Thing-to-Thing Research Group
Internet-Draft
Intended status: Informational
Expires: August 16, 2017
K. Hartke
Universitaet Bremen TZI
February 12, 2017

CoRE Application Descriptions
draft-hartke-core-apps-07

Abstract

The interfaces of RESTful, hypermedia-driven Web applications consist of reusable components such as Internet media types and link relation types. This document proposes CoRE Application Descriptions, a convention for application designers to describe the programmable interfaces of their applications in a structured way so that other parties can easily develop interoperable clients and servers or reuse the components in their own applications.

Note to Readers

This Internet-Draft should be discussed on the Thing-to-Thing Research Group (T2TRG) mailing list <t2trg@irtf.org> <https://www.irtf.org/mailman/listinfo/t2trg>.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

Representational State Transfer (REST) [16] is an architectural style for distributed hypermedia systems. Over the years, REST has gained popularity not only as an approach for large-scale information dissemination, but also as the basic principle for designing and building Internet-based applications in general.

In the coming years, the size and scope of the Internet is expected to greatly increase as physical-world objects become smart enough to communicate over the Internet -- a phenomenon known as the Internet of Things (IoT). As things learn to speak the languages of the net, the idea of applying REST principles to the design of IoT application architectures suggests itself. To this end, the Constrained Application Protocol (CoAP) [24] was created, an application-layer protocol that enables RESTful applications in constrained-node networks [10], thus giving rise to a new setting for Internet-based applications: the Constrained RESTful Environment (CoRE).

To realize the full benefits and advantages of the REST style, a set of constraints needs to be maintained when designing new applications and their application programming interfaces (APIs). One of the fundamental principles is that "REST APIs must be hypertext-driven" [17]. This principle is often ignored by application designers, who instead specify their APIs out-of-band in terms of fixed URI patterns, e.g., in the API documentation or in a machine-readable format that facilitates code generation. Although this approach may appear easy for clients to use, the fixed resource names and data formats lead to a tight coupling between client and server implementations and make the system less flexible. Violations of REST design principles like this result in APIs that may not be as scalable, extensible, and interoperable as promised by REST [20].

REST is intended for long-lived network-based applications that span multiple organizations [17]. Principled REST APIs require some design effort, since application designers do not only have to take current requirements into consideration, but also have to anticipate changes that may be required in the future -- years or even decades after the application has been deployed for the first time. The reward is long-term stability and evolvability, both of which are very desirable features in the Internet of Things.

To aid application designers in the design process, this document proposes CoRE Application Descriptions, a convention for describing the APIs of RESTful, hypermedia-driven Web applications. CoRE Application Descriptions help application designers avoid common mistakes by focusing almost all of the descriptive effort on defining the Internet media type(s) that are used for representing resources and driving application state.

A template provides a consistent format for the description of APIs so that implementers can easily build interoperable clients and servers and other application designers can reuse application components in their own applications.

2. Application Descriptions

A CoRE Application Description is a named set of reusable components. It describes a contract between a server hosting an instance of the described application and clients that wish to interface with that instance.

A CoRE Application Description is comprised of:

- URI schemes that identify communication protocols,
- Internet media types that identify representation formats,
- link relation types that identify link semantics,
- form relation types that identify form semantics,
- variable names that identify the semantics of variables in templated links,
- form field names that identify the semantics of form fields in forms, and
- optionally, well-known locations.

Together, these components provide the specific, in-band instructions for interfacing with a given application.

2.1. URI Schemes

The foundation of a hypermedia-driven REST API are the communication protocol(s) spoken between a client and a server. Although HTTP/1.1 [14] is by far the most common communication protocol for REST APIs, a REST API should typically not be dependent on any specific communication protocol.

The use of a particular protocol by a client is guided by URI schemes [7]. URI schemes specify the syntax and semantics of URI references [1] that the server includes in links (Section 2.2.2) and forms (Section 2.2.3).

A URI scheme refers to a family of protocols, typically distinguished by a version number. For example, the "http" URI scheme refers to the two members of the HTTP family of protocols: HTTP/1.1 [14] and HTTP/2 [8] (as well as some predecessors). The specific HTTP version used is negotiated between a client and a server in-band using the version indicator in the HTTP request-line or the TLS Application-Layer Protocol Negotiation (ALPN) extension [18].

IANA maintains a list of registered URI schemes at
<http://www.iana.org/assignments/uri-schemes>.

2.2. Internet Media Types

One of the most important aspect of hypermedia-driven communications is the concept of Internet media types [2]. Media types are used to label representations so that it is known how the representation should be interpreted and how it is encoded. The centerpiece of a CoRE Application Description should be one or more media types.

A media type identifies a versioned series of representation formats (Section 2.2.1): a media type does not identify a particular version of a representation format; rather, the media type identifies the family, and includes provisions for version indicator(s) embedded in the representations themselves to determine more precisely the nature of how the data is to be interpreted [21]. A new media type is only needed to designate a completely incompatible format [21].

Note: The terms media type and representation format are often used interchangeably. In this document, the term "media type" refers specifically to a string of characters such as "application/xml" that is used to label representations; the term "representation format" refers to the definition of the syntax and semantics of representations, such as XML 1.0 [12] or XML 1.1 [13].

Media types consist of a top-level type and a subtype, structured into trees [2]. Optionally, media types can have parameters. For example, the media type "text/plain; charset=utf-8" is a subtype for plain text under the "text" top-level type in the standards tree and has a parameter "charset" with the value "utf-8".

Media types can be further refined by

- structured type name suffixes (e.g., "+xml" appended to the base subtype name; see Section 4.2.8 of RFC 6838 [2]),
- a "profile" parameter (see Section 3.1 of RFC 6906 [25]),
- subtype information embedded in the representations themselves (e.g., "xmlns" declarations in XML documents [11]),

or a similar annotation. An annotation directly in the media type is generally preferable, since subtype information embedded in representations can typically not be negotiated during content negotiation (e.g., using the CoAP Accept option).

In CoAP, media types are combined with content coding information [15] to indicate the "content format" [24] of a representation. Each content format is assigned a numeric identifier that can be used instead of the (typically much longer) textual name of the media type in representation formats with space constraints. The flat number space loses the structural information that the textual names have, however.

The media type of a representation must be determined from in-band information (e.g., from the CoAP Content-Format option). Clients must not assume a structure from the application context or other out-of-band information.

> IANA maintains a list of registered Internet media types at <http://www.iana.org/assignments/media-types>.

> IANA maintains a list of registered structured suffixes at <http://www.iana.org/assignments/media-type-structured-suffix>.

> IANA maintains a list of registered CoAP content formats at <http://www.iana.org/assignments/core-parameters>.

2.2.1. Representation Formats

In RESTful applications, clients and servers exchange representations that capture the current or intended state of a resource and that are labeled with a media type. A representation is a sequence of bytes whose structure and semantics are specified by a representation format: a set of rules for encoding information.

Representation formats should generally allow clients with different goals, so they can do different things with the same data. The specification of a representation format "describes a problem space, not a prescribed relationship between client and server. Client and server must share an understanding of the representations they're passing back and forth, but they don't need to have the same idea of what the problem is that needs to be solved." [22]

Representation formats and their specifications frequently evolve over time. It is part of the responsibility of the designer of a new version to insure both forward and backward compatibility: new representations should work reasonably (with some fallback) with old processors and old representations should work reasonably with new processors [21].

Representation formats enable hypermedia-driven applications when they support the expression of hypermedia controls, i.e., links (Section 2.2.2) and forms (Section 2.2.3).

2.2.2. Links

As described in RFC 5988 [5], a link is a typed connection between two resources. Additionally, a link is the primary means for a client to navigate from one resource to another.

A link is comprised of:

- o a link context (usually the "current" resource),
- o a link relation type that identifies the semantics of the link (see Section 2.3),
- o a link target, identified by a URI, and
- o optionally, target attributes that further describe the link or the link target.

A link can be viewed as a statement of the form "{link context} has a {link relation type} resource at {link target}, which has {target attributes}" [5]. For example, the resource <http://example.com/> could have a "terms-of-service" resource at <http://example.com/tos>, which has a representation with the media type "text/html".

There are two special kinds of links:

- o An embedding link is a link with an additional hint: when the link is processed, it should be substituted with the representation of the referenced resource rather than cause the client to navigate away from the current resource. Thus, traversing an embedding link adds to the current state rather than replacing it.

  The most well known example for an embedding link is the HTML <img> element. When a Web browser processes this element, it automatically dereferences the "src" and renders the resulting image in place of the <img> element.

- o A templated link is a link where the client constructs the link target URI from provided in-band instructions. The specific rules for such instructions are described by the representation format. URI Templates [3] provide a generic way to construct URIs through variable expansion.

  Templated links allow a client to construct resource URIs without being coupled to the resource structure at the server, provided that the client learns the template from a representation sent by the server and does not have the template hard-coded.

2.2.3. Forms

A form is the primary means for a client to submit information to a server, typically in order to change resource state.

A form is comprised of:

- a form context (usually the "current" resource),

- a form relation type that identifies the semantics of the form (see Section 2.4),

- a form target, identified by a URI,

- a submission method (PUT, POST, PATCH, FETCH, or DELETE),

- a description of a representation that the server expects as part of the form submission, and

- optionally, target attributes that further describe the form or the form target.

A form can be viewed as an instruction of the form "To {form relation type} the {form context}, make a {method} request to {form target}, which has {target attributes}". For example, to "update" the resource <http://example.com/config>, a client could be required to make a PUT request to <http://example.com/config>. (In many cases, the target of a form is the same resource as the context, but this is not required.)

The description of the expected representation can be a set of form fields (see Section 2.4.1) or simply a list of acceptable media types.

Note: A form with a submission method of GET is, strictly speaking, a templated link, since it provides a way to construct a URI and does not submit a representation to the server.

2.3. Link Relation Types

A link relation type identifies the semantics of a link [5]. For example, a link with the relation type "copyright" indicates that the resource identified by the target URI is a statement of the copyright terms applying to the link context.

Relation types are not to be confused with media types; they do not identify the format of the representation that results when the link is dereferenced [5]. Rather, they only describe how the link context is related to another resource [5].

IANA maintains a list of registered link relation types at <http://www.iana.org/assignments/link-relations>.

Applications that don't wish to register a link relation type can use an extension link relation type [5], which is a URI that uniquely identifies the link relation type. For example, an application can use the string "http://example.com/foo" as link relation type without having to register it. Using a URI to identify an extension link relation type, rather than a simple string, reduces the probability of different link relation types using the same identifiers.

In order to minimize the overhead of link relation types in representation formats with space constraints, IANA-registered link relation types are assigned a numeric identifier that can be used in place of the textual name (see also Section 6.2). For example, the link relation type "copyright" has the number 12. A representation format may additionally provide numeric identifiers for extension link relation types.

2.3.1. Template Variable Names

A templated link enables clients to construct the target URI of a link, for example, when the link refers to a space of resources rather than a single resource. The most prominent mechanisms for this are URI Templates [3] and the HTML <form> element with a submission method of GET.

To enable an automated client to construct an URI reference from a URI Template, the name of the variable in the template can be used to identify the semantics of the variable. For example, when retrieving the representation of a collection of temperature readings, a variable named "threshold" could indicate the variable for setting a threshold of the readings to retrieve.

Template variable names are scoped to link relation types, i.e., two variables with the same name can have different semantics if they appear in links with different link relation types.

2.4. Form Relation Types

A form relation type identifies the semantics of a form. For example, a form with the form relation type "create-item" indicates that a new item can be created within the form context by making a request to the resource identified by the target URI.

IANA maintains a list of registered form relation types at <TBD>.

Similar to extension link relation types, applications can use extension form relation types when they don't wish to register a form relation type.

IANA-registered form relation types are assigned a numeric identifier that can be used in place of the textual name. For example, the form relation type "update" has the number 3. A representation format may additionally provide numeric identifiers for extension form relation types.

2.4.1. Form Field Names

Forms can have a detailed description of the representation expected by the server as part of form submission. This description typically consists of a set of form fields where each form field is comprised of a field name, a field type, and optionally a number of attributes such as a default value, a validation rule or a human-readable label.

To enable an automated client to fill out a form, the field name can be used to identify the semantics of the form field. For example, when controlling a smart light bulb, the field name "brightness" could indicate the field for setting the desired brightness of the light bulb.

Field names are scoped to form relation types, i.e., two form fields with the same name can have different semantics if they appear in forms with different form relation types.

The type of a form field is a data type such as "an integer between 1 and 100" or "a RGB color". The type is orthogonal to the field name, i.e., the type should not be determined from the field name even though the client can identify the semantics of the field from the name. This separation makes it easy to change the set of acceptable values in the future.

2.5. Well-Known Locations

Some applications may require the discovery of information about a host, known as "site-wide metadata" in RFC 5785 [4]. For example, RFC 6415 [19] defines a metadata document format for describing a host; similarly, RFC 6690 [23] defines a link format for the discovery of resources hosted by a server.

Applications that need to define a resource for this kind of metadata can register new "well-known locations". RFC 5785 [4] defines the path prefix "/.well-known/" in "http" and "https" URIs for this purpose. RFC 7252 [24] extends this convention to "coap" and "coaps" URIs.

IANA maintains a list of registered well-known URIs at <http://www.iana.org/assignments/well-known-uris>.

3. Application Description Template

As applications are implemented and deployed, it becomes important to describe them in some structured way. This section provides a simple template for CoRE Application Descriptions. A uniform structure allows implementers to easily determine the components that make up the interface of an application.

The template below lists all components of applications that both the client and the server implementation of the application need to understand in order to interoperate. Crucially, items not listed in the template are not part of the contract between clients and servers -- they are implementation details. This includes in particular the URIs of resources (see Section 4).

CoRE Application Descriptions are intended to be published in human-readable format by designers of applications and by operators of deployed application instances. Application designers may publish an application description as a general specification of all application instances, so that implementers can create interoperable clients and servers. Operators of application instances may publish an application description as part of the API documentation of the service, which should also include instructions how the service can be located and which communication protocols and security modes are used.

The fields of the template are as follows:

Application name:
Name of the application. The name is not used to negotiate capabilities; it is purely informational. A name may include a version number or, for example, refer to a living standard that is updated continuously.

URI schemes:
URI schemes identifying the communication protocols that need to be understood by clients and servers. This information is mostly relevant for deploymed instances of the application rather than for the general specification of the application.

Media types:

Internet media types that identify the representation formats that need to be understood by clients and servers. An application description must comprise at least one media type. Additional media types may be required or optional.

Link relation types:
Link relation types that identify the semantics of links. An application description may comprise IANA-registered link relation types and extension link relation types. Both may be required or optional.

Template variable names:
For each link relation type, variable names that identify the semantics of variables in templated links with that link relation type. Whether a template variable is required or optional is indicated in-band inside the templated link.

Form relation types:
Form relation types that identify the semantics of forms and, for each form relation type, the submission method(s) to be used. An application description may comprise IANA-registered form relation types and extension form relation types. Both may be required or optional.

Form field names:
For each form relation type, form field names that identify the semantics of form fields in forms with that form relation type. Whether a form field is required or optional is indicated in-band inside the form.

Well-known locations:
Well-known locations in the resource identifier space of servers that clients can use to discover information given the DNS name or IP address of a server.

Interoperability considerations:
Any issues regarding the interoperable use of the components of the application should be given here.

Security considerations:
Security considerations for the security of the application must be specified here.

Contact:
Person (including contact information) to contact for further information.

Author/Change controller:

Person (including contact information) authorized to change this application description.

Each field should include full citations for all specifications necessary to understand the application components.

4. URI Design Considerations

URIs [1] are a cornerstone of RESTful applications. They enable uniform identification of resources via URI schemes [7] and are used every time a client interacts with a particular resource or when a resource representation references another resource.

URIs often include structured application data in the path and query components, such as paths in a filesystem or keys in a database. It is common for many RESTful applications to use these structures not only as an implementation detail but also make them part of the public REST API, prescribing a fixed format for this data. However, there are a number of problems with this practice [6], in particular if the application designer and the server owner are not the same entity.

In hypermedia-driven applications, URIs are therefore not included in the application interface. A CoRE Application Description must not mandate any particular form of URI substructure.

RFC 7320 [6] describes the problematic practice of fixed URI structures in detail and provides some acceptable alternatives.

Nevertheless, the design of the URI structure on a server is an essential part of implementing a RESTful application, even though it is not part of the application interface. The server implementer is responsible for binding the resources identified by the application designer to URIs.

A good RESTful URI is:

- Short. Short URIs are easier to remember and cause less overhead in requests and representations.

- Meaningful. A URI should describe the resource in a way that is meaningful and useful to humans.

- Consistent. URIs should follow a consistent pattern to make it easy to reason about the application.

- Bookmarkable. Cool URIs don't change [9]. However, in practice, application resource structures do change. That should cause URIs to change as well so they better reflect reality. Implementations should not depend on unchanging URIs.

- Shareable. A URI should not be context sensitive, e.g., to the currently logged-in user. It should be possible to share a URI with third parties so they can access the same resource.

- Extension-less. Some applications return different data for different extensions, e.g., for "contacts.xml" or "contacts.json". But different URIs imply different resources. RESTful URIs should identify a single resource. Different representations of the resource can be negotiated (e.g., using the CoAP Accept option).

5. Security Considerations

The security considerations of RFC 3986 [1], RFC 5785 [4], RFC 5988 [5], RFC 6570 [3], RFC 6838 [2], RFC 7320 [6], and RFC 7595 [7] are inherited.

All components of an application description are expected to contain clear security considerations. CoRE Application Descriptions should furthermore contain security considerations that need to be taken into account for the security of the overall application.

6. IANA Considerations

[Note to RFC Editor: Please replace XXXX in this section with the RFC number of this specification.]

6.1. Content-Format Registry

RFC 6838 [2] establishes a IANA registry for media types. Many of these media types are also useful in constrained environments as CoAP content formats. RFC 7252 [24] establishes a IANA registry for these content formats. This specification tasks IANA with the allocation of a content format for any existing or new media type registration that does not define any parameters (required or optional). The content formats shall be allocated in the range 1000-9999.

6.2. Link Relation Type Registry

RFC 5988 [5] establishes a IANA registry for link relation types. This specification extends the registration template with a "Relation ID": a numeric identifier that can be used instead of the "Relation Name" to identify a link relation type. IANA is tasked with the assignment of an ID to any existing or new link relation type. The IDs shall be assigned in the range 1-9999.

6.3. Form Relation Type Registry

This specification establishes a IANA registry for form relation types.

6.3.1. Registering New Form Relation Types

Form relation types are registered in the same way as link relation types [5], i.e., they are registered on the advice of a Designated Expert with a Specification Required.

The requirements for registered relation types are adopted from Section 4.1 of RFC 5988 [5].

The registration template is:

- Relation Name:
- Relation ID:
- Description:
- Reference:
- Notes: [optional]

The IDs shall be assigned in the range 1-9999.

6.3.2. Initial Registry Contents

The Form Relation Type registry's initial contents are:

- Relation Name: create-item
  Relation ID: 1
  Description: Refers to a resource that can be used to create a resource in a collection of resources.
  Reference: [RFCXXXX]

- Relation Name: delete
  Relation ID: 2
  Description: Refers to a resource that can be used to delete a resource in a collection of resources.
  Reference: [RFCXXXX]

- Relation Name: update
  Relation ID: 3
  Description: Refers to a resource that can be used to update the state of the form context.

Reference: [RFCXXXX]

- Relation Name: search
  Relation ID: 4
  Description: Refers to a resource that can be used to search the form context.
  Reference: [RFCXXXX]

7. References

7.1. Normative References

[1] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <http://www.rfc-editor.org/info/rfc3986>.

[2] Freed, N., Klensin, J., and T. Hansen, "Media Type Specifications and Registration Procedures", BCP 13, RFC 6838, DOI 10.17487/RFC6838, January 2013, <http://www.rfc-editor.org/info/rfc6838>.

[3] Gregorio, J., Fielding, R., Hadley, M., Nottingham, M., and D. Orchard, "URI Template", RFC 6570, DOI 10.17487/RFC6570, March 2012, <http://www.rfc-editor.org/info/rfc6570>.

[4] Nottingham, M. and E. Hammer-Lahav, "Defining Well-Known Uniform Resource Identifiers (URIs)", RFC 5785, DOI 10.17487/RFC5785, April 2010, <http://www.rfc-editor.org/info/rfc5785>.

[5] Nottingham, M., "Web Linking", RFC 5988, DOI 10.17487/RFC5988, October 2010, <http://www.rfc-editor.org/info/rfc5988>.

[6] Nottingham, M., "URI Design and Ownership", BCP 190, RFC 7320, DOI 10.17487/RFC7320, July 2014, <http://www.rfc-editor.org/info/rfc7320>.

[7] Thaler, D., Ed., Hansen, T., and T. Hardie, "Guidelines and Registration Procedures for URI Schemes", BCP 35, RFC 7595, DOI 10.17487/RFC7595, June 2015, <http://www.rfc-editor.org/info/rfc7595>.

7.2. Informative References

[8] Belshe, M., Peon, R., and M. Thomson, Ed., "Hypertext Transfer Protocol Version 2 (HTTP/2)", RFC 7540, DOI 10.17487/RFC7540, May 2015, <http://www.rfc-editor.org/info/rfc7540>.

[9] Berners-Lee, T., "Cool URIs don't change", 1998, <http://www.w3.org/Provider/Style/URI.html>.

[10] Bormann, C., Ersue, M., and A. Keranen, "Terminology for Constrained-Node Networks", RFC 7228, DOI 10.17487/RFC7228, May 2014, <http://www.rfc-editor.org/info/rfc7228>.

[11] Bray, T., Hollander, D., Layman, A., Tobin, R., and H. Thompson, "Namespaces in XML 1.0 (Third Edition)", World Wide Web Consortium Recommendation REC-xml-names-20091208, December 2009, <http://www.w3.org/TR/2009/REC-xml-names-20091208>.

[12] Bray, T., Paoli, J., Sperberg-McQueen, M., Maler, E., and F. Yergeau, "Extensible Markup Language (XML) 1.0 (Fifth Edition)", World Wide Web Consortium Recommendation REC-xml-20081126, November 2008, <http://www.w3.org/TR/2008/REC-xml-20081126>.

[13] Bray, T., Paoli, J., Sperberg-McQueen, M., Maler, E., Yergeau, F., and J. Cowan, "Extensible Markup Language (XML) 1.1 (Second Edition)", World Wide Web Consortium Recommendation REC-xml11-20060816, August 2006, <http://www.w3.org/TR/2006/REC-xml11-20060816>.

