Network Working Group
Internet-Draft
Intended status: Informational
Expires: January 27, 2012

J. Arkko
A. Keranen
Ericsson
July 26, 2011

CoAP Security Architecture
draft-arkko-core-security-arch-00

Abstract

Constrained Application Protocol (CoAP) is a light-weight protocol designed to be used in machine-to-machine applications. This memo describes challenges associated with securing CoAP and proposes a new security model that the authors believe is suitable for these environments. The model requires minimal amount of configuration, but still provides strong security and is a natural fit with the typical communication practices smart object networking environments. This memo also proposes JSON payload format extensions to support the architecture.

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on January 27, 2012.

Copyright Notice

Copyright (c) 2011 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect

to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction

Constrained Application Protocol (CoAP) [I-D.ietf-core-coap] is a light-weight protocol designed to be used in machine-to-machine applications such as smart energy and building automation.

This memo describes implementation and operational challenges associated with securing CoAP in these environments (Section 3), reviews related work in solving these challenges (Section 2), and proposes a security model (Section 4) that the authors believe is suitable for many machine-to-machine application environments. The model requires minimal amount of configuration, but still provides strong security and is a natural fit with the typical communication practices smart object networking environments. Finally, this memo proposes some protocol and payload format extensions to support the architecture (Section 5). Section 6 provides a summary of the approach.

2. Related Work

CoAP base specification [I-D.ietf-core-coap] outlines how to use DTLS [RFC5238] and IPsec [RFC4306] for securing the protocol. DTLS can be applied with group keys, pairwise shared keys, or with certificates. The security model in all cases is mutual authentication, so while there is some commonality to HTTP in verifying the server identity, in practice the models are quite different. The specification says little about how DTLS keys are managed.

The IPsec mode is described with regards to the protocol requirements, noting that small implementations of IKEv2 exist [I-D.kivinen-ipsecme-ikev2-minimal]. However, the specification is silent on policy and other aspects that are normally necessary in order to implement interoperable use of IPsec in any environment [RFC5406].

[I-D.garcia-core-security] discusses the overall security problem for Internet of Things devices. It also discusses various solutions, including IKEv2/IPsec [RFC4306], TLS/SSL [RFC5246], DTLS [RFC5238], HIP [RFC5201] [I-D.ietf-hip-rfc5201-bis] [I-D.moskowitz-hip-rg-dex], PANA [RFC5191], and EAP [RFC3748]. The draft also discusses various operational scenarios, bootstrapping mechanisms, and challenges associated with implementing secity mechanisms in these environments.

[I-D.iab-smart-object-workshop] gives an overview of the security discussions at the March 2011 IAB workshop on smart objects. The workshop recommended that additional work is needed in developing suitable credential management mechanisms (perhaps something similar

to the Bluetooth pairing mechanism), understanding the implementability of standard security mechanisms in small devices (see, for instance, [I-D.kivinen-ipsecme-ikev2-minimal]), and additional research in the area of lightweight cryptographic primitives.

[I-D.sarikaya-core-sbootstrapping] discusses the bootstrapping problem with low-powered nodes, and argues that this problem should be solved at a general level and not left to link layer specific mechanisms. The draft looks at EAP [RFC3748], PANA [RFC5191], HIP Diet Exchange (HIP-DEX) [I-D.moskowitz-hip-rg-dex], and 802.1X [IEEE.802-1X.2010] as potential solutions for bootstrapping.

[I-D.moskowitz-hip-rg-dex] defines a light-weight version of the HIP protocol for low-power nodes. This version uses a fixed set of algorithms, elliptic curve cryptography, and eliminates hash functions. The protocol still operates based on host identities, and runs end-to-end between hosts, protecting IP layer communications. [RFC6078] describes an extension of HIP that can be used to send upper layer protocol messages without running the usual HIP base exchange at all.

[I-D.daniel-6lowpan-security-analysis] makes a comprehensive analysis of security issues related to 6LOWPAN networks, but its findings also apply more generally for all low-powered networks. Some of the issues this document discusses include the need to minimize the number of transmitted bits and simplify implementations, threats in the smart object networking environments, and the suitability of 6LOWPAN security mechanisms, IPsec, and key management protocols for implementation in these environments.

Cryptographically Generated Addresses (CGAs) [RFC3972] and Host Identity Protocol (HIP) [RFC5201] have employed similar ideas as those proposed in this memo, though with slightly different purpose in mind, and at a different protocol layer. Similarly, PGP [RFC4880] and other similar tools have popularized the concept of exchanging key fingerprint values off-line. This is very similar to what is proposed in this memo.

[I-D.rescorla-jsms], [I-D.jones-json-web-signature], and [I-D.jones-json-web-token] propose JSON extensions similar to those discussed in this memo, though constructed for other purposes. Further work is needed to analyze if these proposals could be used as a basis for smart object security communication security as well. Obviously, general-purpose JSON signature mechanisms should be used if they exist, even if some additional data elements might have to be defined to carry all the information that this memo requires.

