tls-extensiontype-values-2>.

[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, December 2005.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.

[RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, January 2012.

8.2 Informative References

[I-D.ietf-lwig-terminology]
Bormann, C., Ersue, M., and Keranen, A., "Terminology for Constrained Node Networks", draft-ietf-lwig-terminology-05 (work in progress), July 2013.

[RFC2401] Kent, S. and R. Atkinson, "Security Architecture for the Internet Protocol", RFC 2401, November 1998. Obsoleted by RFC4301.

[RFC2402] Kent, S. and R. Atkinson, "IP Authentication Header", RFC 2402, November 1998. Obsoleted by RFC4302, RFC4305.

[RFC4949] Shirey, R., "Internet Security Glossary, Version 2", FYI 36, RFC 4949, August 2007.

Authors' Addresses

Ludwig Seitz
SICS Swedish ICT AB
Scheelevagen 17
22370 Lund
SWEDEN
EMail: ludwig@sics.se

Goeran Selander
Ericsson
Farogatan 6
16480 Kista
SWEDEN
EMail: goran.selander@ericsson.com