[14] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <http://www.rfc-editor.org/info/rfc7230>.

[15] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", RFC 7231, DOI 10.17487/RFC7231, June 2014, <http://www.rfc-editor.org/info/rfc7231>.

[16] Fielding, R., "Architectural Styles and the Design of Network-based Software Architectures", Ph.D. Dissertation, University of California, Irvine, 2000, <http://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf>.

[17] Fielding, R., "REST APIs must be hypertext-driven", October 2008, <http://roy.gbiv.com/untangled/2008/rest-apis-must-be-hypertext-driven>.

[18] Friedl, S., Popov, A., Langley, A., and E. Stephan, "Transport Layer Security (TLS) Application-Layer Protocol Negotiation Extension", RFC 7301, DOI 10.17487/RFC7301, July 2014, <http://www.rfc-editor.org/info/rfc7301>.

[19] Hammer-Lahav, E., Ed. and B. Cook, "Web Host Metadata", RFC 6415, DOI 10.17487/RFC6415, October 2011, <http://www.rfc-editor.org/info/rfc6415>.

[20] Li, L., Wei, Z., Luo, M., and W. Chou, "Requirements and Design Patterns for REST Northbound API in SDN", draft-li-sdnrg-design-restapi-02 (work in progress), July 2016.

[21] Masinter, L., "MIME and the Web", draft-masinter-mime-web-info-02 (work in progress), January 2011.

[22] Richardson, L. and M. Amundsen, "RESTful Web APIs", O'Reilly Media, ISBN 978-1-4493-5806-8, September 2013.

[23] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, DOI 10.17487/RFC6690, August 2012, <http://www.rfc-editor.org/info/rfc6690>.

[24] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <http://www.rfc-editor.org/info/rfc7252>.

[25] Wilde, E., "The 'profile' Link Relation Type", RFC 6906, DOI 10.17487/RFC6906, March 2013, <http://www.rfc-editor.org/info/rfc6906>.

Acknowledgements

Jan Algermissen, Mike Amundsen, Mike Kelly, Julian Reschke, and Erik Wilde provided valuable input on link and form relation types.

Thanks to Olaf Bergmann, Carsten Bormann, Stefanie Gerdes, Ari Keranen, Michael Koster, Matthias Kovatsch, Teemu Savolainen, and Bilhanan Silverajan for helpful comments and discussions that have shaped the document.

Some of the text in this document has been borrowed from [5], [6], [17], [20], and [21]. All errors are my own.

This work was funded in part by Nokia.

Author's Address

Klaus Hartke
Universitaet Bremen TZI
Postfach 330440
Bremen D-28359
Germany

Phone: +49-421-218-63905
Email: hartke@tzi.org

Thing-to-Thing Research Group K. Hartke
Internet-Draft Universitaet Bremen TZI
Intended status: Experimental April 25, 2018
Expires: October 27, 2018

The Constrained RESTful Application Language (CoRAL)
draft-hartke-t2trg-coral-05

Abstract

The Constrained RESTful Application Language (CoRAL) defines a data model and interaction model as well as two specialized serialization formats for the description of typed connections between resources on the Web ("links"), possible operations on such resources ("forms"), and simple resource metadata.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

The Constrained RESTful Application Language (CoRAL) is a language for the description of typed connections between resources on the Web ("links"), possible operations on such resources ("forms"), as well as simple resource metadata.

CoRAL is intended for driving automated software agents that navigate a Web application based on a standardized vocabulary of link and form relation types. It is designed to be used in conjunction with a Web transfer protocol such as the Hypertext Transfer Protocol (HTTP) [RFC7230] or the Constrained Application Protocol (CoAP) [RFC7252].

This document defines the CoRAL data and interaction model, as well as two specialized CoRAL serialization formats.

The CoRAL data and interaction model is a superset of the Web Linking model described in RFC 8288 [RFC8288]. The CoRAL data model consists of two elements: _links_ that describe the relationships between pairs of resources and the type of those relationships, and _forms_ that describe possible operations on resources and the type of those operations. In addition, the data model can describe simple resource metadata in a way similar to the Resource Description Framework (RDF) [W3C.REC-rdf11-concepts-20140225]. However, in contrast to RDF, the focus of CoRAL is on the interaction with resources, not just on the relationships between them. The CoRAL interaction model derives from HTML 5 [W3C.REC-html52-20171214] and specifies how an automated software agent can navigate between resources by following links and perform operations on resources by submitting forms.

The primary CoRAL serialization format is a compact, binary encoding of links and forms in Concise Binary Object Representation (CBOR) [RFC7049]. It is intended for environments with constraints on power, memory, and processing resources [RFC7228], and shares many similarities with the message format of the Constrained Application Protocol (CoAP) [RFC7252]: It uses numeric identifiers instead of verbose strings for link and form relation types, and pre-parses URIs into (what CoAP considers to be) their components, which simplifies URI processing greatly. As a result, link serializations are often much more compact than equivalent serializations in CoRE Link Format [RFC6690], including its CBOR variant [I-D.ietf-core-links-json]. Additionally, CoRAL supports the serialization of forms, which CoRE Link Format does not support.

The secondary CoRAL serialization format is a lightweight, textual encoding of links and forms that is intended to be easy to read and write by humans. The format is loosely inspired by the syntax of Turtle [W3C.REC-turtle-20140225] and is used for giving examples throughout the document.

1.1. Requirements Notation

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 BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

2. Examples

2.1. Web Linking

At its core, CoRAL is just yet another serialization format for Web links. For example, if an HTTP client sends the following request:

```
GET /TheBook/chapter3 HTTP/1.1
Host: example.com
```

and receives the following response:

```
HTTP/1.1 200 OK
Content-Type: text/coral

#using <http://www.iana.org/assignments/relation/>

next    <./chapter4>
icon    </favicon.png>
license <http://creativecommons.org/licenses/by/4.0/>
```

then the representation contains the following three links:

- o one link of type "http://www.iana.org/assignments/relation/next" from <http://example.com/TheBook/chapter3> to <http://example.com/TheBook/chapter4>,

- o one link of type "http://www.iana.org/assignments/relation/icon" from <http://example.com/TheBook/chapter3> to <http://example.com/favicon.png>, and

- o one link of type "http://www.iana.org/assignments/relation/license" from <http://example.com/TheBook/chapter3> to <http://creativecommons.org/licenses/by/4.0/>.

This representation is equivalent to the following Link header field [RFC8288]:

```
Link: <./chapter4>; rel="next",
      </favicon.png>; rel="icon",
      <http://creativecommons.org/licenses/by/4.0/>; rel="license"
```

and the following HTML 5 [W3C.REC-html52-20171214] link elements:

```
<link rel="next" href="./chapter4">
<link rel="icon" href="/favicon.png">
<link rel="license"
      href="http://creativecommons.org/licenses/by/4.0/">
```

2.2. Links, Forms, and Metadata

In its entirety, CoRAL is an expressive language for describing Web links between resources, possible operations on these resources, and simple resource metadata. For example, if an HTTP client sends the following request:

```
GET /tasks HTTP/1.1
Host: example.com
```

and receives the following response:

```
HTTP/1.1 200 OK
Content-Type: text/coral

#using <http://example.org/vocabulary#>
#using coral = <urn:ietf:rfc:XXXX#>

task </tasks/1> {
   description "Pick up the kids"
}

task </tasks/2> {
   description "Return the books to the library"
   coral:delete -> DELETE </tasks/2>
}

coral:create -> POST </tasks> [coral:accept "example/task"]
```

then the representation contains the following six elements:

- one link of type "http://example.org/vocabulary#task" from <http://example.com/tasks> to <http://example.com/tasks/1>,

- one link of type "http://example.org/vocabulary#description" from <http://example.com/tasks/1> to "Pick up the kids",

- one link of type "http://example.org/vocabulary#task" from <http://example.com/tasks> to <http://example.com/tasks/2>,

- one link of type "http://example.org/vocabulary#description" from <http://example.com/tasks/2> to "Return the books to the library",

- one form of type "urn:ietf:rfc:XXXX#delete" that can be used to delete <http://example.com/tasks/2> by making a DELETE request to <http://example.com/tasks/2>, and

- one form of type "urn:ietf:rfc:XXXX#create" that can be used to create a new item in <http://example.com/tasks> by making a POST request to <http://example.com/tasks> with an "example/task" payload.

3. Data and Interaction Model

The Constrained RESTful Application Language (CoRAL) is designed for building Web-based applications [W3C.REC-webarch-20041215] in which automated software agents navigate between resources by following links and perform operations on resources by submitting forms.

3.1. Browsing Context

Borrowing from HTML 5 [W3C.REC-html52-20171214], each such agent maintains a _browsing context_ in which the representations of Web resources are processed. (In HTML 5, the browsing context typically corresponds to a tab or window in a Web browser.)

A browsing context has a _session history_ that lists the resource representations that the agent has processed, is processing, or will process. At any time, one representation in each browsing context is designated the _active_ representation.

A session history consists of a flat list of session history entries. Each _session history entry_ consists of a resource representation and the Internationalized Resource Identifier (IRI) [RFC3987] that was used to retrieve the representation. An entry can additionally have other information associated with it. New entries are added to the session history as the agent navigates from resource to resource.

3.2. Documents

A resource representation in one of the CoRAL serialization formats is called a CoRAL _document_. The IRI that was used to retrieve such a document is called the document's _retrieval context_.

A CoRAL document consists of a list of zero or more links, forms, and embedded resource representations, collectively called _elements_. CoRAL serialization formats may define additional types of elements for efficiency or convenience, such as base IRIs for relative IRI references.

3.3. Links

A _link_ describes a relationship between two resources on the Web [RFC8288]. As defined in RFC 8288, it consists of a _link context_, a _link relation type_, and a _link target_. A link can additionally have a nested list of zero or more elements, which takes the place of link target attributes in CoRAL.

A link can be viewed as a statement of the form "_link context_ has a _link relation type_ resource at _link target_" where the link target may be further described by nested links and forms.

The link relation type identifies the semantics of a link. In HTML 5 and the RFC 8288 Link header field, a link relation type is typically denoted by a registered name, such as "stylesheet" or "icon". In CoRAL, a link relation type is denoted in contrast by an IRI or an unsigned integer. IRIs on the one hand allow for the creation of new, unique relation types in a decentralized fashion, but can incur a high overhead in terms of message size. Small, unsigned integers on the other hand minimize the overhead of link relation types in constrained environments, but require the assignment of values by a registry to avoid collisions.

The link context and the link target are both resources on the Web. Resources are denoted in CoRAL either by an IRI reference [RFC3987] or (similar to RDF) by a literal. If the IRI scheme indicates a Web transfer protocol such as HTTP or CoAP, then an agent can dereference the IRI and navigate the browsing context to the referenced resource; this is called _following the link_. A literal directly identifies a value, which in CoRAL can be a Boolean value, an integer, a floating-point number, a byte string, or a text string.

A link can occur as a top-level element in a document or as a nested element within a link. When a link occurs as a top-level element, the link context is implicitly the document's retrieval context. When a link occurs nested within a link, the link context of the inner link is the link target of the outer link.

There are no restrictions on the cardinality of links; there can be multiple links to and from a particular target, and multiple links of the same or different types between a given link context and target. However, the CoRAL data model constrains the description of a resource graph to a tree: Links between linked resources can only be described by further nesting links.

3.4. Forms

A _form_ provides instructions to an agent for performing an operation on a Web resource. It consists of a _form context_, a _form relation type_, a _request method_, and a _submission IRI_. Additionally, a form may be accompanied by _form data_.

A form can be viewed as an instruction of the form "To perform a _form relation type_ operation on _form context_, make a _request method_ request to _submission IRI_" where the payload of the request may be further described by form data.

The form relation type identifies the semantics of the operation. Like a link relation type, it is denoted by an IRI or an unsigned integer.

The form context is the resource on which an operation is ultimately performed. To perform the operation, an agent needs to construct a request with the specified request method and submission IRI. The submission IRI typically refers to the form context, but MAY refer to different resource. Constructing and sending the request is called _submitting the form_.

If a form is accompanied by form data (Section 3.5), the agent MUST also construct a payload that matches the specifications of the form data and include it in the request when submitting the form.

A form can occur as a top-level element in a document or as a nested element within a link. When a form occurs as a top-level element, the form context is implicitly the document's retrieval context. When a form occurs nested within a link, the form context is the link target of the enclosing link.

3.5. Form Data

Form data provides instructions for agents to construct a request payload. It consists of a list of zero or more _form fields_. Each form field consists of a _form field name_ and a _form field value_.

Form fields can either directly identify data items that need to be included in the request payload or reference another resource (such as a schema) that describes the data items. Form fields may also provide other information, such as acceptable representation formats.

The form field name identifies the semantics of the form field. Like a link or form relation type, a form field name is denoted by an IRI or an unsigned integer.

The form field value can be an IRI, a Boolean value, an integer, a floating-point number, a byte string, or a text string.

3.6. Representations

If a representation links to many resources and an agent requires a representation of each link target, it may be inefficient to retrieve each representation individually. To alleviate this, CoRAL supports the embedding of a _representation_ of a resource in a document.

An embedded representation consists of a sequence of bytes, plus _representation metadata_ to describe those bytes. It may be a full, partial, or inconsistent version of the representation served from the IRI of the represented resource.

An embedded representation can occur as a top-level element in a document or as a nested element within a link. When it occurs as a top-level element, it provides an alternate representation of the document's retrieval context. When it occurs nested within a link, it provides a representation of link target of the enclosing link.

3.7. Navigation

An agent begins interacting with an application by performing a GET request on an _entry point IRI_. The entry point IRI is the only IRI an agent is expected to know before interacting with an application. From there, the agent is expected to make all requests by following links and submitting forms provided by the server in responses. The entry point IRI can be obtained by manual configuration or through some discovery process.

If dereferencing the entry point IRI yields a CoRAL document or any other representation that implements the CoRAL data and interaction model, then the agent proceeds as follows:

1. The first step for the agent is to decide what to do next, i.e., which type of link to follow or form to submit, based on the link relation types and form relation types it understands.

2. The agent finds the link(s) or form(s) with the given relation type in the active representation. This may yield one or more candidates, from which the agent must select the most appropriate one in the next step. The set of candidates MAY be empty, for example, if a transition is not supported or not allowed.

3. The agent selects one of the candidates based on the metadata associated with the link(s) or form(s). Metadata typically includes the media type of the target resource representation, the IRI scheme, the request method, and other information that is provided as nested elements in a link and form data in a form.

   If the selected candidate contains an embedded representation, then the agent MAY skip the following steps and immediately proceed with step 8.

4. The agent resolves the IRI reference in the link or form as specified in Section 5 of RFC 3986 [RFC3986] to obtain the _request IRI_. Fragment identifiers are not part of the request IRI and MUST be separated from the rest of the IRI prior to a dereference. The request IRI may need to be converted to a URI (Section 3.1 of RFC 3987 [RFC3987]) for protocols that do not support IRIs.

5. The agent constructs a new request with the request IRI. If the agent follows a link, the request method MUST be GET. If the agent submits a form, the request method MUST be the one specified in the form. The agent SHOULD set HTTP header fields and CoAP request options according to provided metadata (e.g., set the HTTP Accept header field or the CoAP Accept option when

the media type of the target resource is provided). In case of a form with form data, the agent MUST include a request payload that matches the specifications of the form data.

6. The agent sends the request and retrieves the response.

7. If a fragment identifier was separated from the request IRI, the agent dereferences the fragment identifier within the retrieved representation.

8. The agent _updates the session history_: It removes all the entries in the browsing context's session history after the current entry. Then it appends a new entry at the end of the history representing the new resource.

9. Finally, the agent processes the representation. In case of a CoRAL document or any other representation that implements the CoRAL data and interaction model, this means the agent decides again what to do next and the cycle repeats.

3.8. History Traversal

An agent can also navigate a browsing context by traversing the browsing context's session history. An agent can _traverse the session history_ by updating the active representation to the that entry.

4. Binary Format

This section defines the encoding of documents in the CoRAL binary format.

A document in the binary format is a data item in Concise Binary Object Representation (CBOR) [RFC7049]. The structure of this data item is presented in the Concise Data Definition Language (CDDL) [I-D.ietf-cbor-cddl]. The media type is "application/coral+cbor".

4.1. Data Structure

The data structure of a document in the binary format is made up of four kinds of elements: links, forms, embedded representations, and (as an extension to the CoRAL data model) base IRI directives. Base IRI directives provide a way to encode IRI references that have a common base more efficiently.

Elements are processed in the order they appear in the document. Document processors need to maintain an _environment_ while iterating an array of elements. The environment consists of three variables: a _current context IRI_, a _current base IRI_, and a _current relation type_. The current context IRI and current base IRI are initially both set to the document's retrieval context. The current relation type is initially set to the unsigned integer zero.

4.1.1. Documents

The body of a document in the binary format is encoded as an array of zero or more links, forms, embedded representations, and directives.

```
body = [*(link / form / representation / directive)]
```

4.1.2. Links

A link is encoded as an array that consists of the unsigned integer 2, followed by the link relation type and the link target, optionally followed by a link body that contains nested elements.

```
link = [link: 2, relation, target, ?body]
```

The link relation type is encoded either as a text string containing an absolute IRI reference or as an (unsigned or negative) integer representing the difference to the current relation type. A link is processed by updating the current relation type to the result of adding the specified integer (or zero in the case of a text string) to the current relation type.

```
relation = text / int
```

The link target is denoted by an IRI reference or represented by a literal value. The IRI reference MAY be relative or absolute and MUST be resolved against the current base IRI. The encoding of IRI references in the binary format is described in Appendix C. The link target MAY be null, which indicates that the link target is an unidentified resource.

```
target = iri / literal / null

literal = bool / int / float / bytes / text
```

The array of elements in the link body (if any) MUST be processed in a fresh environment. The current context IRI and current base IRI in the new environment are initially both set to the link target of the enclosing link. The current relation type in the new environment is initially set to the current relation type.

4.1.3. Forms

A form is encoded as an array that consists of the unsigned integer 3, followed by the form relation type, the submission method, and a submission IRI reference, optionally followed by form data.

```
form = [form: 3, relation, method, iri, ?form-data]
```

The form relation type is encoded and processed in the same way as a link relation type (Section 4.1.2).

The method MUST refer to one of the request methods defined by the Web transfer protocol identified by the scheme of the submission IRI. It is encoded either as a text string or an unsigned integer.

```
method = text / uint
```

For HTTP [RFC7230], the method MUST be encoded as a text string in the format defined in Section 4.1 of RFC 7231 [RFC7231]; the set of possible values is maintained in the IANA HTTP Method Registry. For CoAP [RFC7252], the method MUST be encoded as an unsigned integer (e.g., the unsigned integer 2 for the POST method); the set of possible values is maintained in the IANA CoAP Method Codes Registry.

The submission IRI reference MAY be relative or absolute and MUST be resolved against the current base IRI. The encoding of IRI references in the binary format is described in Appendix C.

4.1.3.1. Form Data

Form data is encoded as an array of zero or more name-value pairs.

```
form-data = [*(form-field-name, form-field-value)]
```

Form data (if any) MUST be processed in a fresh environment. The current context IRI and current base IRI in the new environment are initially both set to the submission IRI of the enclosing form. The current relation type in the new environment is initially set to the current relation type.

A form field name is encoded and processed in the same way as a link relation type (Section 4.1.2).

```
form-field-name = text / uint
```

A form field value can be an IRI reference, a Boolean value, an integer, a floating-point number, a byte string, a text string, or null. An IRI reference MAY be relative or absolute and MUST be resolved against the current base IRI. The encoding of IRI references in the binary format is described in Appendix C.

```
form-field-value = iri / bool / int / float / bytes / text / null
```

4.1.3.2. Short Forms

Forms in certain shapes can be encoded in a more efficient manner using short forms. The following short forms are available:

```
form /= [form.create: 4, ?accept: uint .size 2]

form /= [form.update: 5, ?accept: uint .size 2]

form /= [form.delete: 6]

form /= [form.search: 7, ?accept: uint .size 2]
```

If the scheme of the submission IRI indicates HTTP, the short forms expand as follows:

```
[4]     ->  [3, "urn:ietf:rfc:XXXX#create", "POST", []]
[4, x]  ->  [3, "urn:ietf:rfc:XXXX#create", "POST", [],
                ["urn:ietf:rfc:XXXX#accept", x]]
[5]     ->  [3, "urn:ietf:rfc:XXXX#update", "PUT", []]
[5, x]  ->  [3, "urn:ietf:rfc:XXXX#update", "PUT", [],
                ["urn:ietf:rfc:XXXX#accept", x]]
[6]     ->  [3, "urn:ietf:rfc:XXXX#delete", "DELETE", []]
[7]     ->  [3, "urn:ietf:rfc:XXXX#search", "POST", []]
[7, x]  ->  [3, "urn:ietf:rfc:XXXX#search", "POST", [],
                ["urn:ietf:rfc:XXXX#accept", x]]
```

If the scheme of the submission IRI indicates CoAP, the short forms expand as follows:

```
[4]     ->  [3, "urn:ietf:rfc:XXXX#create", 2, []]
[4, x]  ->  [3, "urn:ietf:rfc:XXXX#create", 2, [],
                ["urn:ietf:rfc:XXXX#accept", x]]
[5]     ->  [3, "urn:ietf:rfc:XXXX#update", 3, []]
[5, x]  ->  [3, "urn:ietf:rfc:XXXX#update", 3, [],
                ["urn:ietf:rfc:XXXX#accept", x]]
[6]     ->  [3, "urn:ietf:rfc:XXXX#delete", 4, []]
[7]     ->  [3, "urn:ietf:rfc:XXXX#create", 5, []]
[7, x]  ->  [3, "urn:ietf:rfc:XXXX#create", 5, [],
                ["urn:ietf:rfc:XXXX#accept", x]]
```

The form relation types and form field names used in the above expansions are defined in Appendix A.

4.1.4. Representations

An embedded representation is encoded as an array that consists of the unsigned integer 0, followed by the HTTP content type or CoAP content format of the representation and a byte string containing the representation data.

```
representation = [representation: 0, text / uint, bytes]
```

For HTTP, the content type MUST be specified as a text string in the format defined in Section 3.1.1.1 of RFC 7231 [RFC7231]; the set of possible values is maintained in the IANA Media Types Registry. For CoAP, the content format MUST be specified as an unsigned integer; the set of possible values is maintained in the IANA CoAP Content-Formats Registry.