3. Challenges

This section discusses three challenges: implementation difficulties, practical provisioning problems, and layering and communication models.

The most often discussed issues in the security for the Internet of Things relates to implementation difficulties. The desire to build small, battery-operated, and inexpensive devices drives the creation of devices with a limited protocol and application suite. Some of the typical limitations include running CoAP instead of HTTP, limited support for security mechanisms, limited processing power for long key lengths, sleep schedule that does not allow communication at all times, and so on. In addition, the devices typically have very limited support for configuration, making it hard to set up secrets and trust anchors.

The implementation difficulties are important, but they should not be overemphasized. It is important to select the right security mechanisms and avoid duplicated or unnecessary functionality. But at the end of the day, if strong cryptographic security is needed, the implementations have to support that. Also, the use of the most lightweight algorithms and cryptographic primitives is useful, but should not be the only consideration in the design. Interoperability is also important, and often other parts of the system, such as key management protocols or certificate formats are heavier to implement than the algorithms themselves.

The second challenge relates to practical provisioning problems. These are perhaps the most fundamental and difficult issue, and unfortunately often neglected in the design. There are several problems in the provisioning and management of smart object networks:

- Small devices have no natural user interface for configuration that would be required for the installation of shared secrets and other security-related parameters. Typically, there is no keyboard, no display, and there may not even be buttons to press. Some devices may only have one interface, the interface to the network.

- Manual configuration is rarely, if at all, possible, as the necessary skills are missing in typical installation environments (such as in family homes).

- There may be a large number of devices. Configuration tasks that may be acceptable when performed for one device may become unacceptable with dozens or hundreds of devices.

- Network configurations evolve over the lifetime of the devices, as additional devices are introduced or addresses change. Various central nodes may also receive more frequent updates than individual devices such as sensors embedded in building materials.

Finally, layering and communication models present difficulties for straightforward use of the most obvious security mechanisms. Smart object networks typically pass information through multiple participating nodes [I-D.arkko-core-sleepy-sensors] and end-to-end security for IP or transport layers may not fit such communication models very well. The primary reasons for needing middleboxes relates to the need to accommodate for sleeping nodes as well to enable the implementation of nodes that store or aggregate information.

4. Proposed Architecture

The proposed security architecture describes both a deployment model for provisioning as well as a technical model for networks and protocols.

The basis of the architecture are self-generated secure identities, similar to Cryptographically Generated Addresses (CGAs) [RFC3972] or Host Identity Tags (HITs) [RFC5201]. That is, we assume the following holds:

$$I = h(P|O)$$

where I is the secure identity of the device, h is a hash function, P is the public key from a key pair generated by the device, and O is optional other information.

4.1. Provisioning

As provisioning security credentials, shared secrets, and policy information is difficult, the provisioning model is based only on the secure identities. A typical network installation involves physical placement of a number of devices while noting the identities of these devices. This list of short identifiers can then be fed to a central server as a list of authorized devices. Secure communications can then commence with the devices, at least as far as information from from the devices to the server is concerned, which is what is needed for sensor networks. Actuator networks and server-to-device communication is covered in Section 4.4.

Where necessary, the information collected at installation time may also include other parameters relevant to the application, such as

the location or purpose of the devices. This would enable the server to know, for instance, that a particular device is the temperature sensor for the kitchen.

Collecting the identity information at installation time can be arranged in a number of ways. The authors have employed a simple but not completely secure method where the last few digits of the identity are printed on a tiny device just a few millimeters across. Alternatively, the packaging for the device may include the full identity (typically 32 hex digits), retrieved from the device at manufacturing time. This identity can be read, for instance, by a bar code reader carried by the installation personnel. (Note that the identities are not secret, the security of the system is not dependent on the identity information leaking to others. The real owner of an identity can always prove its ownership with the private key which never leaves the device.) Finally, the device may use its wired network interface or proximity-based communications, such as Near-Field Communications (NFC) or Radio-Frequency Identity tags (RFIDs). Such interfaces allow secure communication of the device identity to an information gathering device at installation time.

No matter what the method of information collection is, this provisioning model minimizes the effort required to set up the security. Each devices generates its own identity in a random, secure key generation process. The identities are self-securing in the sense that if you know the identity of the peer you want to communicate with, messages from the peer can be signed by the peer's private key and it is trivial to verify that the message came from the expected peer. There is no need to configure an identity and certificate of that identity separately. There is no need to configure a group secret or a shared secret. There is no need to configure a trust anchor. In addition, the identities are typically collected anyway for application purposes (such as identifying which sensor is in which room). Under most circumstances there is actually no additional configuration effort from provisioning security.

4.2. Device Groups

In some deployment cases it is also possible to configure the identity of an entire group of devices, rather than registering the individual devices. For instance, many installations employ a kit of devices bought from the same manufacturer in one package. It is easy to provide an identity for such a set of devices as follows:

Idev = h(Pdev|Potherdev1|Potherdev2|...|Potherdevn)

Igrp = h(Pdev1|Pdev2|...|Pdevm)

where Idev is the identity of an individual device, Pdev is the public key of that device, and Potherdevi are the public keys of other devices in the group. Now, we can define the secure identity of the group (Igrp) as a hash of all the public keys of the devices in the group (Pdevi).

The installation personnel can scan the identity of the group from the box that the kit came in, and this identity can be stored in a server that is expected to receive information from the nodes. Later when the individual devices contact this server, they will be able to show that they are part of the group, as they can reveal their own public key and the public keys of the other devices. Devices that do not belong to the kit can not claim to be in the group, because the group identity would change if any new keys were added to Igrp.

4.3. Protocol Architecture

As noted above, the starting point of the architecture is that nodes self-generate secure identities which are then communicated out-of-band to the peers that need to know what devices to trust. To support this model in a protocol architecture, we also need to use these secure identities to implement secure messaging between the peers, explain how the system can respond to different types of attacks such as replay attempts, and decide at what protocol layer and endpoints the architecture should use.

Securing the messages is straightforward. A node with identity I should sign each message it sends with the private key associated with the identity I. This allows the recipient to verify that the message was constructed by the sender. This is similar to what Secure Neighbor Discovery (SEND) does with its RSA Signature Option [RFC3971].

However, this simple model needs some enhancements to be able to withstand denial-of-service and replay attacks. As we expect connectivity in smart object networks to be intermittent, traditional active methods such as nonce exchanges are not suitable. Instead, an optional timestamp-based approach SHOULD be used in addition to the basic signatures. This approach is similar to the one used to secure unsolicited SEND messages. Nodes that implement the timestamp approach need to have a real-time clock or they need to synchronize to one using a network time protocol [RFC5905]. Additionally, nodes that have persistent memory, SHOULD implement a monotonically increasing sequence number. Message recipients SHOULD silently ignore messages when they see a timestamp value that is out of range from the current time plus or minus a small time drift factor. Similarly, recipients that have seen multiple messages from the same sender SHOULD silently ignore messages that do not have a sequence

number greater than the one they have seen last.

These exchanges are basic cryptographic protocol tools, and have been used in different layers of the IP protocol stack for different purposes. For instance, HIP in its opportunistic mode could be used to implement largely the same functionality at the IP layer. However, it is our belief that the right layer for this solution is at the application layer. More specifically, in the data formats transported in the payload part of CoAP. This approach provides the following benefits:

- Ability for intermediaries to act as caches to support different sleep schedules, without the security model being impacted.

- Ability for intermediaries to be built to perform aggregation, filtering, storage and other actions, again without impacting the security of the data being transmitted or stored.

- Ability to operate in the presence of traditional middleboxes, such as a protocol translators or even NATs (not that we recommend their use in these environments).

Note that there is no requirement that the secure identities be associated with IP addresses. They can certainly be used as input material for constructing addresses for stateless address autoconfiguration [RFC4862], but this is not required.

4.4. Actuator Networking

The above architecture is a perfect fit for sensor networks where information flows from large number of devices to small number of servers. But it is not sufficient alone for other types of applications. For instance, in actuator applications a large number of devices need to take commands from somewhere else. In such applications it is necessary to secure that the commands come from an authorized source.

This can be supported, with some additional provisioning effort and optional pairing protocols. The basic provisioning approach is as described in Section 4.1, but in addition there must be something that informs the devices of the identity of the trusted server(s). There are multiple ways to provide this information. One simple approach is to feed the identities of the trusted server(s) to devices at installation time. This requires either a separate user interface, local connection (such as USB), or using the network interface of the device for configuration. In any case, as with sensor networks the amount of configuration information is minimized: just one short identity value needs to be fed in. Not both an

identity and a certificate. Not shared secrets that must be kept confidential. An even simpler provisioning approach is that the devices in the device group discussed in Section 4.2 trust each other. Then no configuration is needed at installation time.

When both peers know the expected cryptographic identity of the other peer off-line, secure communications can commence.

Alternatively, various pairing schemes can be employed. Note that these schemes can benefit from the already secure identifiers on the device side. For instance, the server can send a pairing message to each device after their initial power-on and before they have been paired with anyone, encrypted with the public key of the device. As with all pairing schemes that do not employ a shared secret or the secure identity of both parties, there are some remaining vulnerabilities that may or may not be acceptable for the application in question.

In any case, the secure identities help again in ensuring that the operations are as simple as possible. Only identities need to be communicated to the devices, not certificates, not shared secrets or IPsec policy rules.