4.1.5. Directives

Directives provide the ability to manipulate the environment when processing a list of elements. There is one directive available: the Base URI directive.

```
directive = base-directive
```

4.1.5.1. Base URI Directives

A Base IRI directive is encoded as an array that consists of the unsigned integer 1, followed by an IRI reference.

```
base-directive = [base: 1, iri]
```

The IRI reference MAY be relative or absolute and MUST be resolved against the current context IRI. The encoding of IRI references in the binary format is described in Appendix C.

The directive is processed by resolving the IRI reference against the current context IRI and assigning the result to the current base IRI.

5. Textual Format

This section defines the syntax of documents in the CoRAL textual format using two grammars: The lexical grammar defines how Unicode characters are combined to form line terminators, white space, comments, and tokens. The syntactic grammar defines how the tokens are combined to form documents. Both grammars are presented in Augmented Backus-Naur Form (ABNF) [RFC5234].

A document in the textual format is a Unicode string in a Unicode encoding form [UNICODE]. The media type for such documents is "text/coral". The "charset" parameter is not used; charset information is transported inside the document in the form of an OPTIONAL Byte Order Mark (BOM). The use of the UTF-8 encoding scheme [RFC3629], without a BOM, is RECOMMENDED.

5.1. Lexical Structure

The lexical structure of a document in the textual format is made up of four basic elements: line terminators, white space, comments, and tokens. Of these, only tokens are significant in the syntactic grammar. There are four kinds of tokens: identifiers, IRI references, literals, and punctuators.

When several lexical grammar rules match a sequence of characters in a document, the longest match takes priority.

5.1.1. Line Terminators

Line terminators divide text into lines. A line terminator is any Unicode character with Line_Break class BK, CR, LF, or NL. However, any CR character that immediately precedes a LF character is ignored. (This affects only the numbering of lines in error messages.)

5.1.2. White Space

White space is a sequence of one or more white space characters. A white space character is any Unicode character with the White_Space property.

5.1.3. Comments

Comments are sequences of characters that are ignored when parsing text into tokens. Single-line comments begin with the characters "//" and extend to the end of the line. Delimited comments begin with the characters "/*" and end with the characters "*/". Delimited comments can occupy a portion of a line, a single line, or multiple lines.

Comments do not nest. The character sequences "/*" and "*/" have no special meaning within a single-line comment; the character sequences "//" and "/*" have no special meaning within a delimited comment.

5.1.4. Identifiers

An identifier tokens is a user-defined symbolic name. The rules for identifiers correspond to those recommended by the Unicode Standard Annex #31 [UNICODE-UAX31] using the following profile:

```
identifier = start *continue *(medial 1*continue)

start = <Any character with the XID_Start property>

continue = <Any character with the XID_Continue property>

medial = "-" / "." / "~" / %xB7 / %x58A / %xF0B

medial =/ %x2010 / %x2027 / %x30A0 / %x30FB
```

All identifiers MUST be converted into Unicode Normalization Form C (NFC), as defined by the Unicode Standard Annex #15 [UNICODE-UAX15]. Comparison of identifiers is based on NFC and is case-sensitive (unless otherwise noted).

5.1.5. IRI References

An IRI reference is a Unicode string that conforms to the syntax defined in RFC 3987 [RFC3987]. An IRI reference can be absolute or relative and can contain a fragment identifier. IRI references are enclosed in angle brackets ("<" and ">").

```
iri = "<" IRI-reference ">"

IRI-reference = <Defined in Section 2.2 of RFC 3987>
```

5.1.6. Literals

A literal is a textual representation of a value. There are six types of literals: Boolean, integer, floating-point, byte string, text string, and null.

5.1.6.1. Boolean Literals

The case-insensitive tokens "true" and "false" denote the Boolean values true and false, respectively.

```
boolean = "true" / "false"
```

5.1.6.2. Integer Literals

Integer literals denote integer values of unspecified precision. By default, integer literals are expressed in decimal, but they can also be specified in an alternate base using a prefix. Binary literals begin with "0b", octal literals begin with "0o", and hexadecimal literals begin with "0x".

Decimal literals contain the digits "0" through "9". Binary literals contain "0" and "1", octal literals contain "0" through "7", and hexadecimal literals contain "0" through "9" as well as "A" through "F" in upper- or lowercase.

Negative integers are expressed by prepending a minus sign ("-").

```
integer = ["+" / "-"] (decimal / binary / octal / hexadecimal)

decimal = 1*DIGIT

binary = %x30 (%x42 / %x62) 1*BINDIG

octal = %x30 (%x4F / %x6F) 1*OCTDIG

hexadecimal = %x30 (%x58 / %x78) 1*HEXDIG

DIGIT = %x30-39

BINDIG = %x30-31

OCTDIG = %x30-37

HEXDIG = %x30-39 / %x41-46 / %x61-66
```

5.1.6.3. Floating-point Literals

Floating-point literals denote floating-point numbers of unspecified precision.

Floating-point literals consist of a sequence of decimal digits followed by a fraction, an exponent, or both. The fraction consists of a decimal point (".") followed by a sequence of decimal digits. The exponent consists of the letter "e" in upper- or lowercase, followed by an optional sign and a sequence of decimal digits that indicate a power of 10 by which the value preceding the "e" is multiplied.

Negative floating-point values are expressed by prepending a minus sign ("-").

```
floating-point = ["+" / "-"] 1*DIGIT [fraction] [exponent]

fraction = "." 1*DIGIT

exponent = (%x45 / %x65) ["+" / "-"] 1*DIGIT
```

Floating-point literals can additionally denote the special "Not-a-Number" (NaN) value, positive infinity, and negative infinity. The NaN value is produced by the case-insensitive token "NaN". The two infinite values are produced by the case-insensitive tokens "+Infinity" (or simply "Infinity") and "-Infinity".

```
floating-point =/ "NaN"

floating-point =/ ["+" / "-"] "Infinity"
```

5.1.6.4. Byte String Literals

A byte string literal consists of a prefix and zero or more bytes encoded in Base16, Base32, or Base64 [RFC4648] and enclosed in single quotes. Byte string literals encoded in Base16 begin with "h" or "b16", byte string literals encoded in Base32 begin with "b32", and byte string literals encoded in Base64 begin with "b64".

```
bytes = base16 / base32 / base64

base16 = (%x68 / %x62.31.36) SQUOTE <Base16 encoded data> SQUOTE

base32 = %x62.33.32 SQUOTE <Base32 encoded data> SQUOTE

base64 = %x62.36.34 SQUOTE <Base64 encoded data> SQUOTE

SQUOTE = %x27
```

5.1.6.5. Text String Literals

A text string literal consists of zero or more Unicode characters enclosed in double quotes. It can include simple escape sequences (such as \t for the tab character) as well as hexadecimal and Unicode escape sequences.

```
text = DQUOTE *(char / %x5C escape) DQUOTE

char = <Any character except %x22, %x5C, and line terminators>

escape = simple-escape / hexadecimal-escape / unicode-escape

simple-escape = %x30 / %x62 / %x74 / %x6E / %x76
```

```
simple-escape =/ %x66 / %x72 / %x22 / %x27 / %x5C

hexadecimal-escape = (%x78 / %x58) 2HEXDIG

unicode-escape = %x75 4HEXDIG / %x55 8HEXDIG

DQUOTE = %x22
```

An escape sequence denotes a single Unicode code point. For hexadecimal and Unicode escape sequences, the code point is expressed by the hexadecimal number following the "\x", "\X", "\u", or "\U" prefix. Simple escape sequences indicate the code points listed in Table 1.

Escape Sequence	Code Point	Character Name
\0	U+0000	Null
\b	U+0008	Backspace
\t	U+0009	Character Tabulation
\n	U+000A	Line Feed
\v	U+000B	Line Tabulation
\f	U+000C	Form Feed
\r	U+000D	Carriage Return
\"	U+0022	Quotation Mark
\'	U+0027	Apostrophe
\\	U+005C	Reverse Solidus

Table 1: Simple Escape Sequences

5.1.6.6. Null Literal

The case-insensitive tokens "null" and "_" denote the intentional absence of any value.

```
null = "null" / "_"
```

5.1.7. Punctuators

Punctuator tokens are used for grouping and separating.

```
punctuator = "#" | ":" | "*" | "[" | "]" | "{" | "}" | "=" | "->"
```

5.2. Syntactic Structure

The syntactic structure of a document in the textual format is made up of four kinds of elements: links, forms, embedded representations, and (as an extension to the CoRAL data model) directives. Directives provide a way to make documents easier to read and write by defining base IRIs for relative IRI references and introducing shorthands for IRIs.

Elements are processed in the order they appear in the document. Document processors need to maintain an _environment_ while iterating a list of elements. The environment consists of three variables: a _current context IRI_, a _current base IRI_, and a _current mapping from identifiers to IRIs_. The current context IRI and current base IRI are initially both set to the document's retrieval context. The current mapping from identifiers to IRIs is initially empty.

5.2.1. Documents

The body of a document in the textual format consists of zero or more links, forms, and directives.

```
body = *(link / form / representation / directive)
```

5.2.2. Links

A link consists of the link relation type, followed by the link target, optionally followed by a link body enclosed in curly brackets ("{" and "}").

```
link = relation target ["{" body "}"]
```

The link relation type is denoted either by an absolute IRI reference, a simple name, a qualified name, or an integer.

```
relation = iri / simple-name / qualified-name / integer
```

A simple name consists of an identifier. It is resolved to an IRI by looking up the empty string in the current mapping from identifiers to IRIs and appending the specified identifier to the result. It is an error if the empty string is not present in the mapping.

```
simple-name = identifier
```

A qualified name consists of two identifiers separated by a colon (":"). It is resolved to an IRI by looking up the identifier on the left hand side in the current mapping from identifiers to IRIs and appending the identifier on the right hand side to the result. It is an error if the identifier on the left hand side is not present in the mapping.

```
qualified-name = identifier ":" identifier
```

The link target is denoted by an IRI reference or represented by a value literal. The IRI reference MAY be relative or absolute and MUST be resolved against the current base IRI. If the link target is null, the link target is an unidentified resource.

```
target = iri / literal / null

literal = boolean / integer / floating-point / bytes / text
```

The list of elements in the link body (if any) MUST be processed in a fresh environment. The current context IRI and current base IRI in this environment are initially both set to the link target of the enclosing link. The mapping from identifiers to IRIs is initially set to a copy of the mapping from identifiers to IRIs in the current environment.

5.2.3. Forms

A form consists of the form relation type, followed by a "->" token, a method identifier, and a submission IRI reference, optionally followed by form data enclosed in square brackets ("[" and "]").

```
form = relation "->" method iri ["[" form-data "]"]
```

The form relation type is denoted in the same way as a link relation type (Section 5.2.2).

The method identifier refers to one of the request methods defined by the Web transfer protocol identified by the scheme of the submission IRI. Method identifiers are case-insensitive and constrained to Unicode characters in the Basic Latin block.

```
method = identifier
```

For HTTP [RFC7230], the set of possible method identifiers is maintained in the IANA HTTP Method Registry. For CoAP [RFC7252], the set of possible method identifiers is maintained in the IANA CoAP Method Codes Registry.

The submission IRI reference MAY be relative or absolute and MUST be resolved against the current base IRI.

5.2.3.1. Form Data

Form data consists of zero or more name-value pairs.

```
form-data = *(form-field-name form-field-value)
```

Form data MUST be processed in a fresh environment. The current context IRI and current base IRI in this environment are initially both set to the submission IRI of the enclosing form. The mapping from identifiers to IRIs is initially set to a copy of the mapping from identifiers to IRIs in the current environment.

The form field name is denoted in the same way as a link relation type (Section 5.2.2).

```
form-field-name = iri / simple-name / qualified-name / integer
```

The form field value can be an IRI reference, Boolean literal, integer literal, floating-point literal, byte string literal, text string literal, or null. An IRI reference MAY be relative or absolute and MUST be resolved against the current base IRI.

```
form-field-value = iri / boolean / integer

form-field-value =/ floating-point / bytes / text / null
```

5.2.4. Representations

An embedded representation consists of a "*" token, followed by the representation data, optionally followed by representation metadata enclosed in square brackets ("[" and "]").

```
representation = "*" bytes ["[" representation-metadata "]"]
```

Representation metadata consists of zero or more name-value pairs.

```
representation-metadata = *(metadata-name metadata-value)
```

This document specifies only one kind of metadata item, labeled with the name "type": the HTTP content type or CoAP content format of the representation.

```
metadata-name = "type"

metadata-value = text / integer
```

For HTTP, the content type MUST be specified as a text string in the format defined in Section 3.1.1.1 of RFC 7231 [RFC7231]; the set of possible values is maintained in the IANA Media Types Registry. For CoAP, the content format MUST be specified as an integer; the set of possible values is maintained in the IANA CoAP Content-Formats Registry.

A metadata item with the name "type" MUST NOT occur more than once. If absent, its value defaults to content type "application/octet-stream" or content format 42.

5.2.5. Directives

Directives provide the ability to manipulate the environment when processing a list of elements. All directives start with a number sign ("#") followed by a directive identifier. Directive identifiers are case-insensitive and constrained to Unicode characters in the Basic Latin block.

The following directives are available: Base IRI directives and Using directives.

```
directive = base-directive / using-directive
```

5.2.5.1. Base IRI Directives

A Base IRI directive consists of a number sign ("#"), followed by the case-insensitive identifier "base", followed by an IRI reference.

```
base-directive = "#" "base" iri
```

The IRI reference MAY be relative or absolute and MUST be resolved against the current context IRI.

The directive is processed by resolving the IRI reference against the current context IRI and assigning the result to the current base IRI.

5.2.5.2. Using Directives

A Using directive consists of a number sign ("#"), followed by the case-insensitive identifier "using", optionally followed by an identifier and an equals sign ("="), finally followed by an absolute IRI reference. If the identifier is not specified, it is assumed to be the empty string.

```
using-directive = "#" "using" [identifier "="] iri
```

The IRI reference MUST be absolute.

The directive is processed by adding the specified identifier and IRI to the current mapping from identifiers to IRIs. It is an error if the identifier is already present in the mapping.

6. Usage Considerations

This section discusses some considerations in creating CoRAL-based applications and managing link and form relation types.

6.1. Specifying CoRAL-based Applications

CoRAL-based applications naturally implement the Web architecture [W3C.REC-webarch-20041215] and thus are centered around orthogonal specifications for identification, interaction, and representation:

- o Resources are identified by IRIs or represented by value literals.

- o Interactions are based on the hypermedia interaction model of the Web and the methods provided by the Web transfer protocol. The semantics of possible interactions are identified by link and form relation types.

- o Representations are CoRAL documents encoded in the binary format defined in Section 4 or the textual format defined in Section 5. Depending on the application, additional representation formats can be used.

Specifications for CoRAL-based applications need to list the specific components used in the application and their identifiers. This SHOULD include at least the following items:

- o IRI schemes that identify the Web transfer protocol(s) used in the application.

- o Internet media types that identify the representation format(s) used in the application, including the media type(s) of the CoRAL serialization format(s).

- o Link relation types that identify the semantics of links.

- o Form relation types that identify the semantics of forms. Additionally, for each form relation type, the permissible request method(s).

- o Form field names that identify the semantics of form fields. Additionally, for each form field name, the permissible form field value(s) or type(s).

6.1.1. Naming Resources

Resource names -- i.e., URIs [RFC3986] and IRIs [RFC3987] -- are a cornerstone of Web-based applications. They enable uniform identification of resources and are used every time a client interacts with a server or a resource representation needs to refer to another resource.

URIs and IRIs often include structured application data in the path and query components, such as paths in a filesystem or keys in a database. It is a common practice in many HTTP-based applications to make this part of the application specification, i.e., they prescribe fixed URI templates that are hard-coded in implementations. However, there are a number of problems with this practice [RFC7320].

In CoRAL-based applications, resource names are not part of the application specification; they are an implementation detail. The specification of a CoRAL-based application MUST NOT mandate any particular form of resource name structure. BCP 190 [RFC7320] describes the problematic practice of fixed URI structures in more detail and provides some acceptable alternatives.

6.1.2. Implementation Limits

This document places no restrictions on the number of elements in a CoRAL document or the depth of nested elements. Applications using CoRAL (in particular those that run in constrained environments) MAY wish to limit these numbers and specify implementation limits that an application implementation MUST at least support to be interoperable. Implementation limits MAY also include the following as well as other items:

- o use of only either the binary format or the text format;
- o use of only either HTTP or CoAP as the Web transfer protocol;
- o use of only either IRIs or unsigned integers to denote link relation types, form relation types, and form field names;
- o use of only either short forms or long forms in the binary format;
- o use of only either HTTP content types or CoAP content formats;
- o use of IRI references only up to a specific length;
- o use of CBOR in a canonical format (Section 3.9 of RFC 7049 [RFC7049]).

6.2. Minting New Relation Types

New link relation types, form relation types, and form field names can be minted by defining an IRI [RFC3987] that uniquely identifies the item. Although the IRI can point to a resource that contains a definition of the semantics of the relation type, clients SHOULD NOT automatically access that resource to avoid overburdening its server. The IRI SHOULD be under the control of the person or party defining it, or be delegated to them.

Link relation types registered in the IANA Link Relations Registry, such as "collection" [RFC6573] or "icon" [W3C.REC-html52-20171214], can be used in CoRAL by appending the registered name to the IRI <http://www.iana.org/assignments/relation/>:

```
#using iana = <http://www.iana.org/assignments/relation/>

iana:collection </items>
iana:icon       </favicon.png>
```

A good source for link relation types for resource metadata are RDF predicates [W3C.REC-rdf11-concepts-20140225]. An RDF statement says that some relationship, indicated by a predicate, holds between two resources. RDF predicates and link relation types can therefore often be used interchangeably. For example, a CoRAL document could describe its creator by using the FOAF vocabulary [FOAF]:

```
#using iana = <http://www.iana.org/assignments/relation/>
#using foaf = <http://xmlns.com/foaf/0.1/>

foaf:maker _ {
   iana:type       <http://xmlns.com/foaf/0.1/Person>
   foaf:familyName "Hartke"
   foaf:givenName  "Klaus"
   foaf:mbox       <mailto:hartke@tzi.org>
}
```

6.3. Registering Relation Types

IRIs that identify link relation types, form relation types, and form field names do not need to be registered. The inclusion of DNS names in IRIs allows for the decentralized creation of new IRIs without the risk of collisions.

However, IRIs can be relatively verbose and impose a high overhead on representations. This can be a problem in constrained environments [RFC7228]. Therefore, CoRAL alternatively allows the use of unsigned integers to identify link relation types, form relation types, and form field names. These impose a much smaller overhead but instead need to be assigned by a registry to avoid collisions.

This document does not create a registry for such integers. Instead, the media types for CoRAL documents in the binary and textual format are defined to have a "profile" parameter [RFC6906] that determines the registry in use. The registry is identified by a URI [RFC3986]. For example, a CoRAL document that uses the registry identified by the URI <http://example.com/registry> can use the following media type:

```
application/coral+cbor; profile="http://example.com/registry"
```

The URI serves only as an identifier; it does not necessarily have to be dereferencable (or even use a dereferencable URI scheme). It is permissible, though, to use a dereferencable URI and serve a representation that provides information about the registry in a human- or machine-readable way. (The format of such a representation is outside the scope of this document.)

For simplicity, a CoRAL document can use unsigned integers from at most one registry. The "profile" parameter of the CoRAL media types MUST contain a single URI, not a white space separated list of URIs as recommended in RFC 6906 [RFC6906]. If the "profile" parameter is absent, the default profile specified in Appendix B is assumed.

A CoRAL registry SHOULD map each unsigned integer to a full IRI that identifies a link relation type, form relation type, or form field name. The namespaces for these three kinds of identifiers are disjoint, i.e., the same integer MAY be assigned to a link relation type, form relation type, and form field name without ambiguity. Once an integer has been assigned, the assignment MUST NOT be changed or removed. A registry MAY provide additional information about an assignment (for example, whether a link relation type is deprecated).

In CoAP [RFC7252], media types (including specific values for their parameters) are encoded as a small, unsigned integer called the content format. For use with CoAP, each CoRAL registry needs to register a new content format in the IANA CoAP Content-Formats Registry. Each such registered content format MUST specify a CoRAL media type with a "profile" parameter that contains the registry URI.

6.4. Expressing Link Target Attributes

Link target attributes defined for use with CoRE Link Format [RFC6690] (such as "type", "hreflang", "media", "ct", "rt", "if", "sz", and "obs") can be expressed in CoRAL by nesting links under the respective link and specifying the attribute name appended to the IRI <http://TBD/> as the link relation type.

If the expressed link target attribute has a value, the target of the nested link MUST be a text string; otherwise, the target MUST be the Boolean value "true":

```
#using iana = <http://www.iana.org/assignments/relation/>
#using attr = <http://TBD/>

iana:item </patches/1> {
   attr:type "application/json-patch+json"
   attr:ct   "51"
   attr:sz   "247"
   attr:obs  true
}
```

[[NOTE TO RFC EDITOR: Please replace all occurrences of "http://TBD/" with a RFC-Editor-controlled IRI.]]

Link target attributes that do not actually describe the link target but the link itself (such as "rel", "anchor", "rev", "title", and "title*") are excluded from this provision and MUST NOT occur in a CoRAL document.

6.5. Embedding CoRAL in CBOR Structures

Data items in the CoRAL binary format (Section 4) MAY be embedded in other CBOR [RFC7049] data structures. Specifications using CDDL [I-D.ietf-cbor-cddl] SHOULD reference the following CDDL definitions for this purpose:

```
CoRAL-Body = body

CoRAL-Link = link

CoRAL-Form = form

CoRAL-IRI = iri
```

7. Security Considerations

Parsers of CoRAL documents must operate on input that is assumed to be untrusted. This means that parsers MUST fail gracefully in the face of malicious inputs. Additionally, parsers MUST be prepared to deal with resource exhaustion (e.g., resulting from the allocation of big data items) or exhaustion of the stack depth (stack overflow).

See Section 8 of RFC 7049 [RFC7049] for security considerations relating to parsing CBOR.

Implementers of the CoRAL textual format need to consider the security aspects of handling Unicode input. See the Unicode Standard Annex #36 [UNICODE-UAX36] for security considerations relating to visual spoofing and misuse of character encodings. See Section 10 of RFC 3629 [RFC3629] for security considerations relating to UTF-8.

CoRAL makes extensive use of IRIs and URIs. See Section 8 of RFC 3987 [RFC3987] for security considerations relating to IRIs. See Section 7 of RFC 3986 [RFC3986] for security considerations relating to URIs.