5. Proposed Protocol Extensions

The concrete implementation of the proposed architecture involves a specification for the identity format and generation, and a specification of the data format necessary to carry the signature, public key, timestamp, and sequence number data objects.

The data format part of this specification could be implemented in various ways, as S/MIME data [RFC3851], XML signatures [RFC3275], or as additional data in JSON [I-D.jennings-senml] [RFC4627]. We have chosen to use the JSON format in this memo.

5.1. Identity Format

The format of identifiers in binary representation is 128-bit identifiers. These identifiers have no association with any existing number space managed by IANA. In particular, they are not part of the IPv6 address space; they exist at application layer.

The identifiers can be represented in textual form as Universal Resource Names (URNs), with the format "device:cgi-HEX" where "device" is the designated new URN type, "cgi" is a subtype that stands for cryptographically generated identifiers, and HEX is an exactly 32 characters long string of hex digits.

While not at the right layer from the point of view of our architecture, these identities could also be used in the Authority Name part of CoAP DTLS (Section 10 of [I-D.ietf-core-coap]), IKE or other lower-level protocols.

5.2. Identity Generation

The process of generating a new identity takes two input values: the public key of the identity owner as a DER-encoded ASN.1 structure of the type SubjectPublicKeyInfo, and optional other parameters.

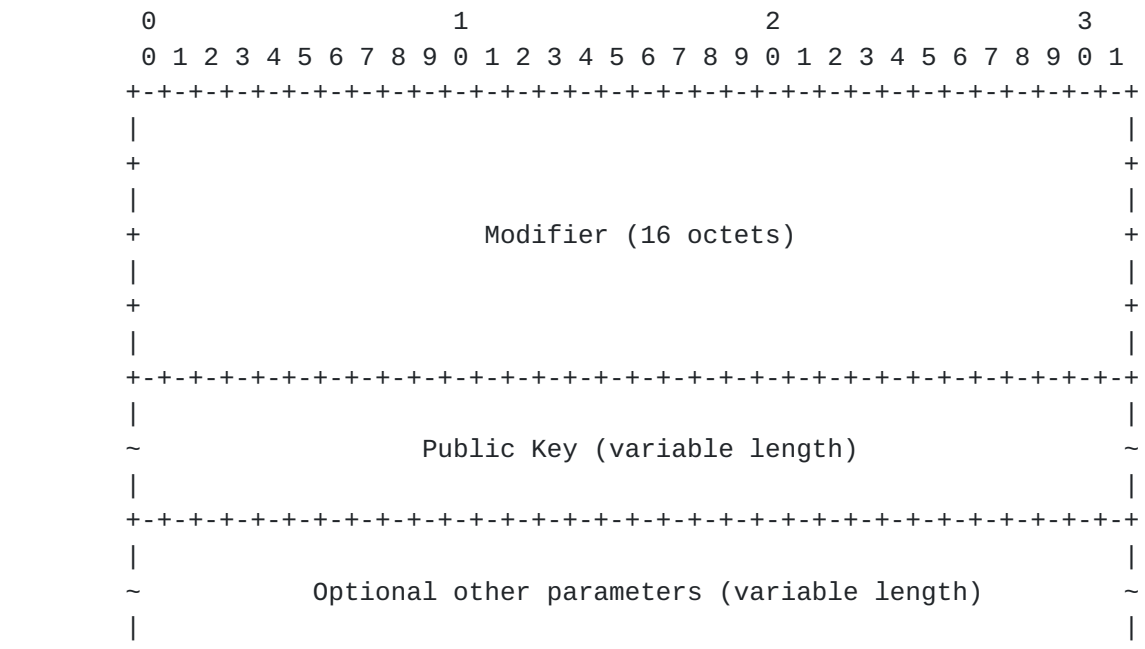
An identity and associated Identity Parameters Block (defined further below) SHOULD be generated as follows:

1. Generate a modifier, a random or pseudo-random 128-bit value.

2. Concatenate from left to right the modifier value, the encoded public key, and any optional other parameters. Execute the SHA-256 algorithm [FIPS.180-3.2008] on the concatenation. Take the 128 leftmost bits of the SHA-256 hash value. The result is the identity.

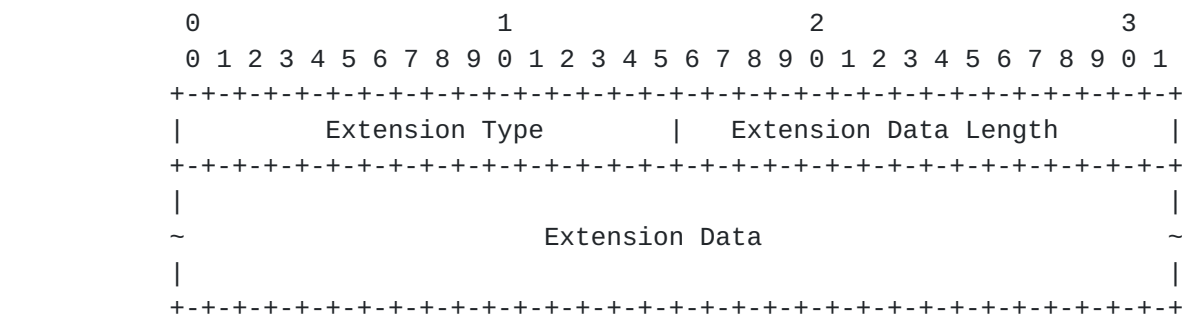
3. Form an Identity Parameters Block data structure by concatenating from left to right the modifier value, the encoded public key, and any optional other parameters.