The security of applications using CoRAL can depend on the proper preparation and comparison of internationalized strings. For example, such strings can be used to make authentication and authorization decisions, and the security of an application could be compromised if an entity providing a given string is connected to the wrong account or online resource based on different interpretations of the string. See RFC 6943 [RFC6943] for security considerations relating to identifiers in IRIs and other locations.

CoRAL is intended to be used in conjunction with a Web transfer protocol such as HTTP or CoAP. See Section 9 of RFC 7320 [RFC7230], Section 9 of RFC 7231 [RFC7231], etc. for security considerations relating to HTTP. See Section 11 of RFC 7252 [RFC7252] for security considerations relating to CoAP.

CoRAL does not define any specific mechanisms for protecting the confidentiality and integrity of CoRAL documents. It relies on application layer or transport layer mechanisms for this, such as Transport Layer Security (TLS) [RFC5246].

CoRAL documents and the structure of a web of resources revealed from automatically following links can disclose personal information and other sensitive information. Implementations need to prevent the unintentional disclosure of such information. See Section of 9 of RFC 7231 [RFC7231] for additional considerations.

Applications using CoRAL ought to consider the attack vectors opened by automatically following, trusting, or otherwise using links and forms in CoRAL documents. In particular, a server that is authoritative for the CoRAL representation of a resource may not necessarily be the authoritative source for nested links and forms.

8. IANA Considerations

8.1. Media Type "application/coral+cbor"

This document registers the media type "application/coral+cbor" according to the procedures of BCP 13 [RFC6838].

Type name:
application

Subtype name:
coral+cbor

Required parameters:
N/A

Optional parameters:
profile - See Section 6.3 of [I-D.hartke-t2trg-coral].

Encoding considerations:
binary - See Section 4 of [I-D.hartke-t2trg-coral].

Security considerations:
See Section 7 of [I-D.hartke-t2trg-coral].

Interoperability considerations:
N/A

Published specification:
[I-D.hartke-t2trg-coral]

Applications that use this media type:
See Section 1 of [I-D.hartke-t2trg-coral].

Fragment identifier considerations:
As specified for "application/cbor".

Additional information:
Deprecated alias names for this type: N/A
Magic number(s): N/A
File extension(s): N/A
Macintosh file type code(s): N/A

Person & email address to contact for further information:
See the Author's Address section of [I-D.hartke-t2trg-coral].

Intended usage:
COMMON

Restrictions on usage:
N/A

Author:
See the Author's Address section of [I-D.hartke-t2trg-coral].

Change controller:
IESG

Provisional registration?
No

8.2. Media Type "text/coral"

This document registers the media type "text/coral" according to the procedures of BCP 13 [RFC6838] and guidelines in RFC 6657 [RFC6657].

Type name:
text

Subtype name:
coral

Required parameters:
N/A

Optional parameters:
profile - See Section 6.3 of [I-D.hartke-t2trg-coral].

Encoding considerations:
binary - See Section 5 of [I-D.hartke-t2trg-coral].

Security considerations:
See Section 7 of [I-D.hartke-t2trg-coral].

Interoperability considerations:
N/A

Published specification:
[I-D.hartke-t2trg-coral]

Applications that use this media type:
See Section 1 of [I-D.hartke-t2trg-coral].

Fragment identifier considerations:
N/A

Additional information:

Deprecated alias names for this type: N/A
Magic number(s): N/A
File extension(s): .coral
Macintosh file type code(s): TEXT

Person & email address to contact for further information:
See the Author's Address section of [I-D.hartke-t2trg-coral].

Intended usage:
COMMON

Restrictions on usage:
N/A

Author:
See the Author's Address section of [I-D.hartke-t2trg-coral].

Change controller:
IESG

Provisional registration?
No

8.3. CoAP Content Formats

This document registers CoAP content formats for the media types "application/coral+cbor" and "text/coral" according to the procedures of RFC 7252 [RFC7252].

- Media Type: application/coral+cbor
  Content Coding: identity
  ID: TBD (maybe 63)
  Reference: [I-D.hartke-t2trg-coral]

- Media Type: text/coral
  Content Coding: identity
  ID: TBD (maybe 10063)
  Reference: [I-D.hartke-t2trg-coral]

9. References

9.1. Normative References

[I-D.ietf-cbor-cddl]
Birkholz, H., Vigano, C., and C. Bormann, "Concise data definition language (CDDL): a notational convention to express CBOR data structures", draft-ietf-cbor-cddl-02 (work in progress), February 2018.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, DOI 10.17487/RFC3629, November 2003, <https://www.rfc-editor.org/info/rfc3629>.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <https://www.rfc-editor.org/info/rfc3986>.

[RFC3987] Duerst, M. and M. Suignard, "Internationalized Resource Identifiers (IRIs)", RFC 3987, DOI 10.17487/RFC3987, January 2005, <https://www.rfc-editor.org/info/rfc3987>.

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, <https://www.rfc-editor.org/info/rfc4648>.

[RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <https://www.rfc-editor.org/info/rfc5234>.

[RFC6657] Melnikov, A. and J. Reschke, "Update to MIME regarding "charset" Parameter Handling in Textual Media Types", RFC 6657, DOI 10.17487/RFC6657, July 2012, <https://www.rfc-editor.org/info/rfc6657>.

[RFC6838] Freed, N., Klensin, J., and T. Hansen, "Media Type Specifications and Registration Procedures", BCP 13, RFC 6838, DOI 10.17487/RFC6838, January 2013, <https://www.rfc-editor.org/info/rfc6838>.

[RFC6943] Thaler, D., Ed., "Issues in Identifier Comparison for Security Purposes", RFC 6943, DOI 10.17487/RFC6943, May 2013, <https://www.rfc-editor.org/info/rfc6943>.

[RFC7049] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", RFC 7049, DOI 10.17487/RFC7049, October 2013, <https://www.rfc-editor.org/info/rfc7049>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[RFC8288] Nottingham, M., "Web Linking", RFC 8288, DOI 10.17487/RFC8288, October 2017, <https://www.rfc-editor.org/info/rfc8288>.

[UNICODE] The Unicode Consortium, "The Unicode Standard", <http://www.unicode.org/versions/latest/>.

Note that this reference is to the latest version of Unicode, rather than to a specific release. It is not expected that future changes in the Unicode specification will have any impact on this document.

[UNICODE-UAX15] The Unicode Consortium, "Unicode Standard Annex #15: Unicode Normalization Forms", <http://unicode.org/reports/tr15/>.

[UNICODE-UAX31] The Unicode Consortium, "Unicode Standard Annex #31: Unicode Identifier and Pattern Syntax", <http://unicode.org/reports/tr31/>.

[UNICODE-UAX36] The Unicode Consortium, "Unicode Standard Annex #36: Unicode Security Considerations", <http://unicode.org/reports/tr36/>.

9.2. Informative References

[FOAF] Brickley, D. and L. Miller, "FOAF Vocabulary Specification 0.99", January 2014, <http://xmlns.com/foaf/spec/20140114.html>.

[I-D.ietf-core-links-json] Li, K., Rahman, A., and C. Bormann, "Representing Constrained RESTful Environments (CoRE) Link Format in JSON and CBOR", draft-ietf-core-links-json-10 (work in progress), February 2018.

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

[RFC5789] Dusseault, L. and J. Snell, "PATCH Method for HTTP", RFC 5789, DOI 10.17487/RFC5789, March 2010, <https://www.rfc-editor.org/info/rfc5789>.

[RFC5952] Kawamura, S. and M. Kawashima, "A Recommendation for IPv6 Address Text Representation", RFC 5952, DOI 10.17487/RFC5952, August 2010, <https://www.rfc-editor.org/info/rfc5952>.

[RFC6573] Amundsen, M., "The Item and Collection Link Relations", RFC 6573, DOI 10.17487/RFC6573, April 2012, <https://www.rfc-editor.org/info/rfc6573>.

[RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, DOI 10.17487/RFC6690, August 2012, <https://www.rfc-editor.org/info/rfc6690>.

[RFC6903] Snell, J., "Additional Link Relation Types", RFC 6903, DOI 10.17487/RFC6903, March 2013, <https://www.rfc-editor.org/info/rfc6903>.

[RFC6906] Wilde, E., "The 'profile' Link Relation Type", RFC 6906, DOI 10.17487/RFC6906, March 2013, <https://www.rfc-editor.org/info/rfc6906>.

[RFC7228] Bormann, C., Ersue, M., and A. Keranen, "Terminology for Constrained-Node Networks", RFC 7228, DOI 10.17487/RFC7228, May 2014, <https://www.rfc-editor.org/info/rfc7228>.

[RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <https://www.rfc-editor.org/info/rfc7230>.

[RFC7231] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", RFC 7231, DOI 10.17487/RFC7231, June 2014, <https://www.rfc-editor.org/info/rfc7231>.

[RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <https://www.rfc-editor.org/info/rfc7252>.

[RFC7320] Nottingham, M., "URI Design and Ownership", BCP 190, RFC 7320, DOI 10.17487/RFC7320, July 2014, <https://www.rfc-editor.org/info/rfc7320>.

[RFC8132] van der Stok, P., Bormann, C., and A. Sehgal, "PATCH and FETCH Methods for the Constrained Application Protocol (CoAP)", RFC 8132, DOI 10.17487/RFC8132, April 2017, <https://www.rfc-editor.org/info/rfc8132>.

[W3C.REC-html52-20171214]
Faulkner, S., Eicholz, A., Leithead, T., Danilo, A., and S. Moon, "HTML 5.2", World Wide Web Consortium Recommendation REC-html52-20171214, December 2017, <https://www.w3.org/TR/2017/REC-html52-20171214>.

[W3C.REC-rdf11-concepts-20140225]
Cyganiak, R., Wood, D., and M. Lanthaler, "RDF 1.1 Concepts and Abstract Syntax", World Wide Web Consortium Recommendation REC-rdf11-concepts-20140225, February 2014, <http://www.w3.org/TR/2014/REC-rdf11-concepts-20140225>.

[W3C.REC-turtle-20140225]
Prud'hommeaux, E. and G. Carothers, "RDF 1.1 Turtle", World Wide Web Consortium Recommendation REC-turtle-20140225, February 2014, <http://www.w3.org/TR/2014/REC-turtle-20140225>.

[W3C.REC-webarch-20041215]
Jacobs, I. and N. Walsh, "Architecture of the World Wide Web, Volume One", World Wide Web Consortium Recommendation REC-webarch-20041215, December 2004, <http://www.w3.org/TR/2004/REC-webarch-20041215>.

Appendix A. Core Vocabulary

This section defines the core vocabulary for CoRAL. It is RECOMMENDED that all CoRAL registries assign an unsigned integer to each of these link relation types, form relation types, and form field names.

[[NOTE TO RFC EDITOR: Please replace all occurrences of "urn:ietf:rfc:XXXX#" with a RFC-Editor-controlled IRI.]]

A.1. Link Relation Types

<http://www.iana.org/assignments/relation/type>
  Indicates that the link's context is an instance of the type specified as the link's target; see Section 6 of RFC 6903 [RFC6903].

This link relation type serves in CoRAL the same purpose as the RDF predicate identified by the IRI <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>.

<http://www.iana.org/assignments/relation/item>
Indicates that the link's context is a collection and that the link's target is a member of that collection; see Section 2.1 of RFC 6573 [RFC6573].

<http://www.iana.org/assignments/relation/collection>
Indicates that the link's target is a collection and that the link's context is a member of that collection; see Section 2.2 of RFC 6573 [RFC6573].

A.2. Form Relation Types

<urn:ietf:rfc:XXXX#create>
Indicates that the form's context is a collection and that a new item can be created in that collection by submitting a suitable representation. This form relation type is typically used with the POST method [RFC7231] [RFC7252].

<urn:ietf:rfc:XXXX#update>
Indicates that the form's context can be updated by submitting a suitable representation. This form relation type is typically used with the PUT method [RFC7231] [RFC7252], PATCH method [RFC5789] [RFC8132], or iPATCH method [RFC8132].

<urn:ietf:rfc:XXXX#delete>
Indicates that the form's context can be deleted. This form relation type is typically used with the DELETE method [RFC7231] [RFC7252].

<urn:ietf:rfc:XXXX#search>
Indicates that the form's context can be searched by submitting a search query. This form relation type is typically used with the POST method [RFC7231] [RFC7252] or FETCH method [RFC8132].

A.3. Form Field Names

<urn:ietf:rfc:XXXX#accept>
Specifies an acceptable HTTP content type or CoAP content format for the request payload. There MAY be multiple form fields with this name. If a form does not include a form field with this name, the server accepts any or no request payload, depending on the form relation type.

For HTTP, the content type MUST be specified as a text string in the format defined in Section 3.1.1.1 of RFC 7231 [RFC7231]; the set of possible values is maintained in the IANA Media Types Registry. For CoAP, the content format MUST be specified as an unsigned integer; the set of possible values is maintained in the IANA CoAP Content-Formats Registry.

Appendix B. Default Profile

This section defines a default registry that is assumed when a CoRAL media type without a "profile" parameter is used.

Link Relation Types

0 = <http://www.iana.org/assignments/relation/type>
1 = <http://www.iana.org/assignments/relation/item>
2 = <http://www.iana.org/assignments/relation/collection>

Form Relation Types

0 = <urn:ietf:rfc:XXXX#create>
1 = <urn:ietf:rfc:XXXX#update>
2 = <urn:ietf:rfc:XXXX#delete>
3 = <urn:ietf:rfc:XXXX#search>

Form Fields

0 = <urn:ietf:rfc:XXXX#accept>

Appendix C. CBOR-encoded IRI References

URI references [RFC3986] and, secondarily, IRI references [RFC3987] and are the most common usage of resource identifiers in hypertext representation formats such as HTML 5 [W3C.REC-html52-20171214] and the CoRE Link Format [RFC6690]. They encode the components of a resource identifier either as an absolute URI/IRI or as a relative reference that is resolved against a base URI/IRI.

URI and IRI references are sequences of characters chosen from limited subsets of the repertoires of US-ASCII characters and Unicode characters, respectively. The individual components of a URI or IRI are delimited by several reserved characters, which necessitates the use of percent-encoding for reserved characters in a non-delimiting function. The resolution of references involves parsing URI/IRI references into their components, combining the components with those of the base URI/IRI, merging paths, removing dot segments, and recomposing the result into a URI/IRI reference string.

Altogether, proper processing of URIs is quite complex. This can be a problem in particular in constrained environments [RFC7228] with severe code size limitations. As a result, many implementations in these environments choose to implement only an ad-hoc, informally-specified, bug-ridden, non-interoperable subset of half of RFC 3986.

This section specifies CBOR-encoded IRI References, a serialization format for IRI references that encodes the IRI components as CBOR data items rather than text. Assuming that a CBOR implementation is already present, typical operations on CBOR-encoded IRI references such as parsing, reference resolution, and comparison can be implemented much more easily than with the text-based format. A full implementation that covers all corner cases of the specification can be implemented in a relatively small amount of code.

CBOR-encoded IRI References are not capable of expressing all IRI references permitted by the syntax of RFC 3987 [RFC3987]. The supported subset covers all CoAP URIs [RFC7252] and most HTTP URIs [RFC7230].

C.1. Data Structure

The encoding is very similar to the encoding of the request URI in CoAP messages [RFC7252]. The components of an IRI reference are encoded as a sequence of _options_. Each option consists of an _option number_ identifying the type of option (scheme, host name, etc.) and the _option value_.

```
iri = [?(scheme:    1, text),
       ?(host.name: 2, text //
         host.ip:   3, bytes .size 4 / bytes .size 16),
       ?(port:      4, uint .size 2),
       ?(path.type: 5, path-type),
       *(path:      6, text),
       *(query:     7, text),
       ?(fragment:  8, text)]

path-type = &(absolute-path:   0,
              append-path:     1,
              relative-path:   2,
              append-relation: 3)
```

C.2. Options

The following options are defined:

scheme

Specifies the IRI scheme. IRI schemes have the same syntax as URI schemes. The option value therefore MUST match the "scheme" rule defined in Section 3.1 of RFC 3986.

host.name
: Specifies the host of the IRI authority as a registered name.

host.ip
: Specifies the host of the IRI authority as an IPv4 address (4 bytes) or an IPv6 address (16 bytes).

port
: Specifies the port number. The option value MUST be an unsigned integer in the range 0 to 65535 (inclusive).

path.type
: Specifies the type of the IRI path for reference resolution. Possible values are 0 (absolute-path), 1 (append-path), 2 (relative-path), and 3 (append-relation).

path
: Specifies one segment of the IRI path. This option can occur more than once.

query
: Specifies one argument of the IRI query. This option can occur more than once.

fragment
: Specifies the fragment identifier.

The value of a "host.name", "path", "query", and "fragment" option can be any Unicode string. No percent-encoding is performed.

C.3. Properties

A sequence of options is considered _well-formed_ if:

- the sequence of options is empty or starts with a "scheme", "host.name", "host.ip", "port", "path.type", "path", "query", or "fragment" option;

- a "scheme" option is followed by either a "host.name" or a "host.ip" option;

- a "host.name" option is followed by a "port" option;

- a "host.ip" option is followed by a "port" option;

- a "port" option is followed by a "path", "query", or "fragment" option or is at the end of the sequence;

- a "path.type" option is followed by a "path", "query", or "fragment" option or is at the end of the sequence;

- a "path" option is followed by a "path", "query", or "fragment" option or is at the end of the sequence;

- a "query" option is followed by a "query" or "fragment" option or is at the end of the sequence; and

- a "fragment" option is at the end of the sequence.

A well-formed sequence of options is considered _absolute_ if the sequence of options starts with a "scheme" option. A well-formed sequence of options is considered _relative_ if the sequence of options is empty or starts with an option other than the "scheme" option.

An absolute sequence of options is considered _normalized_ if the result of resolving the sequence of options against any base IRI reference is equal to the input. (It doesn't matter what it is resolved against, since it is already absolute.)

C.4. Reference Resolution

This section defines how to resolve a CBOR-encoded IRI reference that might be relative to a given base IRI.

Applications MUST resolve a well-formed sequence of options `href` against an absolute sequence of options `base` by using an algorithm that is functionally equivalent to the following Python 3.5 code.

<CODE BEGINS>

```
def resolve(base, href, relation=None):
  if not is_absolute(base) or not is_well_formed(href):
    return None
  result = []
  type = PathType.RELATIVE_PATH
  (option, value) = href[0]
  if option == Option.HOST_IP:
    option = Option.HOST_NAME
  elif option == Option.PATH_TYPE:
    href = href[1:]
    type = value
    option = Option.PATH
```

```
        if option != Option.PATH or type == PathType.ABSOLUTE_PATH:
          _copy_until(base, result, option)
        else:
          _copy_until(base, result, Option.QUERY)
          if type == PathType.APPEND_RELATION:
            _append_and_normalize(result, Option.PATH,
                                  format(relation, "x"))
            return result
          if type == PathType.RELATIVE_PATH:
            _remove_last_path_segment(result)
        _copy_until(href, result, Option.END)
        _append_and_normalize(href, Option.END, None)
        return result

      def _copy_until(input, output, end):
        for (option, value) in input:
          if option >= end:
            break
          _append_and_normalize(output, option, value)

      def _append_and_normalize(output, option, value):
        if option == Option.PATH:
          if value == ".":
            return
          if value == "..":
            _remove_last_path_segment(output)
            return
        elif option > Option.PATH:
          if len(output) >= 2 and                                      \
              output[-1] == (Option.PATH, "") and                      \
              (output[-2][0] < Option.PATH_TYPE or                     \
              output[-2] == (Option.PATH_TYPE, PathType.ABSOLUTE_PATH)):
            _remove_last_path_segment(output)
          if option >= Option.END:
            return
        output.append((option, value))

      def _remove_last_path_segment(output):
        if len(output) >= 1 and output[-1][0] == Option.PATH:
          del output[-1]

   <CODE ENDS>
```

C.5. IRI Recomposition

This section defines how to recompose an IRI from a sequence of options that encodes an absolute IRI reference.

Applications MUST recompose an IRI from a sequence of options by using an algorithm that is functionally equivalent to the following Python 3.5 code.

To reduce variability, the hexadecimal notation when percent-encoding octets SHOULD use uppercase letters. The text representation of IPv6 addresses SHOULD follow the recommendations in Section 4 of RFC 5952 [RFC5952].

<CODE BEGINS>

```
def recompose(href):
  if not is_absolute(href):
    return None
  result = ""
  no_path = True
  first_query = True
  for (option, value) in href:
    if option == Option.SCHEME:
      result += value + ":"
    elif option == Option.HOST_NAME:
      result += "//" + _encode_ireg_name(value)
    elif option == Option.HOST_IP:
      result += "//" + _encode_ip_address(value)
    elif option == Option.PORT:
      result += ":" + str(value)
    elif option == Option.PATH:
      result += "/" + _encode_path_segment(value)
      no_path = False
    elif option == Option.QUERY:
      if no_path:
        result += "/"
        no_path = False
      result += "?" if first_query else "&"
      result += _encode_query_argument(value)
      first_query = False
    elif option == Option.FRAGMENT:
      if no_path:
        result += "/"
        no_path = False
      result += "#" + _encode_fragment(value)
  if no_path:
    result += "/"
    no_path = False
  return result

def _encode_ireg_name(s):
  return "".join(c if _is_ireg_name_char(c) else
```

```
             _encode_pct(c) for c in s)

       def _encode_ip_address(b):
         if len(b) == 4:
           return ".".join(str(c) for c in b)
         elif len(b) == 16:
           return "[" + ... + "]"  # see RFC 5952

       def _encode_path_segment(s):
         return "".join(c if _is_isegment_char(c) else
             _encode_pct(c) for c in s)

       def _encode_query_argument(s):
         return "".join(c if _is_iquery_char(c) and c != "&" else
             _encode_pct(c) for c in s)

       def _encode_fragment(s):
         return "".join(c if _is_ifragment_char(c) else
             _encode_pct(c) for c in s)

       def _encode_pct(s):
         return "".join(
             "%{0:0>2X}".format(c) for c in s.encode("utf-8"))

       def _is_ireg_name_char(c):
         return _is_iunreserved(c) or _is_sub_delim(c)

       def _is_isegment_char(c):
         return _is_ipchar(c)

       def _is_iquery_char(c):
         return _is_ipchar(c) or _is_iprivate(c) or c == "/" or c == "?"

       def _is_ifragment_char(c):
         return _is_ipchar(c) or c == "/" or c == "?"

       def _is_ipchar(c):
         return _is_iunreserved(c) or _is_sub_delim(c) or              \
                c == ":" or c == "@"

       def _is_iunreserved(c):
         return _is_alpha(c) or _is_digit(c) or                        \
                c == "-" or c == "." or c == "_" or c == "~" or        \
                _is_ucschar(c)

       def _is_alpha(c):
         return c >= "A" and c <= "Z" or c >= "a" and c <= "z"
```

```
      def _is_digit(c):
        return c >= "0" and c <= "9"

      def _is_sub_delim(c):
        return c == "!" or c == "$" or c == "&" or c == "'" or          \
               c == "(" or c == ")" or c == "*" or c == "+" or          \
               c == "," or c == ";" or c == "="

      def _is_ucschar(c):
        return c >= "\U000000A0" and c <= "\U0000D7FF" or               \
               c >= "\U0000F900" and c <= "\U0000FDCF" or               \
               c >= "\U0000FDF0" and c <= "\U0000FFEF" or               \
               c >= "\U00010000" and c <= "\U0001FFFD" or               \
               c >= "\U00020000" and c <= "\U0002FFFD" or               \
               c >= "\U00030000" and c <= "\U0003FFFD" or               \
               c >= "\U00040000" and c <= "\U0004FFFD" or               \
               c >= "\U00050000" and c <= "\U0005FFFD" or               \
               c >= "\U00060000" and c <= "\U0006FFFD" or               \
               c >= "\U00070000" and c <= "\U0007FFFD" or               \
               c >= "\U00080000" and c <= "\U0008FFFD" or               \
               c >= "\U00090000" and c <= "\U0009FFFD" or               \
               c >= "\U000A0000" and c <= "\U000AFFFD" or               \
               c >= "\U000B0000" and c <= "\U000BFFFD" or               \
               c >= "\U000C0000" and c <= "\U000CFFFD" or               \
               c >= "\U000D0000" and c <= "\U000DFFFD" or               \
               c >= "\U000E1000" and c <= "\U000EFFFD"

      def _is_iprivate(c):
        return c >= "\U0000E000" and c <= "\U0000F8FF" or               \
               c >= "\U000F0000" and c <= "\U000FFFFD" or               \
               c >= "\U00100000" and c <= "\U0010FFFD"
```

<CODE ENDS>

C.6. CoAP Encoding

This section defines how to construct CoAP options from an absolute, normalized, CBOR-encoded IRI Reference.

Applications MUST construct CoAP options by recomposing the sequence of options to an IRI (Appendix C.5 of this document), mapping the IRI to an URI (Section 3.1 of RFC 3987), and decomposing the URI into CoAP options (Section 6.4 of RFC 7252).

The following illustrative Python 3.5 code is roughly equivalent to this.

<CODE BEGINS>

```
def coap(href, to_proxy=False):
  if not is_absolute(href):
    return None
  result = b""
  previous = 0
  for (option, value) in href:
    if option == Option.SCHEME:
      pass
    elif option == Option.HOST_NAME:
      opt = 3  # Uri-Host
      val = value.encode("utf-8")
      result += _encode_coap_option(opt - previous, val)
      previous = opt
    elif option == Option.HOST_IP:
      opt = 3  # Uri-Host
      if len(value) == 4:
        val = ".".join(str(c) for c in b).encode("utf-8")
      elif len(value) == 16:
        val = b"[" + ... + b"]"  # see RFC 5952
      result += _encode_coap_option(opt - previous, val)
      previous = opt
    elif option == Option.PORT:
      opt = 7  # Uri-Port
      val = value.to_bytes((value.bit_length() + 7) // 8, "big")
      result += _encode_coap_option(opt - previous, val)
      previous = opt
    elif option == Option.PATH:
      opt = 11  # Uri-Path
      val = value.encode("utf-8")
      result += _encode_coap_option(opt - previous, val)
      previous = opt
    elif option == Option.QUERY:
      opt = 15  # Uri-Query
      val = value.encode("utf-8")
      result += _encode_coap_option(opt - previous, val)
      previous = opt
    elif option == Option.FRAGMENT:
      pass
  if to_proxy:
    (option, value) = href[0]
    opt = 39  # Proxy-Scheme
    val = value.encode("utf-8")
    result += _encode_coap_option(opt - previous, val)
    previous = opt
  return result

def _encode_coap_option(delta, value):
  length = len(value)
```

```
        delta_nibble = _encode_coap_option_nibble(delta)
        length_nibble = _encode_coap_option_nibble(length)
        result = bytes([delta_nibble << 4 | length_nibble])
        if delta_nibble == 13:
          delta -= 13
          result += bytes([delta])
        elif delta_nibble == 14:
          delta -= 256 + 13
          result += bytes([delta >> 8, delta & 255])
        if length_nibble == 13:
          length -= 13
          result += bytes([length])
        elif length_nibble == 14:
          length -= 256 + 13
          result += bytes([length >> 8, length & 255])
        result += value
        return result

      def _encode_coap_option_nibble(n):
        if n < 13:
          return n
        elif n < 256 + 13:
          return 13
        elif n < 65536 + 256 + 13:
          return 14

   <CODE ENDS>
```

Author's Address

Klaus Hartke
Universitaet Bremen TZI
Postfach 330440
Bremen D-28359
Germany

Phone: +49-421-218-63905
Email: hartke@tzi.org

Network Working Group O. Garcia-Morchon
Internet-Draft Philips IP&S
Intended status: Informational S. Kumar
Expires: November 20, 2018 Philips Research
M. Sethi
Ericsson
May 19, 2018

State-of-the-Art and Challenges for the Internet of Things Security
draft-irtf-t2trg-iot-seccons-15

Abstract

The Internet of Things (IoT) concept refers to the usage of standard Internet protocols to allow for human-to-thing and thing-to-thing communication. The security needs for IoT systems are well-recognized and many standardization steps to provide security have been taken, for example, the specification of Constrained Application Protocol (CoAP) secured with Datagram Transport Layer Security (DTLS). However, security challenges still exist, not only because there are some use cases that lack a suitable solution, but also because many IoT devices and systems have been designed and deployed with very limited security capabilities. In this document, we first discuss the various stages in the lifecycle of a thing. Next, we document the security threats to a thing and the challenges that one might face to protect against these threats. Lastly, we discuss the next steps needed to facilitate the deployment of secure IoT systems. This document can be used by implementors and authors of IoT specifications as a reference for details about security considerations while documenting their specific security challenges, threat models, and mitigations.

This document is a product of the IRTF Thing-to-Thing Research Group (T2TRG).

Status of This Memo

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 November 20, 2018.

Copyright Notice

Table of Contents

1. Introduction

The Internet of Things (IoT) denotes the interconnection of highly heterogeneous networked entities and networks that follow a number of different communication patterns such as: human-to-human (H2H), human-to-thing (H2T), thing-to-thing (T2T), or thing-to-things (T2Ts). The term IoT was first coined by the Auto-ID center [AUTO-ID] in 1999 which had envisioned a world where every physical object is tagged with a radio-frequency identification (RFID) tag having a globally unique identifier. This would not only allow tracking of objects in real-time but also allow querying of data about them over the Internet. However, since then, the meaning of the Internet of Things has expanded and now encompasses a wide variety of technologies, objects and protocols. It is not surprising that the IoT has received significant attention from the research community to (re)design, apply, and use standard Internet technology and protocols for the IoT.

The things that are part of the Internet of Things are computing devices that understand and react to the environment they reside in. These things are also often referred to as smart objects or smart devices. The introduction of IPv6 [RFC6568] and CoAP [RFC7252] as fundamental building blocks for IoT applications allows connecting IoT hosts to the Internet. This brings several advantages including: (i) a homogeneous protocol ecosystem that allows simple integration with other Internet hosts; (ii) simplified development for devices that significantly vary in their capabilities; (iii) a unified interface for applications, removing the need for application-level proxies. These building blocks greatly simplify the deployment of the envisioned scenarios which range from building automation to production environments and personal area networks.

This document presents an overview of important security aspects for the Internet of Things. We begin by discussing the lifecycle of a thing in Section 2. In Section 3, we discuss security threats for the IoT and methodologies for managing these threats when designing a secure system. Section 4 reviews existing IP-based (security) protocols for the IoT and briefly summarizes existing guidelines and regulations. Section 5 identifies remaining challenges for a secure IoT and discusses potential solutions. Section 6 includes final remarks and conclusions. This document can be used by IoT standards specifications as a reference for details about security considerations applying to the specified system or protocol.

The first draft version of this document was submitted in March 2011. Initial draft versions of this document were presented and discussed during the CORE meetings at IETF 80 and later. Discussions on security lifecycle at IETF 92 (March 2015) evolved into more general security considerations. Thus, the draft was selected to address the T2TRG work item on the security considerations and challenges for the Internet of Things. Further updates of the draft were presented and discussed during the T2TRG meetings at IETF 96 (July 2016) and IETF 97 (November 2016) and at the joint interim in Amsterdam (March 2017). This document has been reviewed by, commented on, and discussed extensively for a period of nearly six years by a vast majority of T2TRG and related group members; the number of which certainly exceeds 100 individuals. It is the consensus of T2TRG that the security considerations described in this document should be published in the IRTF Stream of the RFC series. This document does not constitute a standard.

2. The Thing Lifecycle

The lifecycle of a thing refers to the operational phases of a thing in the context of a given application or use case. Figure 1 shows the generic phases of the lifecycle of a thing. This generic lifecycle is applicable to very different IoT applications and scenarios. For instance, [RFC7744] provides an overview of relevant IoT use cases.

In this document, we consider a Building Automation and Control (BAC) system to illustrate the lifecycle and the meaning of these different phases. A BAC system consists of a network of interconnected nodes that performs various functions in the domains of HVAC (Heating, Ventilating, and Air Conditioning), lighting, safety, etc. The nodes vary in functionality and a large majority of them represent resource-constrained devices such as sensors and luminaries. Some devices may be battery operated or may rely on energy harvesting. This requires us to also consider devices that sleep during their operation to save energy. In our BAC scenario, the life of a thing starts when it is manufactured. Due to the different application areas (i.e., HVAC, lighting, or safety) nodes/things are tailored to a specific task. It is therefore unlikely that one single manufacturer will create all nodes in a building. Hence, interoperability as well as trust bootstrapping between nodes of different vendors is important.

The thing is later installed and commissioned within a network by an installer during the bootstrapping phase. Specifically, the device identity and the secret keys used during normal operation may be provided to the device during this phase. Different subcontractors may install different IoT devices for different purposes. Furthermore, the installation and bootstrapping procedures may not be a discrete event and may stretch over an extended period. After being bootstrapped, the device and the system of things are in operational mode and execute the functions of the BAC system. During this operational phase, the device is under the control of the system owner and used by multiple system users. For devices with lifetimes spanning several years, occasional maintenance cycles may be required. During each maintenance phase, the software on the device can be upgraded or applications running on the device can be reconfigured. The maintenance tasks can be performed either locally or from a backend system. Depending on the operational changes to the device, it may be required to re-bootstrap at the end of a maintenance cycle. The device continues to loop through the operational phase and the eventual maintenance phases until the device is decommissioned at the end of its lifecycle. However, the end-of-life of a device does not necessarily mean that it is defective and rather denotes a need to replace and upgrade the network to the next-generation devices for additional functionality. Therefore, the device can be removed and re-commissioned to be used in a different system under a different owner thereby starting the lifecycle all over again.

We note that the presented lifecycle represents to some extent a simplified model. For instance, it is possible to argue that the lifecycle does not start when a tangible device is manufactured but rather when the oldest bit of code that ends up in the device - maybe from an open source project or from the used operating system - was written. Similarly, the lifecycle could also include an on-the-shelf phase where the device is in the supply-chain before an owner/user purchases and installs it. Another phase could involve the device being re-badged by some vendor who is not the original manufacturer. Such phases can significantly complicate other phases such as maintenance and bootstrapping. Finally, other potential end-states can be, e.g., a vendor that no longer supports a device type because it is at end-of-life or a situation in which a device is simply forgotten but remains functional.

```
    _Manufactured           _SW update          _Decommissioned
   /                       /                   /
   |   _Installed          |   _ Application   |   _Removed &
   |  /                    |  / reconfigured   |  /  replaced
   |  |   _Commissioned    |  |                |  |
   |  |  /                 |  |                |  |   _Reownership &
   |  |  |    _Application |  |   _Application |  |  / recommissioned
   |  |  |   /   running   |  |  / running     |  |  |
   |  |  |   |             |  |  |             |  |  |             \\
   +##+##+###+#############+##+##+#############+##+##+##############>>>
       \/  \______________/ \/  \_____________/ \___/         time //
       /           /         \          \          \
   Bootstrapping  /      Maintenance &   \     Maintenance &
                 /      re-bootstrapping  \   re-bootstrapping
           Operational                Operational
```

Figure 1: The lifecycle of a thing in the Internet of Things

Security is a key requirement in any communication system. However, security is an even more critical requirement in real-world IoT deployments for several reasons. First, compromised IoT systems can not only endanger the privacy and security of a user, but can also cause physical harm. This is because IoT systems often comprise sensors, actuators and other connected devices in the physical environment of the user which could adversely affect the user if they are compromised. Second, a vulnerable IoT system means that an attacker can alter the functionality of a device from a given manufacturer. This not only affects the manufacturer's brand image, but can also leak information that is very valuable for the manufacturer (such as proprietary algorithms). Third, the impact of attacking an IoT system goes beyond a specific device or an isolated system since compromised IoT systems can be misused at scale. For example, they may be used to perform a Distributed Denial of Service (DDoS) attack that limits the availability of other networks and services. The fact that many IoT systems rely on standard IP protocols allows for easier system integration, but this also makes attacks on standard IP protocols widely applicable in other environments. This results in new requirements regarding the implementation of security.

The term security subsumes a wide range of primitives, protocols, and procedures. Firstly, it includes the basic provision of security services that include confidentiality, authentication, integrity, authorization, source authentication, and availability along with some augmented services, such as duplicate detection and detection of stale packets (timeliness). These security services can be implemented by means of a combination of cryptographic mechanisms, such as block ciphers, hash functions, or signature algorithms, and non-cryptographic mechanisms, which implement authorization and other security policy enforcement aspects. For ensuring security in IoT networks, we should not only focus on the required security services, but also pay special attention to how these services are realized in the overall system and how the security functionalities are executed in practice.

3. Security Threats and Managing Risk

Security threats in related IP protocols have been analyzed in multiple documents including Hypertext Transfer Protocol (HTTP) over Transport Layer Security (TLS) (HTTPS) [RFC2818], Constrained Application Protocol (COAP) [RFC7252], IPv6 over Low-Power Wireless Personal Area Networks (6LoWPAN) [RFC4919], Access Node Control Protocol (ANCP) [RFC5713], Domain Name System (DNS) [RFC3833], IPv6 Neighbor Discovery (ND) [RFC3756], and Protocol for Carrying Authentication and Network Access (PANA) [RFC4016]. In this section, we specifically discuss the threats that could compromise an individual thing or the network as a whole. Some of these threats might go beyond the scope of Internet protocols but we gather them here for the sake of completeness. The threats in the following list are not in any particular order and some threats might be more critical than others depending on the deployment scenario under consideration:

1. Vulnerable Software/Code: Things in the Internet of Things rely on software that might contain severe bugs and/or bad design choices. This makes the things vulnerable to many different types of attacks, depending on the criticality of the bugs, e.g., buffer overflows or lack of authentication. This can be considered as one of the most important security threat. The large-scale distributed denial-of-service (DDoS) attack, popularly known as the Mirai botnet [mirai], was caused by things that had well-known or easy-to-guess passwords for configuration.

2. Privacy threat: The tracking of a thing's location and usage may pose a privacy risk to people around it. For instance, an attacker can infer privacy sensitive information from the data gathered and communicated by individual things. Such information may subsequently be sold to interested parties for marketing purposes and targeted advertising. In extreme cases, such information might be used to track dissidents in oppressive regimes. Unlawful surveillance and interception of traffic to/from a thing by intelligence agencies is also a privacy threat.

3. Cloning of things: During the manufacturing process of a thing, an untrusted factory can easily clone the physical

characteristics, firmware/software, or security configuration of the thing. Deployed things might also be compromised and their software reverse engineered allowing for cloning or software modifications. Such a cloned thing may be sold at a cheaper price in the market, and yet can function normally as a genuine thing. For example, two cloned devices can still be associated and work with each other. In the worst-case scenario, a cloned device can be used to control a genuine device or perform an attack. One should note here, that an untrusted factory may also change functionality of the cloned thing, resulting in degraded functionality with respect to the genuine thing (thereby, inflicting potential damage to the reputation of the original thing manufacturer). Moreover, additional functionality can be introduced in the cloned thing. An example of such functionality is a backdoor.

4. Malicious substitution of things: During the installation of a thing, a genuine thing may be substituted with a similar variant (of lower quality) without being detected. The main motivation may be cost savings, where the installation of lower-quality things (for example, non-certified products) may significantly reduce the installation and operational costs. The installers can subsequently resell the genuine things to gain further financial benefits. Another motivation may be to inflict damage to the reputation of a competitor's offerings.

5. Eavesdropping attack: During the commissioning of a thing into a network, it may be susceptible to eavesdropping, especially if operational keying materials, security parameters, or configuration settings, are exchanged in clear using a wireless medium or if used cryptographic algorithms are not suitable for the envisioned lifetime of the device and the system. After obtaining the keying material, the attacker might be able to recover the secret keys established between the communicating entities, thereby compromising the authenticity and confidentiality of the communication channel, as well as the authenticity of commands and other traffic exchanged over this communication channel. When the network is in operation, T2T communication can be eavesdropped if the communication channel is not sufficiently protected or if a session key is compromised due to protocol weaknesses. An adversary may also be able to eavesdrop if keys are not renewed or updated appropriately. Lastly, messages can also be recorded and decrypted offline at a later point of time. The Venona project [venona-project] is one such example where messages were recorded for offline decryption.

6. Man-in-the-middle attack: Both the commissioning phase and operational phases may also be vulnerable to man-in-the-middle attacks. For example, when keying material between communicating entities is exchanged in the clear and the security of the key establishment protocol depends on the tacit assumption that no third party can eavesdrop during the execution of this protocol. Additionally, device authentication or device authorization may be non-trivial, or may need support of a human decision process, since things usually do not have a-priori knowledge about each other and cannot always differentiate friends and foes via completely automated mechanisms.

7. Firmware attacks: When a thing is in operation or maintenance phase, its firmware or software may be updated to allow for new functionality or new features. An attacker may be able to exploit such a firmware upgrade by maliciously replacing the thing's firware, thereby influencing its operational behavior. For example, an attacker could add a piece of malicious code to the firmware that will cause it to periodically report the energy usage of the thing to a data repository for analysis. The attacker can then use this information to determine when a home or enterprise (where the thing is installed) is unoccupied and break in. Similarly, devices whose software has not been properly maintained and updated might contain vulnerabilities that might be exploited by attackers to replace the firmware on the device.

8. Extraction of private information: IoT devices (such as sensors, actuators, etc.) are often physically unprotected in their ambient environment and they could easily be captured by an attacker. An attacker with physical access may then attempt to extract private information such as keys (for example, device's key, private-key, group key), sensed data (for example, healthcare status of a user), configuration parameters (for example, the Wi-Fi key), or proprietary algorithms (for example, algorithm performing some data analytics task). Even when the data originating from a thing is encrypted, attackers can perform traffic analysis to deduce meaningful information which might compromise the privacy of the thing's owner and/or user.

9. Routing attack: As highlighted in [ID-Daniel], routing information in IoT networks can be spoofed, altered, or replayed, in order to create routing loops, attract/repel network traffic, extend/shorten source routes, etc. A non-exhaustive list of routing attacks includes 1) Sinkhole attack (or blackhole attack), where an attacker declares himself to have a high-quality route/path to the base station, thus

allowing him to do manipulate all packets passing through it. 2) Selective forwarding, where an attacker may selectively forward packets or simply drop a packet. 3) Wormhole attack, where an attacker may record packets at one location in the network and tunnel them to another location, thereby influencing perceived network behavior and potentially distorting statistics, thus greatly impacting the functionality of routing. 4) Sybil attack, whereby an attacker presents multiple identities to other things in the network. We refer to [ID-Daniel] for further router attacks and a more detailed description.

10. Elevation of privilege: An attacker with low privileges can misuse additional flaws in the implemented authentication and authorization mechanisms of a thing to gain more privileged access to the thing and its data.

11. Denial-of-Service (DoS) attack: Often things have very limited memory and computation capabilities. Therefore, they are vulnerable to resource exhaustion attack. Attackers can continuously send requests to specific things so as to deplete their resources. This is especially dangerous in the Internet of Things since an attacker might be located in the backend and target resource-constrained devices that are part of a constrained node network [RFC7228]. DoS attack can also be launched by physically jamming the communication channel. Network availability can also be disrupted by flooding the network with a large number of packets. On the other hand, things compromised by attackers can be used to disrupt the operation of other networks or systems by means of a Distributed DoS (DDoS) attack.

To deal with above threats it is required to find and apply suitable security mitigations. However, new threats and exploits appear on a daily basis and products are deployed in different environments prone to different types of threats. Thus, ensuring a proper level of security in an IoT system at any point of time is challenging. To address this challenge, some of the following methodologies can be used:

1. A Business Impact Analysis (BIA) assesses the consequences of the loss of basic security attributes: confidentiality, integrity and availability in an IoT system. These consequences might include the impact from lost data, reduced sales, increased expenses, regulatory fines, customer dissatisfaction, etc. Performing a business impact analysis allows a business to determine the relevance of having a proper security design.

2. A Risk Assessment (RA) analyzes security threats to an IoT system while considering their likelihood and impact. It also includes categorizing each of them with a risk level. Risks classified as moderate or high must be mitigated, i.e., the security architecture should be able to deal with those threat.

3. A privacy impact assessment (PIA) aims at assessing the Personally Identifiable Information (PII) that is collected, processed, or used in an IoT system. By doing so, the goal is to fulfill applicable legal requirements, determine risks and effects of manipulation and loss of PII.

4. Procedures for incident reporting and mitigation refer to the methodologies that allow becoming aware of any security issues that affect an IoT system. Furthermore, this includes steps towards the actual deployment of patches that mitigate the identified vulnerabilities.

BIA, RA, and PIA should generally be realized during the creation of a new IoT system or when deploying significant system/feature upgrades. In general, it is recommended to re-assess them on a regular basis taking into account new use cases and/or threats. The way a BIA, RA, PIA are performed depends on the environment and the industry. More information can be found in NIST documents such as [NISTSP800-34r1], [NISTSP800-30r1], and [NISTSP800-122].

4. State-of-the-Art

This section is organized as follows. Section 4.1 summarizes state-of-the-art on IP-based IoT systems, within IETF and in other standardization bodies. Section 4.2 summarizes state-of-the-art on IP-based security protocols and their usage. Section 4.3 discusses guidelines and regulations for securing IoT as proposed by other bodies.