The output of the address generation algorithm is a new identity and a new Identity Parameters Block data structure. The latter data structure has the following format:



The Public Key field MUST be formatted as a DER-encoded [CCITT.X690.2002] ASN.1 structure of the type SubjectPublicKeyInfo, defined in the Internet X.509 certificate profile [RFC3280]. RSA public/private key pair SHOULD be used. When RSA is used, the algorithm identifier MUST be rsaEncryption, which is 1.2.840.113549.1.1.1, and the RSA public key MUST be formatted by using the RSAPublicKey type as specified in Section 2.3.1 of RFC 3279 [RFC3279].

The other parameters is a sequence of extension blocks with the following format:



Where

Extension Type

16-bit identifier of the type of the Extension Field. Identifier for the one currently defined extension is defined in Section 5.2.1, and some reserved values and values for testing use are given in Section 8. The summary of the defined values is as follows:

Value	Name
0x0000	Reserved (Section 8)
0x0001	Identifier_Group (Section 5.2.1)
0xFFFD	Exp_FFFD (Section 8)
0xFFFE	Exp_FFFE (Section 8)
0xFFFF	Exp_FFFF (Section 8)

Extension Data Length

16-bit unsigned integer. Length of the Extension Data field of this option, in octets.

Extension Data

Variable-length field. Extension-Type-specific data.

5.2.1. Identifier Groups

This extension has the Extension Type 0x0001 (Identifier_Group). The purpose of the extension is to carry the public keys of other devices in a group of devices. As discussed in Section 4.2, this can be used to show membership of a group and ease the provisioning process.

The extension data should consist of a 16-bit length field that expresses the number of public keys that follow, followed by each public key, encoded as described in Section 5.2.

5.3. JSON Identity

Messages that employ secure identities and carry JSON [RFC4627] payloads need to carry information about the identity of the device that ultimately provided the payload. This information is necessary to understand the source of the information, and is also necessary to verify a cryptographic signature attached to the payload. However, the mechanisms for transporting information about the identity and making a signature are kept separate.

An identity is represented by a two-field object in JSON, for instance:

```
{ "id": "device:cgi-27611bc81020716627ff0000cfaa1234",
  "ipb":  "4e26b808cd05d4e26b80912ae3e26b809143fe4e26b4GFTR35f8266" }
```

The "id" field MUST be included, and an additional "ipb" field for the Identity Parameters Block MAY be included. To save communications bandwidth, the optional field MAY be omitted even when the sender has the information. However, the "ipb" field SHOULD appear frequently enough in messages that recipients have likely cached it.

5.3.1. The id Field

This field MUST contain an identity string in the format defined in Section 5.1.

5.3.2. The ipb Field

This field MUST contain the BASE64-encoded Identity Parameters Block associated with the same identity as given in the "id" field.

5.4. JSON Signature Envelope

Messages that employ secure identities and carry JSON [RFC4627] payloads need to carry enough information to prove that the message came from the right source. The JSON Signature Envelope is a JSON object that carries a signature. Together with the JSON identity fields it becomes possible for the recipients to verify the signature. This object can be used to implement secure communication for devices that have the secure identifiers described above and that use JSON to transport information. Other signature envelope formats are needed for other payload formats, but the authors believe that the JSON format is widely applicable to smart objects.

Note that multiple competing ways to represent signature envelopes in JSON are under development [I-D.rescorla-jsms], [I-D.jones-json-web-signature], and [I-D.jones-json-web-token]. The exact choice of encoding remains to be determined; this memo provides its own signature envelope format only for completeness.

Every secure message MUST carry a JSON envelope object. This object MUST have exactly one "jmsg" field for the actual payload, "jid" field for the identity, and "jsig" field for the signature. The fields MUST also appear in this order. The messages MAY carry an additional "jts" field for the timestamp, and "jsq" field for the sequence number. If these fields are included, they MUST appear after the mandatory fields and in the given order.