4.1. IP-based IoT Protocols and Standards

Nowadays, there exists a multitude of control protocols for IoT. For BAC systems, the ZigBee standard [ZB], BACNet [BACNET], and DALI [DALI] play key roles. Recent trends, however, focus on an all-IP approach for system control.

In this setting, a number of IETF working groups are designing new protocols for resource-constrained networks of smart things. The 6LoWPAN working group [WG-6LoWPAN] for example has defined methods and protocols for the efficient transmission and adaptation of IPv6 packets over IEEE 802.15.4 networks [RFC4944].

The CoRE working group [WG-CoRE] has specified the Constrained Application Protocol (CoAP) [RFC7252]. CoAP is a RESTful protocol for constrained devices that is modeled after HTTP and typically runs over UDP to enable efficient application-level communication for things.

In many smart object networks, the smart objects are dispersed and have intermittent reachability either because of network outages or because they sleep during their operational phase to save energy. In such scenarios, direct discovery of resources hosted on the constrained server might not be possible. To overcome this barrier, the CoRE working group is specifying the concept of a Resource Directory (RD) [ID-rd]. The Resource Directory hosts descriptions of resources which are located on other nodes. These resource descriptions are specified as CoRE link format [RFC6690].

While CoAP defines a standard communication protocol, a format for representing sensor measurements and parameters over CoAP is required. The Sensor Measurement Lists (SenML) [ID-senml] is a specification that defines media types for simple sensor measurements and parameters. It has a minimalistic design so that constrained devices with limited computational capabilities can easily encode their measurements and, at the same time, servers can efficiently collect large number of measurements.

In many IoT deployments, the resource-constrained smart objects are connected to the Internet via a gateway that is directly reachable. For example, an IEEE 802.11 Access Point (AP) typically connects the client devices to the Internet over just one wireless hop. However, some deployments of smart object networks require routing between the smart objects themselves. The IETF has therefore defined the IPv6 Routing Protocol for Low-Power and Lossy Networks (RPL) [RFC6550]. RPL provides support for multipoint-to-point traffic from resource-constrained smart objects towards a more resourceful central control point, as well as point-to-multipoint traffic in the reverse direction. It also supports point-to-point traffic between the resource-constrained devices. A set of routing metrics and constraints for path calculation in RPL are also specified [RFC6551].

The IPv6 over Networks of Resource-constrained Nodes (6lo) [WG-6lo] working group of the IETF has specified how IPv6 packets can be transmitted over various link layer protocols that are commonly employed for resource-constrained smart object networks. There is also ongoing work to specify IPv6 connectivity for a Non-Broadcast Multi-Access (NBMA) mesh network that is formed by IEEE 802.15.4 TimeSlotted Channel Hopping (TSCH} links [ID-6tisch]. Other link layer protocols for which IETF has specified or is currently specifying IPv6 support include Bluetooth [RFC7668], Digital Enhanced

Cordless Telecommunications (DECT) Ultra Low Energy (ULE) air interface [RFC8105], and Near Field Communication (NFC) [ID-6lonfc].

Baker and Meyer [RFC6272] identify which IP protocols can be used in smart grid environments. They give advice to smart grid network designers on how they can decide on a profile of the Internet protocol suite for smart grid networks.

The Low Power Wide-Area Network (LPWAN) working [WG-LPWAN] group is analyzing features, requirements, and solutions to adapt IP-based protocols to networks such as LORA [lora], SigFox [sigfox], NB-IoT [nbiot], etc. These networking technologies enable a smart thing to run for years on a single coin-cell by relying on a star network topology and using optimized radio modulation with frame sizes in the order of tens of bytes. Such networks bring new security challenges since most existing security mechanism do not work well with such resource constraints.

JavaScript Object Notation (JSON) is a lightweight text representation format for structured data [RFC8259]. It is often used for transmitting serialized structured data over the network. IETF has defined specifications for encoding cryptographic keys, encrypted content, signed content, and claims to be transferred between two parties as JSON objects. They are referred to as JSON Web Keys (JWK) [RFC7517], JSON Web Encryption (JWE) [RFC7516], JSON Web Signatures (JWS) [RFC7515] and JSON Web Token (JWT) [RFC7519].

An alternative to JSON, Concise Binary Object Representation (CBOR) [RFC7049] is a concise binary data format that is used for serialization of structured data. It is designed for resource-constrained nodes and therefore it aims to provide a fairly small message size with minimal implementation code, and extensibility without the need for version negotiation. CBOR Object Signing and Encryption (COSE) [RFC8152] specifies how to encode cryptographic keys, message authentication codes, encrypted content, and signatures with CBOR.

The Light-Weight Implementation Guidance (LWIG) working group [WG-LWIG] is collecting experiences from implementers of IP stacks in constrained devices. The working group has already produced documents such as RFC7815 [RFC7815] which defines how a minimal Internet Key Exchange Version 2 (IKEv2) initiator can be implemented.

The Thing-2-Thing Research Group (T2TRG) [RG-T2TRG] is investigating the remaining research issues that need to be addressed to quickly turn the vision of IoT into a reality where resource-constrained nodes can communicate with each other and with other more capable nodes on the Internet.

Additionally, industry alliances and other standardization bodies are creating constrained IP protocol stacks based on the IETF work. Some important examples of this include:

1. Thread [Thread]: Specifies the Thread protocol that is intended for a variety of IoT devices. It is an IPv6-based network protocol that runs over IEEE 802.15.4.

2. Industrial Internet Consortium [IIoT]: The consortium defines reference architectures and security frameworks for development, adoption and widespread use of Industrial Internet technologies based on existing IETF standards.

3. Internet Protocol for Smart Objects IPSO [IPSO]: The alliance specifies a common object model that enables application software on any device to interoperate with other conforming devices.

4. OneM2M [OneM2M]: The standards body defines technical and API specifications for IoT devices. It aims to create a service layer that can run on any IoT device hardware and software.

5. Open Connectivity Foundation (OCF) [OCF]: The foundation develops standards and certifications primarily for IoT devices that use Constrained Application Protocol (CoAP) as the application layer protocol.

6. Fairhair Alliance [Fairhair]: Specifies an IoT middleware to enable a common IP network infrastructure between different application standards used in building automation and lighting systems such as BACnet, KNX and ZigBee.

7. OMA LWM2M [LWM2M]: OMA Lightweight M2M is a standard from the Open Mobile Alliance for M2M and IoT device management. LWM2M relies on CoAP as the application layer protocol and uses a RESTful architecture for remote management of IoT devices.

4.2. Existing IP-based Security Protocols and Solutions

There are three main security objectives for IoT networks: 1. protecting the IoT network from attackers. 2. protecting IoT applications and thus, the things and users. 3. protecting the rest of the Internet and other things from attacks that use compromised things as an attack platform.

In the context of the IP-based IoT deployments, consideration of existing Internet security protocols is important. There are a wide range of specialized as well as general-purpose security solutions for the Internet domain such as IKEv2/IPsec [RFC7296], TLS [RFC5246], DTLS [RFC6347], HIP [RFC7401], PANA [RFC5191], and EAP [RFC3748].

TLS provides security for TCP and requires a reliable transport. DTLS secures and uses datagram-oriented protocols such as UDP. Both protocols are intentionally kept similar and share the same ideology and cipher suites. The CoAP base specification [RFC7252] provides a description of how DTLS can be used for securing CoAP. It proposes three different modes for using DTLS: the PreSharedKey mode, where nodes have pre-provisioned keys for initiating a DTLS session with another node, RawPublicKey mode, where nodes have asymmetric-key pairs but no certificates to verify the ownership, and Certificate mode, where public keys are certified by a certification authority. An IoT implementation profile [RFC7925] is defined for TLS version 1.2 and DTLS version 1.2 that offers communication security for resource-constrained nodes.

There is ongoing work to define an authorization and access-control framework for resource-constrained nodes. The Authentication and Authorization for Constrained Environments (ACE) [WG-ACE] working group is defining a solution to allow only authorized access to resources that are hosted on a smart object server and are identified by a URI. The current proposal [ID-aceoauth] is based on the OAuth 2.0 framework [RFC6749] and it comes with profiles intended for different communication scenarios, e.g. DTLS Profile for Authentication and Authorization for Constrained Environments [ID-acedtls].

The CoAP base specification [RFC7252] provides a description of how DTLS can be used for securing CoAP. It proposes three different modes for using DTLS: the PreSharedKey mode, where nodes have pre-provisioned keys for initiating a DTLS session with another node, RawPublicKey mode, where nodes have asymmetric-key pairs but no certificates to verify the ownership, and Certificate mode, where public keys are certified by a certification authority. An IoT implementation profile [RFC7925] is defined for TLS version 1.2 and DTLS version 1.2 that offers communication security for resource-constrained nodes.

OSCORE [ID-OSCORE] is a proposal that protects CoAP messages by wrapping them in the CBOR Object Signing and Encryption (COSE) [RFC8152] format. Thus, OSCORE falls in the category of object security and it can be applied wherever CoAP can be used. The advantage of OSCORE over DTLS is that it provides some more flexibility when dealing with end-to-end security. Section 5.1.3 discusses this further.

The Automated Certificate Management Environment (ACME) [WG-ACME] working group is specifying conventions for automated X.509 certificate management. This includes automatic validation of certificate issuance, certificate renewal, and certificate revocation. While the initial focus of working group is on domain name certificates (as used by web servers), other uses in some IoT deployments is possible.

The Internet Key Exchange (IKEv2)/IPsec - as well as the less used Host Identity protocol (HIP) - reside at or above the network layer in the OSI model. Both protocols are able to perform an authenticated key exchange and set up the IPsec for secure payload delivery. Currently, there are also ongoing efforts to create a HIP variant coined Diet HIP [ID-HIP-DEX] that takes constrained networks and nodes into account at the authentication and key exchange level.

Migault et al. [ID-dietesp] are working on a compressed version of IPsec so that it can easily be used by resource-constrained IoT devices. They rely on the Internet Key Exchange Protocol version 2 (IKEv2) for negotiating the compression format.

The Extensible Authentication Protocol (EAP) [RFC3748] is an authentication framework supporting multiple authentication methods. EAP runs directly over the data link layer and, thus, does not require the deployment of IP. It supports duplicate detection and retransmission, but does not allow for packet fragmentation. The Protocol for Carrying Authentication for Network Access (PANA) is a network-layer transport for EAP that enables network access authentication between clients and the network infrastructure. In EAP terms, PANA is a UDP-based EAP lower layer that runs between the EAP peer and the EAP authenticator.

4.3. IoT Security Guidelines

Attacks on and from IoT devices have become common in the last years, for instance, large scale Denial of Service (DoS) attacks on the Internet Infrastructure from compromised IoT devices. This fact has prompted many different standards bodies and consortia to provide guidelines for developers and the Internet community at large to build secure IoT devices and services. A subset of the different guidelines and ongoing projects are as follows:

1. Global System for Mobile Communications (GSM) Association (GSMA) IoT security guidelines [GSMAsecurity]: GSMA has published a set of security guidelines for the benefit of new IoT product and service providers. The guidelines are aimed at device manufacturers, service providers, developers and network operators. An enterprise can complete an IoT Security Self-

Assessment to demonstrate that its products and services are aligned with the security guidelines of the GSMA.

2. Broadband Internet Technical Advisory Group (BITAG) IoT Security and Privacy Recommendations [BITAG]: BITAG has published recommendations for ensuring security and privacy of IoT device users. BITAG observes that many IoT devices are shipped from the factory with software that is already outdated and vulnerable. The report also states that many devices with vulnerabilities will not be fixed either because the manufacturer does not provide updates or because the user does not apply them. The recommendations include that IoT devices should function without cloud and Internet connectivity, and that all IoT devices should have methods for automatic secure software updates.

3. United Kingdom Department for Digital, Culture, Media and Sport (DCMS) [DCMS]: UK DCMS has releasead a report that includes a list of 13 steps for improving IoT security. These steps, for example, highlight the need for implementing a vulnerability disclosure policy and keeping software updated. The report is aimed at device manufacturers, IoT service providers, mobile application developers and retailers.

4. Cloud Security Alliance (CSA) New Security Guidance for Early Adopters of the IoT [CSA]: CSA recommendations for early adopters of IoT encourages enterprises to implement security at different layers of the protocol stack. It also recommends implementation of an authentication/authorization framework for IoT deployments. A complete list of recommendations is available in the report [CSA].

5. United States Department of Homeland Security [DHS]: DHS has put forth six strategic principles that would enable IoT developers, manufacturers, service providers and consumers to maintain security as they develop, manufacture, implement or use network-connected IoT devices.

6. National Institute of Standards and Technology (NIST) [NIST-Guide]: The NIST special publication urges enterprise and US federal agencies to address security throughout the systems engineering process. The publication builds upon the International Organization for Standardization (ISO)/International Electrotechnical Commission (IEC) 15288 standard and augments each process in the system lifecycle with security enhancements.

7. National Institute of Standards and Technology (NIST) [nist_lightweight_project]: NIST is running a project on lightweight cryptography with the purpose of: (i) identifying application areas for which standard cryptographic algorithms are too heavy, classifying them according to some application profiles to be determined; (ii) determining limitations in those existing cryptographic standards; and (iii) standardizing lightweight algorithms that can be used in specific application profiles.

8. Open Web Application Security Project (OWASP) [OWASP]: OWASP provides security guidance for IoT manufactures, developers and consumers. OWASP also includes guidelines for those who intend to test and analyze IoT devices and applications.

9. IoT Security foundation [IoTSecFoundation]: IoT security foundation has published a document that enlists various considerations that need to be taken into account when developing IoT applications. For example, the document states that IoT devices could use hardware-root of trust to ensure that only authorized software runs on the devices.

10. National Highway Traffic Safety Administration (NHTSA) [NHTSA]: The US NHTSA provides guidance to the automotive industry for improving the cyber security of vehicles. While some of the guidelines are general, the document provides specific recommendations for the automotive industry such as how various automotive manufacturer can share cyber security vulnerabilities discovered.

11. Best Current Practices (BCP) for IoT devices [ID-Moore]: This document provides a list of minimum requirements that vendors of Internet of Things (IoT) devices should to take into account while developing applications, services and firmware updates in order to reduce the frequency and severity of security incidents that arise from compromised IoT devices.

12. European Union Agency for Network and Information Security (ENISA) [ENISA_ICS]: ENISA published a document on communication network dependencies for Industrial Control Systems (ICS)/Supervisory Control And Data Acquisition (SCADA) systems in which security vulnerabilities, guidelines and general recommendations are summarized.

Other guideline and recommendation documents may exist or may later be published. This list should be considered non-exhaustive. Despite the acknowledgment that security in the Internet is needed and the existence of multiple guidelines, the fact is that many IoT devices and systems have very limited security. There are multiple reasons for this. For instance, some manufactures focus on delivering a product without paying enough attention to security. This may be because of lack of expertise or limited budget. However, the deployment of such insecure devices poses a severe threat on the privacy and safety of users. The vast amount of devices and their inherent mobile nature also implies that an initially secure system can become insecure if a compromised device gains access to the system at some point in time. Even if all other devices in a given environment are secure, this does not prevent external attacks caused by insecure devices. Recently the Federal Communications Commission (FCC) [FCC] has stated the need for additional regulation of IoT systems. It is possible that we may see other such regional regulations in the future.

5. Challenges for a Secure IoT

In this section, we take a closer look at the various security challenges in the operational and technical features of IoT and then discuss how existing Internet security protocols cope with these technical and conceptual challenges through the lifecycle of a thing. This discussion should neither be understood as a comprehensive evaluation of all protocols, nor can it cover all possible aspects of IoT security. Yet, it aims at showing concrete limitations and challenges in some IoT design areas rather than giving an abstract discussion. In this regard, the discussion handles issues that are most important from the authors' perspectives.

5.1. Constraints and Heterogeneous Communication

Coupling resource-constrained networks and the powerful Internet is a challenge because the resulting heterogeneity of both networks complicates protocol design and system operation. In the following we briefly discuss the resource constraints of IoT devices and the consequences for the use of Internet Protocols in the IoT domain.

5.1.1. Resource Constraints

IoT deployments are often characterized by lossy and low-bandwidth communication channels. IoT devices are also often constrained in terms of CPU, memory, and energy budget available [RFC7228]. These characteristics directly impact the design of protocols for the IoT domain. For instance, small packet size limits at the physical layer (127 Bytes in IEEE 802.15.4) can lead to (i) hop-by-hop fragmentation and reassembly or (ii) small IP-layer maximum transmission unit (MTU). In the first case, excessive fragmentation of large packets that are often required by security protocols may open new attack vectors for state exhaustion attacks. The second case might lead to more fragmentation at the IP layer which commonly downgrades the overall system performance due to packet loss and the need for retransmission.

The size and number of messages should be minimized to reduce memory requirements and optimize bandwidth usage. In this context, layered approaches involving a number of protocols might lead to worse performance in resource-constrained devices since they combine the headers of the different protocols. In some settings, protocol negotiation can increase the number of exchanged messages. To improve performance during basic procedures such as, for example, bootstrapping, it might be a good strategy to perform those procedures at a lower layer.

Small CPUs and scarce memory limit the usage of resource-expensive cryptographic primitives such as public-key cryptography as used in most Internet security standards. This is especially true if the basic cryptographic blocks need to be frequently used or the underlying application demands low delay.

There are ongoing efforts to reduce the resource consumption of security protocols by using more efficient underlying cryptographic primitives such as Elliptic Curve Cryptography [RFC5246]. The specification of elliptic curve X25519 [ecc25519], stream ciphers such as ChaCha [ChaCha], Diet HIP [ID-HIP-DEX], and ECC goups for IKEv2 [RFC5903] are all examples of efforts to make security protocols more resource efficient. Additionally, most modern security protocols have been revised in the last few years to enable cryptographic agility, making cryptographic primitives interchangeable. However, these improvements are only a first step in reducing the computation and communication overhead of Internet protocols. The question remains if other approaches can be applied to leverage key agreement in these heavily resource-constrained environments.

A further fundamental need refers to the limited energy budget available to IoT nodes. Careful protocol (re)design and usage is required to reduce not only the energy consumption during normal operation, but also under DoS attacks. Since the energy consumption of IoT devices differs from other device classes, judgments on the energy consumption of a particular protocol cannot be made without tailor-made IoT implementations.

5.1.2. Denial-of-Service Resistance

The tight memory and processing constraints of things naturally alleviate resource exhaustion attacks. Especially in unattended T2T communication, such attacks are difficult to notice before the service becomes unavailable (for example, because of battery or memory exhaustion). As a DoS countermeasure, DTLS, IKEv2, HIP, and Diet HIP implement return routability checks based on a cookie mechanism to delay the establishment of state at the responding host until the address of the initiating host is verified. The effectiveness of these defenses strongly depend on the routing topology of the network. Return routability checks are particularly effective if hosts cannot receive packets addressed to other hosts and if IP addresses present meaningful information as is the case in today's Internet. However, they are less effective in broadcast media or when attackers can influence the routing and addressing of hosts (for example, if hosts contribute to the routing infrastructure in ad-hoc networks and meshes).

In addition, HIP implements a puzzle mechanism that can force the initiator of a connection (and potential attacker) to solve cryptographic puzzles with variable difficulties. Puzzle-based defense mechanisms are less dependent on the network topology but perform poorly if CPU resources in the network are heterogeneous (for example, if a powerful Internet host attacks a thing). Increasing the puzzle difficulty under attack conditions can easily lead to situations where a powerful attacker can still solve the puzzle while weak IoT clients cannot and are excluded from communicating with the victim. Still, puzzle-based approaches are a viable option for sheltering IoT devices against unintended overload caused by misconfiguration or malfunctioning things.

5.1.3. End-to-end security, protocol translation, and the role of middleboxes

The term end-to-end security often has multiple interpretations. Here, we consider end-to-end security in the context end-to-end IP connectivity, from a sender to a receiver. Services such as confidentiality and integrity protection on packet data, message authentication codes or encryption are typically used to provide end-to-end security. These protection methods render the protected parts of the packets immutable as rewriting is either not possible because a) the relevant information is encrypted and inaccessible to the gateway or b) rewriting integrity-protected parts of the packet would invalidate the end-to-end integrity protection.

Protocols for constrained IoT networks are not exactly identical to their larger Internet counterparts for efficiency and performance reasons. Hence, more or less subtle differences between protocols for constrained IoT networks and Internet protocols will remain. While these differences can be bridged with protocol translators at middleboxes, they may become major obstacles if end-to-end security measures between IoT devices and Internet hosts are needed.

If access to data or messages by the middleboxes is required or acceptable, then a diverse set of approaches for handling such a scenario are available. Note that some of these approaches affect the meaning of end-to-end security in terms of integrity and confidentiality since the middleboxes will be able to either decrypt or modify partially the exchanged messages:

1. Sharing credentials with middleboxes enables them to transform (for example, decompress, convert, etc.) packets and re-apply the security measures after transformation. This method abandons end-to-end security and is only applicable to simple scenarios with a rudimentary security model.

2. Reusing the Internet wire format for IoT makes conversion between IoT and Internet protocols unnecessary. However, it can lead to poor performance in some use cases because IoT specific optimizations (for example, stateful or stateless compression) are not possible.

3. Selectively protecting vital and immutable packet parts with a message authentication code or with encryption requires a careful balance between performance and security. Otherwise this approach might either result in poor performance or poor security depending on which parts are selected for protection, where they are located in the original packet, and how they are processed. [ID-OSCORE] proposes a solution in this direction by encrypting and integrity protecting most of the message fields except those parts that a middlebox needs to read or change.

4. Homomorphic encryption techniques can be used in the middlebox to perform certain operations. However, this is limited to data processing involving arithmetic operations. Furthermore, performance of existing libraries, for example, SEAL [SEAL] is still too limited and homomorphic encryption techniques are not widely applicable yet.

5. Message authentication codes that sustain transformation can be realized by considering the order of transformation and protection (for example, by creating a signature before compression so that the gateway can decompress the packet without recalculating the signature). Such an approach enables IoT specific optimizations but is more complex and may require application-specific transformations before security is applied. Moreover, the usage of encrypted or integrity-protected data prevents middleboxes from transforming packets.

6. Mechanisms based on object security can bridge the protocol worlds, but still require that the two worlds use the same object

security formats. Currently the object security format based on CBOR Object Signing and Encryption (COSE) [RFC8152] is different from JSON Object Signing and Encryption (JOSE) [RFC7520] or Cryptographic Message Syntax (CMS) [RFC5652]. Legacy devices relying on traditional Internet protocols will need to update to the newer protocols for constrained environments to enable real end-to-end security. Furthermore, middleboxes do not have any access to the data and this approach does not prevent an attacker who is capable of modifying relevant fields in the payload.