For instance, the following example contains a JSON signature envelope and a JSON payload from a temperature sensor:

```
 { "jmsg": { "temp": 27.5 },
   "jid":  { "id": "device:cgi-27611bc81020716627ff0000cfaa1234",
             "ipb":  "4e26b808cd05d4e26b912ae3e26b809143fe4eb4GFTR35f82" },
   "jts":  { "s": 1311176727, "f": 123987 },
   "jsq":  23,
   "jsig": "18929abqxc67juil7ff231000912927755bRRwlkadbfddceab"}
```

Note that signatures envelopes can be nested; a JSON signature envelope can be placed inside another signature envelope in the "jmsg" field and signed. This is useful to implement secure intermediaries that want to include additional information beyond what the device itself provided.

5.4.1. The jmsg Field

This field MUST contain the actual payload that the device wants to send, in the usual JSON format.

Note that the JSON envelope needs to be useful without securing information in the rest of the CoAP message carrying it, as well as in situations where it is retransmitted in CoAP or HTTP via an intermediary. For this reason all the relevant information MUST be in the payload part. This is usually the case when taking an information centric approach as in [I-D.arkko-core-sleepy-sensors]. The jid field carries the identity of the device, and the jmsg carries all relevant information about what the devices wants to communicate. Consequently, the payload SHOULD be self-contained, without reference to the source or destination IP addresses of the CoAP message, or to the CoAP/HTTP method or URI.

5.4.2. The jid Field

This field MUST contain an identity as defined in Section 5.3.

5.4.3. The jts Field

This field MUST contain an object with two fields. The first field, "s", indicates the number of seconds since January 1, 1970, 00:00 UTC. At least 48 bits of accuracy is required. The second field, "f" indicate the number of 1/64K fractions of a second, with 16 bits of accuracy.

Implementation note: This format is compatible with the usual representation of time under UNIX, although the number of bits available for the integer and fraction parts may vary.

5.4.4. The jsq Field

This field MUST contain an integer representing a monotonically increasing sequence number of all messages sent by the sender. At least 32 bits of accuracy are required.

5.4.5. The jsig Field

This field MUST contain a variable-length string containing a BASE64-encoded PKCS#1 v1.5 signature, constructed by using the sender's private key over the following sequence of octets:

1. The 128-bit CGI Usage Discriminator value for this specification, 0x53eb e540 4a92 5517 57b6 e398 7aaf a085. (The value has been generated randomly by the editor of this specification.)

2. The entire JSON payload, verbatim and in text as carried in the message, with the contents of the jsig field set to an empty string (jsig: "").

The signature value is computed with the RSASSA-PKCS1-v1_5 algorithm and SHA-256 hash, as defined in [PKCS.1.1993]. Senders use their private key associated with the claimed identity. The "jsig" field MUST be the last one in JSON payload. The resulting PKCS#1 v1.5 signature is put in the "jsig" field.

Receivers MUST treat messages without the "jsig" field as unsecured. A received "jsig" field MUST be checked as follows:

- The receiver MUST ignore any fields that come after the first "jsig" field, for both verification and other processing purposes.
- There must be an associated JSON identity information, so that both the identity and associated public key must be apparent from the secured message, or learned from a preceding message.
- The "jsig" field MUST have correct encoding.
- The signature verification MUST show that the signature has been calculated as specified above.

Messages that do not pass all the above tests MUST be silently discarded if the host has been configured to accept only secured CoAP messages. The messages MAY be accepted if the host has been configured to accept both secured and unsecured messages but MUST be treated as an unsecured message. The receiver MAY also otherwise silently discard packets (e.g., as a response to an apparent CPU exhausting DoS attack).

6. Concluding Remarks

This memo has presented a deployment model, security architecture, and an initial sketch of protocol design to support the architecture. To recap, the main benefits of this model are

- Minimal configuration: per device or per group registration of identities in a server, but no configuration in every device.
- Support for deployment models that are easily implementable by installation personnel. The necessary practices are already employed in typical current smart object networks, even when there is particular support for security.
- Architecture that naturally supports information-centric networking, multicast, middleboxes, aggregation, sleeping nodes, and other aspects that are typical for networking for smart objects.

7. Security Considerations

This entire memo deals with security issues. Some analysis of the security of the mechanisms proposed in this memo is necessary, however.

The security of the architecture rests on the choice of the number of bits in the identifier and the used hash and signature algorithm. With the use of 128 bits identifiers and SHA-256 and RSA, it is expected that the security level is similar to the one in HIP, and goes beyond the 59 bit security of CGAs.

The basic architecture concerns itself only with integrity and data origin verification, not about confidentiality. Where confidentiality or identity privacy is required, additional mechanisms are needed.

Replay attacks can be prevented beyond a small time window of acceptable clock drift, when devices employ the optional timestamp mechanism. This rests on the assumption of secure time synchronization or configuration in the nodes, however. Where NTP is used, its security properties in different modes are discussed in Section 15 of [RFC5905]. In general, no major security problems have been experienced with NTP protocol or reference implementation [NTP.Wikipedia], but protection against determined hostile attackers does require authentication at NTP the layer. Alternative, simpler approaches include relying on the accuracy of clocks set at manufacturing time.