To the best of our knowledge, none of the mentioned security approaches that focus on the confidentiality and integrity of the communication exchange between two IP end-points provide the perfect solution in this problem space.

5.1.4. New network architectures and paradigm

There is a multitude of new link layer protocols that aim to address the resource-constrained nature of IoT devices. For example, the IEEE 802.11 ah [IEEE802ah] has been specified for extended range and lower energy consumption to support Internet of Things (IoT) devices. Similarly, Low-Power Wide-Area Network (LPWAN) protocols such as LoRa [lora], Sigfox [sigfox], NarrowBand IoT (NB-IoT) [nbiot] are all designed for resource-constrained devices that require long range and low bit rates. [ID-lpwan] provides an informational overview of the set of LPWAN technologies being considered by the IETF. It also identifies the potential gaps that exist between the needs of those technologies and the goal of running IP in such networks. While these protocols allow IoT devices to conserve energy and operate efficiently, they also add additional security challenges. For example, the relatively small MTU can make security handshakes with large X509 certificates a significant overhead. At the same time, new communication paradigms also allow IoT devices to communicate directly amongst themselves with or without support from the network. This communication paradigm is also referred to as Device-to-Device (D2D) or Machine-to-Machine (M2M) or Thing-to-Thing (T2T) communication and it is motivated by a number of features such as improved network performance, lower latency and lower energy requirements.

5.2. Bootstrapping of a Security Domain

Creating a security domain from a set of previously unassociated IoT devices is a key operation in the lifecycle of a thing in an IoT network. This aspect is further elaborated and discussed in the T2TRG draft on bootstrapping [ID-bootstrap].

5.3. Operational Challenges

After the bootstrapping phase, the system enters the operational phase. During the operational phase, things can use the state information created during the bootstrapping phase in order to exchange information securely. In this section, we discuss the security challenges during the operational phase. Note that many of the challenges discussed in Section 5.1 apply during the operational phase.

5.3.1. Group Membership and Security

Group key negotiation is an important security service for IoT communication patterns in which a thing sends some data to multiple things or data flows from multiple things towards a thing. All discussed protocols only cover unicast communication and therefore, do not focus on group-key establishment. This applies in particular to (D)TLS and IKEv2. Thus, a solution is required in this area. A potential solution might be to use the Diffie-Hellman keys - that are used in IKEv2 and HIP to setup a secure unicast link - for group Diffie-Hellman key-negotiations. However, Diffie-Hellman is a relatively heavy solution, especially if the group is large.

Symmetric and asymmetric keys can be used in group communication. Asymmetric keys have the advantage that they can provide source authentication. However, doing broadcast encryption with a single public/private key pair is also not feasible. Although a single symmetric key can be used to encrypt the communication or compute a message authentication code, it has inherent risks since the capture of a single node can compromise the key shared throughout the network. The usage of symmetric-keys also does not provide source authentication. Another factor to consider is that asymmetric cryptography is more resource-intensive than symmetric key solutions. Thus, the security risks and performance trade-offs of applying either symmetric or asymmetric keys to a given IoT use case need to be well-analyzed according to risk and usability assessments. [ID-multicast] is looking at a combination of symmetric (for encryption) and asymmetric (for authentication) in the same packet.

Conceptually, solutions that provide secure group communication at the network layer (IPsec/IKEv2, HIP/Diet HIP) may have an advantage in terms of the cryptographic overhead when compared to application-focused security solutions (TLS/ DTLS). This is due to the fact that application-focused solutions require cryptographic operations per group application, whereas network layer approaches may allow sharing secure group associations between multiple applications (for example, for neighbor discovery and routing or service discovery). Hence, implementing shared features lower in the communication stack can avoid redundant security measures. However, it is important to note that sharing security contexts among different applications involves potential security threats, e.g., if one of the applications is malicious and monitors exchanged messages or injects fake messages. In the case of OSCORE, it provides security for CoAP group communication as defined in RFC7390, i.e., based on multicast IP. If the same security association is reused for each application, then this solution does not seem to have more cryptographic overhead compared to IPsec.

Several group key solutions have been developed by the MSEC working group [WG-MSEC] of the IETF. The MIKEY architecture [RFC4738] is one example. While these solutions are specifically tailored for multicast and group broadcast applications in the Internet, they should also be considered as candidate solutions for group key agreement in IoT. The MIKEY architecture for example describes a coordinator entity that disseminates symmetric keys over pair-wise end-to-end secured channels. However, such a centralized approach may not be applicable in a distributed IoT environment, where the choice of one or several coordinators and the management of the group key is not trivial.

5.3.2. Mobility and IP Network Dynamics

It is expected that many things (for example, wearable sensors, and user devices) will be mobile in the sense that they are attached to different networks during the lifetime of a security association. Built-in mobility signaling can greatly reduce the overhead of the cryptographic protocols because unnecessary and costly re-establishments of the session (possibly including handshake and key agreement) can be avoided. IKEv2 supports host mobility with the MOBIKE [RFC4555] and [RFC4621] extension. MOBIKE refrains from applying heavyweight cryptographic extensions for mobility. However, MOBIKE mandates the use of IPsec tunnel mode which requires the transmission of an additional IP header in each packet.

HIP offers a simple yet effective mobility management by allowing hosts to signal changes to their associations [RFC8046]. However, slight adjustments might be necessary to reduce the cryptographic costs, for example, by making the public-key signatures in the mobility messages optional. Diet HIP does not define mobility yet but it is sufficiently similar to HIP and can use the same mechanisms. DTLS provides some mobility support by relying on a connection ID (CID). The use of connection IDs can provide all the mobility functionality described in [ID-Williams], except, sending the updated location. The specific need for IP-layer mobility mainly depends on the scenario in which the nodes operate. In many cases, mobility supported by means of a mobile gateway may suffice to enable mobile IoT networks, such as body sensor networks. Using message based application-layer security solutions such as OSCORE [ID-OSCORE] can also alleviate the problem of re-establishing lower-layer sessions for mobile nodes.

5.4. Secure software update and cryptographic agility

IoT devices are often expected to stay functional for several years and decades even though they might operate unattended with direct Internet connectivity. Software updates for IoT devices are therefore not only required for new functionality, but also to eliminate security vulnerabilities due to software bugs, design flaws, or deprecated algorithms. Software bugs might remain even after careful code review. Implementations of security protocols might contain (design) flaws. Cryptographic algorithms can also become insecure due to advances in cryptanalysis. Therefore, it is necessary that devices which are incapable of verifying a cryptographic signature are not exposed to the Internet (even indirectly).

Schneier [SchneierSecurity] in his essay highlights several challenges that hinder mechanisms for secure software update of IoT devices. First, there is a lack of incentives for manufactures, vendors and others on the supply chain to issue updates for their devices. Second, parts of the software running on IoT devices is simply a binary blob without any source code available. Since the complete source code is not available, no patches can be written for that piece of code. Lastly Schneier points out that even when updates are available, users generally have to manually download and install them. However, users are never alerted about security updates and at many times do not have the necessary expertise to manually administer the required updates.

The FTC staff report on Internet of Things - Privacy & Security in a Connected World [FTCreport] and the Article 29 Working Party Opinion 8/2014 on the Recent Developments on the Internet of Things [Article29] also document the challenges for secure remote software update of IoT devices. They note that even providing such a software update capability may add new vulnerabilities for constrained devices. For example, a buffer overflow vulnerability in the implementation of a software update protocol (TR69) [TR69] and an expired certificate in a hub device [wink] demonstrate how the software update process itself can introduce vulnerabilities.

Powerful IoT devices that run general purpose operating systems can make use of sophisticated software update mechanisms known from the desktop world. However, resource-constrained devices typically do not have any operating system and are often not equipped with a memory management unit or similar tools. Therefore, they might require more specialized solutions.

An important requirement for secure software and firmware updates is source authentication. Source authentication requires the resource-constrained things to implement public-key signature verification algorithms. As stated in Section 5.1.1, resource-constrained things have limited amount of computational capabilities and energy supply available which can hinder the amount and frequency of cryptographic processing that they can perform. In addition to source authentication, software updates might require confidential delivery over a secure (encrypted) channel. The complexity of broadcast encryption can force the usage of point-to-point secure links - however, this increases the duration of a software update in a large system. Alternatively, it may force the usage of solutions in which the software update is delivered to a gateway, and then distributed to the rest of the system with a network key. Sending large amounts of data that later needs to be assembled and verified over a secure channel can consume a lot of energy and computational resources. Correct scheduling of the software updates is also a crucial design challenge. For example, a user of connected light bulbs would not want them to update and restart at night. More importantly, the user would not want all the lights to update at the same time.

Software updates in IoT systems are also needed to update old and insecure cryptographic primitives. However, many IoT systems, some of which are already deployed, are not designed with provisions for cryptographic agility. For example, many devices come with a wireless radio that has an AES128 hardware co-processor. These devices solely rely on the co-processor for encrypting and authenticating messages. A software update adding support for new cryptographic algorithms implemented solely in software might not fit on these devices due to limited memory, or might drastically hinder its operational performance. This can lead to the use of old and insecure devices. Therefore, it is important to account for the fact that cryptographic algorithms would need to be updated and consider the following when planning for cryptographic agility:

1. Would it be safe to use the existing cryptographic algorithms available on the device for updating with new cryptographic algorithms that are more secure?

2. Will the new software-based implementation fit on the device given the limited resources?

3. Would the normal operation of existing IoT applications on the device be severely hindered by the update?

Finally, we would like to highlight the previous and ongoing work in the area of secure software and firmware updates at the IETF. [RFC4108] describes how Cryptographic Message Syntax (CMS) [RFC5652] can be used to protect firmware packages. The IAB has also organized a workshop to understand the challenges for secure software update of IoT devices. A summary of the recommendations to the standards community derived from the discussions during that workshop have been documented [RFC8240]. A new working group called Software Updates for Internet of Things (suit) [WG-SUIT] is currently being chartered at the IETF. The working group aims to standardize a new version [RFC4108] that reflects the best current practices for firmware update based on experience with IoT deployments. It will specifically work on describing an IoT firmware update architecture and specifying a manifest format that contains meta-data about the firmware update package. Finally, the Trusted Execution Environment Provisioning working group [WG-TEEP] aims at developing a protocol for lifecycle management of trusted applications running on the secure area of a processor (Trusted Execution Enviornment (TEE)).

5.5. End-of-Life

Like all commercial devices, IoT devices have a given useful lifetime. The term end-of-life (EOL) is used by vendors or network operators to indicate the point of time in which they limit or end support for the IoT device. This may be planned or unplanned (for example when the manufacturer goes bankrupt, when the vendor just decides to abandon a product, or when a network operator moves to a different type of networking technology). A user should still be able to use and perhaps even update the device. This requires for some form of authorization handover.

Although this may seem far-fetched given the commercial interests and market dynamics, we have examples from the mobile world where the devices have been functional and up-to-date long after the original vendor stopped supporting the device. CyanogenMod for Android devices, and OpenWrt for home routers are two such instances where users have been able to use and update their devices even after the official EOL. Admittedly it is not easy for an average user to install and configure their devices on their own. With the deployment of millions of IoT devices, simpler mechanisms are needed to allow users to add new root-of-trusts and install software and firmware from other sources once the device is EOL.

5.6. Verifying device behavior

Users using new IoT appliances such as Internet-connected smart televisions, speakers and cameras are often unaware that these devices can undermine their privacy. Recent revelations have shown that many IoT device vendors have been collecting sensitive private data through these connected appliances with or without appropriate user warnings [cctv].

An IoT device user/owner would like to monitor and verify its operational behavior. For instance, the user might want to know if the device is connecting to the server of the manufacturer for any reason. This feature - connecting to the manufacturer's server - may be necessary in some scenarios, such as during the initial configuration of the device. However, the user should be kept aware of the data that the device is sending back to the vendor. For example, the user might want to know if his/her TV is sending data when he/she inserts a new USB stick.

Providing such information to the users in an understandable fashion is challenging. This is because IoT devices are not only resource-constrained in terms of their computational capability, but also in terms of the user interface available. Also, the network infrastructure where these devices are deployed will vary significantly from one user environment to another. Therefore, where and how this monitoring feature is implemented still remains an open question.

Manufacturer Usage Description (MUD) files [ID-MUD] are perhaps a first step towards implementation of such a monitoring service. The idea behind MUD files is relatively simple: IoT devices would disclose the location of their MUD file to the network during installation. The network can then retrieve those files, and learn about the intended behavior of the devices stated by the device manufacturer. A network monitoring service could then warn the user/owner of devices if they don't behave as expected.

Many devices and software services that automatically learn and monitor the behavior of different IoT devices in a given network are commercially available. Such monitoring devices/services can be configured by the user to limit network traffic and trigger alarms when unexpected operation of IoT devices is detected.

5.7. Testing: bug hunting and vulnerabilities

Given that IoT devices often have inadvertent vulnerabilities, both users and developers would want to perform extensive testing on their IoT devices, networks, and systems. Nonetheless, since the devices are resource-constrained and manufactured by multiple vendors, some of them very small, devices might be shipped with very limited testing, so that bugs can remain and can be exploited at a later stage. This leads to two main types of challenges:

1. It remains to be seen how the software testing and quality assurance mechanisms used from the desktop and mobile world will be applied to IoT devices to give end users the confidence that the purchased devices are robust. Bodies such as the European Cyber Security Organization (ECSO) [ECSO] are working on processes for security certification of IoT devices.

2. It is also an open question how the combination of devices from multiple vendors might actually lead to dangerous network configurations. For example, if combination of specific devices can trigger unexpected behavior. It is needless to say that the security of the whole system is limited by its weakest point.

5.8. Quantum-resistance

Many IoT systems that are being deployed today will remain operational for many years. With the advancements made in the field of quantum computers, it is possible that large-scale quantum computers are available in the future for performing cryptanalysis on existing cryptographic algorithms and cipher suites. If this happens, it will have two consequences. First, functionalities enabled by means of RSA/ECC - namely key exchange, public-key encryption and signature - would not be secure anymore due to Shor's algorithm. Second, the security level of symmetric algorithms will decrease, for example, the security of a block cipher with a key size of b bits will only offer b/2 bits of security due to Grover's algorithm.

The above scenario becomes more urgent when we consider the so called "harvest and decrypt" attack in which an attacker can start to harvest (store) encrypted data today, before a quantum-computer is available, and decrypt it years later, once a quantum computer is available. Such "harvest and decrypt" attacks are not new and were used in the Venona project [venona-project]. Many IoT devices that are being deployed today will remain operational for a decade or even longer. During this time, digital signatures used to sign software updates might become obsolete making the secure update of IoT devices challenging.

This situation would require us to move to quantum-resistant alternatives, in particular, for those functionalities involving key exchange, public-key encryption and signatures. [ID-c2pq] describes when quantum computers may become widely available and what steps are necessary for transition to cryptographic algorithms that provide security even in presence of quantum computers. While future planning is hard, it may be a necessity in certain critical IoT deployments which are expected to last decades or more. Although increasing the key-size of the different algorithms is definitely an option, it would also incur additional computational overhead and network traffic. This would be undesirable in most scenarios. There have been recent advancements in quantum-resistant cryptography. We refer to [ETSI_GR_QSC_001] for an extensive overview of existing quantum-resistant cryptography and [RFC7696] provides guidelines for cryptographic algorithm agility.

5.9. Privacy protection

People will eventually be surrounded by hundreds of connected IoT devices. Even if the communication links are encrypted and protected, information about people might still be collected or processed for different purposes. The fact that IoT devices in the vicinity of people might enable more pervasive monitoring can negatively impact their privacy. For instance, imagine the scenario where a static presence sensor emits a packet due to the presence or absence of people in its vicinity. In such a scenario, anyone who can observe the packet, can gather critical privacy-sensitive information.

Such information about people is referred to as personal data in the European Union (EU) or Personally identifiable information (PII) in the United States (US), In particular, the General Data Protection Regulation (GDPR) [GDPR] defines personal data as: 'any information relating to an identified or identifiable natural person ('data subject'); an identifiable natural person is one who can be identified, directly or indirectly, in particular by reference to an identifier such as a name, an identification number, location data, an online identifier or to one or more factors specific to the physical, physiological, genetic, mental, economic, cultural or social identity of that natural person'.

Ziegeldorf [Ziegeldorf] defines privacy in IoT as a threefold guarantee:

1. Awareness of the privacy risks imposed by IoT devices and services. This awareness is achieved by means of transparent practices by the data controller, i.e., the entity that is providing IoT devices and/or services.

2. Individual control over the collection and processing of personal information by IoT devices and services.

3. Awareness and control of the subsequent use and dissemination of personal information by data controllers to any entity outside the subject's personal control sphere. This point implies that the data controller must be accountable for its actions on the personal information.

Based on this definition, several threats to the privacy of users have been documented [Ziegeldorf] and [RFC6973], in particular considering the IoT environment and its lifecycle:

1. Identification - refers to the identification of the users, their IoT devices, and generated data.

2. Localization - relates to the capability of locating a user and even tracking them, e.g., by tracking MAC addresses in Wi-Fi or Bluetooth.

3. Profiling - is about creating a profile of the user and their preferences.

4. Interaction - occurs when a user has been profiled and a given interaction is preferred, presenting (for example, visually) some information that discloses private information.

5. Lifecycle transitions - take place when devices are, for example, sold without properly removing private data.

6. Inventory attacks - happen if specific information about IoT devices in possession of a user is disclosed.

7. Linkage - is about when information of two of more IoT systems (or other data sets) is combined so that a broader view of the personal data captured can be created.

When IoT systems are deployed, the above issues should be considered to ensure that private data remains private. These issues are particularly challenging in environments in which multiple users with different privacy preferences interact with the same IoT devices. For example, an IoT device controlled by user A (low privacy settings) might leak private information about another user B (high privacy settings). How to deal with these threats in practice is an area of ongoing research.

5.10. Reverse engineering considerations

Many IoT devices are resource-constrained and often deployed in unattended environments. Some of these devices can also be purchased off-the-shelf or online without any credential-provisioning process. Therefore, an attacker can have direct access to the device and apply advanced techniques to retrieve information that a traditional black box model does not consider. Example of those techniques are side-channel attacks or code disassembly. By doing this, the attacker can try to retrieve data such as:

1. long term keys. These long term keys can be extracted by means of a side-channel attack or reverse engineering. If these keys are exposed, then they might be used to perform attacks on devices deployed in other locations.

2. source code. Extraction of source code might allow the attacker to determine bugs or find exploits to perform other types of attacks. The attacker might also just sell the source code.

3. proprietary algorithms. The attacker can analyze these algorithms gaining valuable know-how. The attacker can also create copies of the product (based on those proprietary algorithms) or modify the algorithms to perform more advanced attacks.

4. configuration or personal data. The attacker might be able to read personal data, e.g., healthcare data, that has been stored on a device.

One existing solution to prevent such data leaks is the use of a secure element, a tamper-resistant device that is capable of securely hosting applications and their confidential data. Another potential solution is the usage of of Physical Unclonable Function (PUFs) that serves as unique digital fingerprint of a hardware device. PUFs can also enable other functionalities such as secure key storage. Protection against such data leakage patterns is non-trivial since devices are inherently resource-constrained. An open question is whether there are any viable techniques to protect IoT devices and the data in the devices in such an adversarial model.

5.11. Trustworthy IoT Operation

Flaws in the design and implementation of IoT devices and networks can lead to security vulnerabilities. A common flaw is the use of well-known or easy-to-guess passwords for configuration of IoT devices. Many such compromised IoT devices can be found on the Internet by means of tools such as Shodan [shodan]. Once discovered, these compromised devices can be exploited at scale, for example, to launch DDoS attacks. Dyn, a major DNS , was attacked by means of a DDoS attack originating from a large IoT botnet composed of thousands of compromised IP-cameras [dyn-attack]. There are several open research questions in this area:

1. How to avoid vulnerabilities in IoT devices that can lead to large-scale attacks?

2. How to detect sophisticated attacks against IoT devices?

3. How to prevent attackers from exploiting known vulnerabilities at a large scale?

Some ideas are being explored to address this issue. One of the approaches relies on the use of Manufacturer Usage Description (MUD) files [ID-MUD]. As explained earlier, this proposal requires IoT devices to disclose the location of their MUD file to the network during installation. The network can then (i) retrieve those files, (ii) learn from the manufacturers the intended usage of the devices, for example, which services they need to access, and then (iii) create suitable filters and firewall rules.

6. Conclusions and Next Steps

This Internet Draft provides IoT security researchers, system designers and implementers with an overview of security requirements in the IP-based Internet of Things. We discuss the security threats, state-of-the-art, and challenges.

Although plenty of steps have been realized during the last few years (summarized in Section 4.1) and many organizations are publishing general recommendations (Section 4.3) describing how IoT should be secured, there are many challenges ahead that require further attention. Challenges of particular importance are bootstrapping of security, group security, secure software updates, long-term security and quantum-resistance, privacy protection, data leakage prevention - where data could be cryptographic keys, personal data, or even algorithms - and ensuring trustworthy IoT operation.

Authors of new IoT specifications and implementors need to consider how all the security challenges discussed in this draft (and those that emerge later) affect their work. The authors of IoT specifications not only need to put in a real effort towards addressing the security challenges, but also clearly documenting how the security challenges are addressed. This would reduce the chances of security vulnerabilities in the code written by implementors of those specifications.

7. Security Considerations

This entire memo deals with security issues.

8. IANA Considerations

This document contains no request to IANA.

9. Acknowledgments

We gratefully acknowledge feedback and fruitful discussion with Tobias Heer, Robert Moskowitz, Thorsten Dahm, Hannes Tschofenig, Carsten Bormann, Barry Raveendran, Ari Keranen, Goran Selander, Fred Baker, Vicent Roca, Thomas Fossati and Eliot Lear. We acknowledge the additional authors of the previous version of this document Sye Loong Keoh, Rene Hummen and Rene Struik.

10. Informative References

[Article29]
"Opinion 8/2014 on the on Recent Developments on the Internet of Things", Web http://ec.europa.eu/justice/data-protection/article-29/documentation/opinion-recommendation/files/2014/wp223_en.pdf, n.d..

[AUTO-ID] "AUTO-ID LABS", Web http://www.autoidlabs.org/, September 2010.

[BACNET] "BACnet", Web http://www.bacnet.org/, February 2011.