The optional sequence number mechanism can prevent all replay attacks for persistent communications between two peers. Without the use of these two mechanisms there is no support for preventing replay attacks. This may be acceptable in some environments, but not in all.

Any information centric communication model is resistant to attacks against nodes only sending information, as they are not expected to process any security-related messages. Thus, the "sleep torture deprivation attack" described by Stajano and Anderson in [Resurrecting-Duckling] and other denial-of-service attacks of the same nature are not applicable in the architecture proposed in this memo. However, by the same token nodes that receive information become more vulnerable to denial-of-service attacks, as nonce exchanges, puzzles and other standard protocol mechanisms are not used to guard against the receiver having to verify a cryptographic operation on a received packet. The authors believe that this is the right tradeoff for sensor networking, given that server and gateway implementations are more likely to have the necessary capabilities to

deal with attacks than sensor nodes.

8. IANA Considerations

IANA should reserve the new URN type "device" (Section 5.1). A new registry should be created to hold subtypes of this URN type, with the initial value "cgi" defined in this memo. New values can be created through IETF Review or IESG Approval [RFC5226].

IANA should also create a new registry for Cryptographically Generated Identifiers, and add a new name space Extension Type (Section 5.2) there. Policy for adding new extensions in this registry is RFC Required or IESG Approval [RFC5226]. Initial values for the Extension Type field are given below. Assignments consist of a name and the value.

Extension Type 0x0000 should be marked as reserved. Section 5.2.1 allocates Extension Type 0x0001. As recommended in [RFC3692], this document also makes the following assignments for experimental and testing use: the value 0xFFFD, with name Exp_FFFD; the value 0xFFFE, with name Exp_FFFE, and the value 0xFFFF, with name Exp_FFFF.

IANA should also add another new name space to the same registry, for 128-bit CGI Usage Discriminators. These values are allocated on a First Come, First Served basis [RFC5226]. The one initial value in the registry is given in Section 5.4.5.

9. References

9.1. Normative References

[I-D.ietf-core-coap]
Shelby, Z., Hartke, K., Bormann, C., and B. Frank, "Constrained Application Protocol (CoAP)", draft-ietf-core-coap-06 (work in progress), May 2011.

[I-D.jennings-senml]
Jennings, C., "Media Type for Sensor Markup Language (SENML)", draft-jennings-senml-05 (work in progress), March 2011.

[RFC3279] Bassham, L., Polk, W., and R. Housley, "Algorithms and Identifiers for the Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 3279, April 2002.

[RFC3280] Housley, R., Polk, W., Ford, W., and D. Solo, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 3280, April 2002.

[RFC4627] Crockford, D., "The application/json Media Type for JavaScript Object Notation (JSON)", RFC 4627, July 2006.

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, May 2008.

[RFC5905] Mills, D., Martin, J., Burbank, J., and W. Kasch, "Network Time Protocol Version 4: Protocol and Algorithms Specification", RFC 5905, June 2010.

[PKCS.1.1993] RSA Laboratories, "RSA Encryption Standard, Version 1.5", PKCS 1, November 1993.

[FIPS.180-3.2008] National Institute of Standards and Technology, "Secure Hash Standard", FIPS PUB 180-3, October 2008, <http://csrc.nist.gov/publications/fips/fips180-3/fips180-3.pdf>.

[CCITT.X690.2002] International International Telephone and Telegraph Consultative Committee, "ASN.1 encoding rules: Specification of basic encoding Rules (BER), Canonical encoding rules (CER) and Distinguished encoding rules (DER)", CCITT Recommendation X.690, July 2002.

9.2. Informative References

[RFC3275] Eastlake, D., Reagle, J., and D. Solo, "(Extensible Markup Language) XML-Signature Syntax and Processing", RFC 3275, March 2002.

[RFC3692] Narten, T., "Assigning Experimental and Testing Numbers Considered Useful", BCP 82, RFC 3692, January 2004.

[RFC3748] Aboba, B., Blunk, L., Vollbrecht, J., Carlson, J., and H. Levkowetz, "Extensible Authentication Protocol (EAP)", RFC 3748, June 2004.

[RFC3851] Ramsdell, B., "Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.1 Message Specification", RFC 3851, July 2004.

[RFC3971] Arkko, J., Kempf, J., Zill, B., and P. Nikander, "SEcure Neighbor Discovery (SEND)", RFC 3971, March 2005.

[RFC3972] Aura, T., "Cryptographically Generated Addresses (CGA)", RFC 3972, March 2005.

[RFC4306] Kaufman, C., "Internet Key Exchange (IKEv2) Protocol", RFC 4306, December 2005.

[RFC4862] Thomson, S., Narten, T., and T. Jinmei, "IPv6 Stateless Address Autoconfiguration", RFC 4862, September 2007.

[RFC4880] Callas, J., Donnerhacke, L., Finney, H., Shaw, D., and R. Thayer, "OpenPGP Message Format", RFC 4880, November 2007.

[RFC5191] Forsberg, D., Ohba, Y., Patil, B., Tschofenig, H., and A. Yegin, "Protocol for Carrying Authentication for Network Access (PANA)", RFC 5191, May 2008.

[RFC5201] Moskowitz, R., Nikander, P., Jokela, P., and T. Henderson, "Host Identity Protocol", RFC 5201, April 2008.

[RFC5238] Phelan, T., "Datagram Transport Layer Security (DTLS) over the Datagram Congestion Control Protocol (DCCP)", RFC 5238, May 2008.

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

[RFC5406] Bellovin, S., "Guidelines for Specifying the Use of IPsec Version 2", BCP 146, RFC 5406, February 2009.

[RFC6078] Camarillo, G. and J. Melen, "Host Identity Protocol (HIP) Immediate Carriage and Conveyance of Upper-Layer Protocol Signaling (HICCUPS)", RFC 6078, January 2011.

[I-D.arkko-core-sleepy-sensors]
Arkko, J., Rissanen, H., Loreto, S., Turanyi, Z., and O. Novo, "Implementing Tiny COAP Sensors", draft-arkko-core-sleepy-sensors-01 (work in progress), July 2011.

[I-D.daniel-6lowpan-security-analysis]
Park, S., Kim, K., Haddad, W., Chakrabarti, S., and J. Laganier, "IPv6 over Low Power WPAN Security Analysis", draft-daniel-6lowpan-security-analysis-05 (work in progress), March 2011.

[I-D.garcia-core-security]
Garcia-Morchon, O., Keoh, S., Kumar, S., Hummen, R., and R. Struik, "Security Considerations in the IP-based Internet of Things", draft-garcia-core-security-02 (work in progress), July 2011.

[I-D.iab-smart-object-workshop]
Tschofenig, H. and J. Arkko, "Report from the 'Interconnecting Smart Objects with the Internet' Workshop, 25th March 2011, Prague", draft-iab-smart-object-workshop-01 (work in progress), July 2011.

[I-D.ietf-hip-rfc5201-bis]
Moskowitz, R., Heer, T., Jokela, P., and T. Henderson, "Host Identity Protocol Version 2 (HIPv2)", draft-ietf-hip-rfc5201-bis-06 (work in progress), July 2011.

[I-D.jones-json-web-signature]
Jones, M., Balfanz, D., Bradley, J., Goland, Y., Panzer, J., Sakimura, N., and P. Tarjan, "JSON Web Signature (JWS)", draft-jones-json-web-signature-02 (work in progress), April 2011.

[I-D.jones-json-web-token]
Jones, M., Balfanz, D., Bradley, J., Goland, Y., Panzer, J., Sakimura, N., and P. Tarjan, "JSON Web Token (JWT)", draft-jones-json-web-token-05 (work in progress), July 2011.

[I-D.kivinen-ipsecme-ikev2-minimal]
Kivinen, T., "Minimal IKEv2", draft-kivinen-ipsecme-ikev2-minimal-00 (work in progress), February 2011.

[I-D.moskowitz-hip-rg-dex]
Moskowitz, R., "HIP Diet EXchange (DEX)", draft-moskowitz-hip-rg-dex-05 (work in progress), March 2011.

[I-D.rescorla-jsms]
Rescorla, E. and J. Hildebrand, "JavaScript Message Security Format", draft-rescorla-jsms-00 (work in progress), March 2011.

[I-D.sarikaya-core-sbootstrapping]
Sarikaya, B., Ohba, Y., Moskowitz, R., Cao, Z., and R.

Cragie, "Security Bootstrapping of Resource-Constrained Devices", draft-sarikaya-core-sbootstrapping-02 (work in progress), June 2011.

[IEEE.802-1X.2010]
Institute of Electrical and Electronics Engineers, "IEEE 802.1X Port-Based Network Access Control", IEEE IEEE Standard 802.1X, February 2010.

[Resurrecting-Duckling]
Stajano, F. and R. Anderson, "The Resurrecting Duckling: Security Issues for Ubiquitous Computing", IEEE Computer Journal Volume 42, Issue 5, 2002.

[NTP.Wikipedia]
Wikipedia, "Network Time Protocol", Wikipedia article , July 2011, <http://en.wikipedia.org/wiki/Network_Time_Protocol>.

Appendix A. Acknowledgments

The authors would like to thank to Oscar Novo, Heidi-Maria Rissanen, Samita Chakrabarti, and Fredrik Garneij for interesting discussions in this problem space.

Authors' Addresses

Jari Arkko
Ericsson
Jorvas 02420
Finland

Email: jari.arkko@piuha.net

Ari Keranen
Ericsson
Jorvas 02420
Finland

Email: ari.keranen@ericsson.com