[BITAG] "Internet of Things (IoT) Security and Privacy Recommendations", Web http://www.bitag.org/report-internet-of-things-security-privacy-recommendations.php, n.d..

[cctv] "Backdoor In MVPower DVR Firmware Sends CCTV Stills To an Email Address In China", Web https://hardware.slashdot.org/story/16/02/17/0422259/backdoor-in-mvpower-dvr-firmware-sends-cctv-stills-to-an-email-address-in-china, n.d..

[ChaCha] Bernstein, D., "ChaCha, a variant of Salsa20", Web http://cr.yp.to/chacha/chacha-20080128.pdf, n.d..

[CSA] "Security Guidance for Early Adopters of the Internet of Things (IoT)", Web https://downloads.cloudsecurityalliance.org/whitepapers/Security_Guidance_for_Early_Adopters_of_the_Internet_of_Things.pdf, n.d..

[DALI] "DALI", Web http://www.dalibydesign.us/dali.html, February 2011.

[DCMS] "Secure by Design: Improving the cyber security of consumer Internet of Things Report", Web https://www.gov.uk/government/publications/secure-by-design, n.d..

[DHS] "Strategic Principles For Securing the Internet of Things (IoT)", Web https://www.dhs.gov/sites/default/files/publications/Strategic_Principles_for_Securing_the_Internet_of_Things-2016-1115-FINAL....pdf, n.d..

[dyn-attack] "Dyn Analysis Summary Of Friday October 21 Attack", Web https://dyn.com/blog/dyn-analysis-summary-of-friday-october-21-attack/, n.d..

[ecc25519] Bernstein, D., "Curve25519: new Diffie-Hellman speed records", Web https://cr.yp.to/ecdh/curve25519-20060209.pdf, n.d..

[ECSO] "European Cyber Security Organization", Web https://www.ecs-org.eu/, n.d..

[ENISA_ICS] "Communication network dependencies for ICS/SCADA Systems", European Union Agency For Network And Information Security , February 2017.

[ETSI_GR_QSC_001] "Quantum-Safe Cryptography (QSC);Quantum-safe algorithmic framework", European Telecommunications Standards Institute (ETSI) , June 2016.

[Fairhair] "Fairhair Alliance", Web https://www.fairhair-alliance.org/, n.d..

[FCC] "Federal Communications Comssion Response 12-05-2016", FCC , February 2016.

[FTCreport] "FTC Report on Internet of Things Urges Companies to Adopt Best Practices to Address Consumer Privacy and Security Risks", Web https://www.ftc.gov/news-events/press-releases/2015/01/ftc-report-internet-things-urges-companies-adopt-best-practices, n.d..

[GDPR] "The EU General Data Protection Regulation", Web https://www.eugdpr.org/, n.d..

[GSMAsecurity]
"GSMA IoT Security Guidelines", Web http://www.gsma.com/connectedliving/future-iot-networks/iot-security-guidelines/, n.d..

[ID-6lonfc]
Choi, Y., Hong, Y., Youn, J., Kim, D., and J. Choi, "Transmission of IPv6 Packets over Near Field Communication", draft-ietf-6lo-nfc-09 (work in progress), January 2018.

[ID-6tisch]
Thubert, P., "An Architecture for IPv6 over the TSCH mode of IEEE 802.15.4", draft-ietf-6tisch-architecture-14 (work in progress), April 2018.

[ID-acedtls]
Gerdes, S., Bergmann, O., Bormann, C., Selander, G., and L. Seitz, "Datagram Transport Layer Security (DTLS) Profile for Authentication and Authorization for Constrained Environments (ACE)", draft-ietf-ace-dtls-authorize-03 (work in progress), March 2018.

[ID-aceoauth]
Seitz, L., Selander, G., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "Authentication and Authorization for Constrained Environments (ACE) using the OAuth 2.0 Framework (ACE-OAuth)", draft-ietf-ace-oauth-authz-11 (work in progress), March 2018.

[ID-bootstrap]
Sarikaya, B., Sethi, M., and A. Sangi, "Secure IoT Bootstrapping: A Survey", draft-sarikaya-t2trg-sbootstrapping-03 (work in progress), February 2017.

[ID-c2pq] Hoffman, P., "The Transition from Classical to Post-Quantum Cryptography", draft-hoffman-c2pq-03 (work in progress), February 2018.

[ID-Daniel]
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.

[ID-dietesp]
Migault, D., Guggemos, T., and C. Bormann, "Diet-ESP: a flexible and compressed format for IPsec/ESP", draft-mglt-6lo-diet-esp-02 (work in progress), July 2016.

[ID-HIP-DEX]
Moskowitz, R., "HIP Diet EXchange (DEX)", draft-moskowitz-hip-rg-dex-06 (work in progress), May 2012.

[ID-lpwan]
Farrell, S., "LPWAN Overview", draft-ietf-lpwan-overview-10 (work in progress), February 2018.

[ID-Moore]
Moore, K., Barnes, R., and H. Tschofenig, "Best Current Practices for Securing Internet of Things (IoT) Devices", draft-moore-iot-security-bcp-01 (work in progress), July 2017.

[ID-MUD] Lear, E., Droms, R., and D. Romascanu, "Manufacturer Usage Description Specification", draft-ietf-opsawg-mud-21 (work in progress), May 2018.

[ID-multicast]
Tiloca, M., Selander, G., Palombini, F., and J. Park, "Secure group communication for CoAP", draft-ietf-core-oscore-groupcomm-01 (work in progress), March 2018.

[ID-OSCORE]
Selander, G., Mattsson, J., Palombini, F., and L. Seitz, "Object Security for Constrained RESTful Environments (OSCORE)", draft-ietf-core-object-security-12 (work in progress), March 2018.

[ID-rd] Shelby, Z., Koster, M., Bormann, C., Stok, P., and C. Amsuess, "CoRE Resource Directory", draft-ietf-core-resource-directory-13 (work in progress), March 2018.

[ID-senml]
Jennings, C., Shelby, Z., Arkko, J., Keranen, A., and C. Bormann, "Sensor Measurement Lists (SenML)", draft-ietf-core-senml-16 (work in progress), May 2018.

[ID-Williams]
Williams, M. and J. Barrett, "Mobile DTLS", draft-barrett-mobile-dtls-00 (work in progress), March 2009.

[IEEE802ah]
"Status of Project IEEE 802.11ah, IEEE P802.11- Task Group AH-Meeting Update.", Web http://www.ieee802.org/11/Reports/tgah_update.htm, n.d..

[IIoT] "Industrial Internet Consortium", Web http://www.iiconsortium.org/, n.d..

[IoTSecFoundation]
"Establishing Principles for Internet of Things Security", Web https://iotsecurityfoundation.org/establishing-principles-for-internet-of-things-security/, n.d..

[IPSO] "IPSO Alliance", Web http://www.ipso-alliance.org, n.d..

[lora] "LoRa - Wide Area Networks for IoT", Web https://www.lora-alliance.org/, n.d..

[LWM2M] "OMA LWM2M", Web http://openmobilealliance.org/iot/lightweight-m2m-lwm2m, n.d..

[mirai] Kolias, C., Kambourakis, G., Stavrou, A., and J. Voas,, "DDoS in the IoT: Mirai and Other Botnets", IEEE Computer , 2017.

[nbiot] "NarrowBand IoT", Web http://www.3gpp.org/ftp/tsg_ran/TSG_RAN/TSGR_69/Docs/RP-151621.zip, n.d..

[NHTSA] "Cybersecurity Best Practices for Modern Vehicles", Web https://www.nhtsa.gov/staticfiles/nvs/pdf/812333_CybersecurityForModernVehicles.pdf, n.d..

[NIST-Guide]
Ross, R., McEvilley, M., and J. Oren, "Systems Security Engineering", Web http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-160.pdf, n.d..

[nist_lightweight_project]
"NIST lightweight Project", Web www.nist.gov/programs-projects/lightweight-cryptography, www.nist.gov/sites/default/files/documents/2016/10/17/sonmez-turan-presentation-lwc2016.pdf, n.d..

[NISTSP800-122]
Erika McCallister, ., Tim Grance, ., and . Karen Scarfone, "NIST SP800-122 - Guide to Protecting the Confidentiality of Personally Identifiable Information", Web https://nvlpubs.nist.gov/nistpubs/legacy/sp/nistspecialpublication800-122.pdf, n.d..

[NISTSP800-30r1]
"NIST SP 800-30r1 - Guide for Conducting Risk Assessments", Web https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-30r1.pdf, n.d..

[NISTSP800-34r1]
Marianne Swanson, ., Pauline Bowen, ., Amy Wohl Phillips, ., Dean Gallup, ., and . David Lynes, "NIST SP800-34r1 - Contingency Planning Guide for Federal Information Systems", Web https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-34r1.pdf, n.d..

[OCF] "Open Connectivity Foundation", Web https://openconnectivity.org/, n.d..

[OneM2M] "OneM2M", Web http://www.onem2m.org/, n.d..

[OWASP] "IoT Security Guidance", Web https://www.owasp.org/index.php/IoT_Security_Guidance, n.d..

[RFC2818] Rescorla, E., "HTTP Over TLS", RFC 2818, DOI 10.17487/RFC2818, May 2000, <https://www.rfc-editor.org/info/rfc2818>.

[RFC3748] Aboba, B., Blunk, L., Vollbrecht, J., Carlson, J., and H. Levkowetz, Ed., "Extensible Authentication Protocol (EAP)", RFC 3748, DOI 10.17487/RFC3748, June 2004, <https://www.rfc-editor.org/info/rfc3748>.

[RFC3756] Nikander, P., Ed., Kempf, J., and E. Nordmark, "IPv6 Neighbor Discovery (ND) Trust Models and Threats", RFC 3756, DOI 10.17487/RFC3756, May 2004, <https://www.rfc-editor.org/info/rfc3756>.

[RFC3833] Atkins, D. and R. Austein, "Threat Analysis of the Domain Name System (DNS)", RFC 3833, DOI 10.17487/RFC3833, August 2004, <https://www.rfc-editor.org/info/rfc3833>.

[RFC4016] Parthasarathy, M., "Protocol for Carrying Authentication and Network Access (PANA) Threat Analysis and Security Requirements", RFC 4016, DOI 10.17487/RFC4016, March 2005, <https://www.rfc-editor.org/info/rfc4016>.

[RFC4108] Housley, R., "Using Cryptographic Message Syntax (CMS) to Protect Firmware Packages", RFC 4108, DOI 10.17487/RFC4108, August 2005, <https://www.rfc-editor.org/info/rfc4108>.

[RFC4555] Eronen, P., "IKEv2 Mobility and Multihoming Protocol (MOBIKE)", RFC 4555, DOI 10.17487/RFC4555, June 2006, <https://www.rfc-editor.org/info/rfc4555>.

[RFC4621] Kivinen, T. and H. Tschofenig, "Design of the IKEv2 Mobility and Multihoming (MOBIKE) Protocol", RFC 4621, DOI 10.17487/RFC4621, August 2006, <https://www.rfc-editor.org/info/rfc4621>.

[RFC4738] Ignjatic, D., Dondeti, L., Audet, F., and P. Lin, "MIKEY-RSA-R: An Additional Mode of Key Distribution in Multimedia Internet KEYing (MIKEY)", RFC 4738, DOI 10.17487/RFC4738, November 2006, <https://www.rfc-editor.org/info/rfc4738>.

[RFC4919] Kushalnagar, N., Montenegro, G., and C. Schumacher, "IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs): Overview, Assumptions, Problem Statement, and Goals", RFC 4919, DOI 10.17487/RFC4919, August 2007, <https://www.rfc-editor.org/info/rfc4919>.

[RFC4944] Montenegro, G., Kushalnagar, N., Hui, J., and D. Culler, "Transmission of IPv6 Packets over IEEE 802.15.4 Networks", RFC 4944, DOI 10.17487/RFC4944, September 2007, <https://www.rfc-editor.org/info/rfc4944>.

[RFC5191] Forsberg, D., Ohba, Y., Ed., Patil, B., Tschofenig, H., and A. Yegin, "Protocol for Carrying Authentication for Network Access (PANA)", RFC 5191, DOI 10.17487/RFC5191, May 2008, <https://www.rfc-editor.org/info/rfc5191>.

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

[RFC5652] Housley, R., "Cryptographic Message Syntax (CMS)", STD 70, RFC 5652, DOI 10.17487/RFC5652, September 2009, <https://www.rfc-editor.org/info/rfc5652>.

[RFC5713] Moustafa, H., Tschofenig, H., and S. De Cnodder, "Security Threats and Security Requirements for the Access Node Control Protocol (ANCP)", RFC 5713, DOI 10.17487/RFC5713, January 2010, <https://www.rfc-editor.org/info/rfc5713>.

[RFC5903] Fu, D. and J. Solinas, "Elliptic Curve Groups modulo a Prime (ECP Groups) for IKE and IKEv2", RFC 5903, DOI 10.17487/RFC5903, June 2010, <https://www.rfc-editor.org/info/rfc5903>.

[RFC6272] Baker, F. and D. Meyer, "Internet Protocols for the Smart Grid", RFC 6272, DOI 10.17487/RFC6272, June 2011, <https://www.rfc-editor.org/info/rfc6272>.

[RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, DOI 10.17487/RFC6347, January 2012, <https://www.rfc-editor.org/info/rfc6347>.

[RFC6550] Winter, T., Ed., Thubert, P., Ed., Brandt, A., Hui, J., Kelsey, R., Levis, P., Pister, K., Struik, R., Vasseur, JP., and R. Alexander, "RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks", RFC 6550, DOI 10.17487/RFC6550, March 2012, <https://www.rfc-editor.org/info/rfc6550>.

[RFC6551] Vasseur, JP., Ed., Kim, M., Ed., Pister, K., Dejean, N., and D. Barthel, "Routing Metrics Used for Path Calculation in Low-Power and Lossy Networks", RFC 6551, DOI 10.17487/RFC6551, March 2012, <https://www.rfc-editor.org/info/rfc6551>.

[RFC6568] Kim, E., Kaspar, D., and JP. Vasseur, "Design and Application Spaces for IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs)", RFC 6568, DOI 10.17487/RFC6568, April 2012, <https://www.rfc-editor.org/info/rfc6568>.

[RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, DOI 10.17487/RFC6690, August 2012, <https://www.rfc-editor.org/info/rfc6690>.

[RFC6749] Hardt, D., Ed., "The OAuth 2.0 Authorization Framework", RFC 6749, DOI 10.17487/RFC6749, October 2012, <https://www.rfc-editor.org/info/rfc6749>.

[RFC6973] Cooper, A., Tschofenig, H., Aboba, B., Peterson, J., Morris, J., Hansen, M., and R. Smith, "Privacy Considerations for Internet Protocols", RFC 6973, DOI 10.17487/RFC6973, July 2013, <https://www.rfc-editor.org/info/rfc6973>.

[RFC7049] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", RFC 7049, DOI 10.17487/RFC7049, October 2013, <https://www.rfc-editor.org/info/rfc7049>.

[RFC7228] Bormann, C., Ersue, M., and A. Keranen, "Terminology for Constrained-Node Networks", RFC 7228, DOI 10.17487/RFC7228, May 2014, <https://www.rfc-editor.org/info/rfc7228>.

[RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <https://www.rfc-editor.org/info/rfc7252>.

[RFC7296] Kaufman, C., Hoffman, P., Nir, Y., Eronen, P., and T. Kivinen, "Internet Key Exchange Protocol Version 2 (IKEv2)", STD 79, RFC 7296, DOI 10.17487/RFC7296, October 2014, <https://www.rfc-editor.org/info/rfc7296>.

[RFC7401] Moskowitz, R., Ed., Heer, T., Jokela, P., and T. Henderson, "Host Identity Protocol Version 2 (HIPv2)", RFC 7401, DOI 10.17487/RFC7401, April 2015, <https://www.rfc-editor.org/info/rfc7401>.

[RFC7515] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Signature (JWS)", RFC 7515, DOI 10.17487/RFC7515, May 2015, <https://www.rfc-editor.org/info/rfc7515>.

[RFC7516] Jones, M. and J. Hildebrand, "JSON Web Encryption (JWE)", RFC 7516, DOI 10.17487/RFC7516, May 2015, <https://www.rfc-editor.org/info/rfc7516>.

[RFC7517] Jones, M., "JSON Web Key (JWK)", RFC 7517, DOI 10.17487/RFC7517, May 2015, <https://www.rfc-editor.org/info/rfc7517>.

[RFC7519] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Token (JWT)", RFC 7519, DOI 10.17487/RFC7519, May 2015, <https://www.rfc-editor.org/info/rfc7519>.

[RFC7520] Miller, M., "Examples of Protecting Content Using JSON Object Signing and Encryption (JOSE)", RFC 7520, DOI 10.17487/RFC7520, May 2015, <https://www.rfc-editor.org/info/rfc7520>.

[RFC7668] Nieminen, J., Savolainen, T., Isomaki, M., Patil, B., Shelby, Z., and C. Gomez, "IPv6 over BLUETOOTH(R) Low Energy", RFC 7668, DOI 10.17487/RFC7668, October 2015, <https://www.rfc-editor.org/info/rfc7668>.

[RFC7696] Housley, R., "Guidelines for Cryptographic Algorithm Agility and Selecting Mandatory-to-Implement Algorithms", BCP 201, RFC 7696, DOI 10.17487/RFC7696, November 2015, <https://www.rfc-editor.org/info/rfc7696>.

[RFC7744] Seitz, L., Ed., Gerdes, S., Ed., Selander, G., Mani, M., and S. Kumar, "Use Cases for Authentication and Authorization in Constrained Environments", RFC 7744, DOI 10.17487/RFC7744, January 2016, <https://www.rfc-editor.org/info/rfc7744>.

[RFC7815] Kivinen, T., "Minimal Internet Key Exchange Version 2 (IKEv2) Initiator Implementation", RFC 7815, DOI 10.17487/RFC7815, March 2016, <https://www.rfc-editor.org/info/rfc7815>.

[RFC7925] Tschofenig, H., Ed. and T. Fossati, "Transport Layer Security (TLS) / Datagram Transport Layer Security (DTLS) Profiles for the Internet of Things", RFC 7925, DOI 10.17487/RFC7925, July 2016, <https://www.rfc-editor.org/info/rfc7925>.

[RFC8046] Henderson, T., Ed., Vogt, C., and J. Arkko, "Host Mobility with the Host Identity Protocol", RFC 8046, DOI 10.17487/RFC8046, February 2017, <https://www.rfc-editor.org/info/rfc8046>.

[RFC8105] Mariager, P., Petersen, J., Ed., Shelby, Z., Van de Logt, M., and D. Barthel, "Transmission of IPv6 Packets over Digital Enhanced Cordless Telecommunications (DECT) Ultra Low Energy (ULE)", RFC 8105, DOI 10.17487/RFC8105, May 2017, <https://www.rfc-editor.org/info/rfc8105>.

[RFC8152] Schaad, J., "CBOR Object Signing and Encryption (COSE)", RFC 8152, DOI 10.17487/RFC8152, July 2017, <https://www.rfc-editor.org/info/rfc8152>.

[RFC8240] Tschofenig, H. and S. Farrell, "Report from the Internet of Things Software Update (IoTSU) Workshop 2016", RFC 8240, DOI 10.17487/RFC8240, September 2017, <https://www.rfc-editor.org/info/rfc8240>.

[RFC8259] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", STD 90, RFC 8259, DOI 10.17487/RFC8259, December 2017, <https://www.rfc-editor.org/info/rfc8259>.

[RG-T2TRG] "IRTF Thing-to-Thing (T2TRG) Research Group", Web https://datatracker.ietf.org/rg/t2trg/charter/, n.d..

[SchneierSecurity] "The Internet of Things Is Wildly Insecure--And Often Unpatchable", Web https://www.schneier.com/essays/archives/2014/01/the_internet_of_thin.html, n.d..

[SEAL] "Simple Encrypted Arithmetic Library - SEAL", Web https://www.microsoft.com/en-us/research/publication/simple-encrypted-arithmetic-library-seal-v2-0/, n.d..

[shodan] "Shodan", Web https://www.shodan.io/, n.d..

[sigfox] "Sigfox - The Global Communications Service Provider for the Internet of Things (IoT)", Web https://www.sigfox.com/, n.d..

[Thread] "Thread Group", Web http://threadgroup.org/, n.d..

[TR69] "Too Many Cooks - Exploiting the Internet-of-TR-069-Things", Web https://media.ccc.de/v/31c3_-_6166_-_en_-_saal_6_-_201412282145_-_too_many_cooks_-_exploiting_the_internet-of-tr-069-things_-_lior_oppenheim_-_shahar_tal, n.d..

[venona-project] "Venona Project", Web https://www.nsa.gov/news-features/declassified-documents/venona/index.shtml, n.d..

[WG-6lo] "IETF IPv6 over Networks of Resource-constrained Nodes (6lo) Working Group", Web https://datatracker.ietf.org/wg/6lo/charter/, n.d..

[WG-6LoWPAN]
"IETF IPv6 over Low power WPAN (6lowpan) Working Group", Web http://tools.ietf.org/wg/6lowpan/, n.d..

[WG-ACE] "IETF Authentication and Authorization for Constrained Environments (ACE) Working Group", Web https://datatracker.ietf.org/wg/ace/charter/, n.d..

[WG-ACME] "Automated Certificate Management Environment Working Group", Web https://datatracker.ietf.org/wg/acme/about/, n.d..

[WG-CoRE] "IETF Constrained RESTful Environment (CoRE) Working Group", Web https://datatracker.ietf.org/wg/core/charter/, n.d..

[WG-LPWAN]
"IETF Low Power Wide-Area Networks Working Group", Web https://datatracker.ietf.org/wg/lpwan/, n.d..

[WG-LWIG] "IETF Light-Weight Implementation Guidance (LWIG) Working Group", Web https://datatracker.ietf.org/wg/lwig/charter/, n.d..

[WG-MSEC] "IETF MSEC Working Group", Web https://datatracker.ietf.org/wg/msec/, n.d..

[WG-SUIT] "IETF Software Updates for Internet of Things (suit)", Web https://datatracker.ietf.org/group/suit/about/, n.d..

[WG-TEEP] "IETF Trusted Execution Environment Provisioning (teep)", Web https://datatracker.ietf.org/wg/teep/about/, n.d..

[wink] "Wink's Outage Shows Us How Frustrating Smart Homes Could Be", Web http://www.wired.com/2015/04/smart-home-headaches/, n.d..

[ZB] "ZigBee Alliance", Web http://www.zigbee.org/, February 2011.

[Ziegeldorf]
Ziegeldorf, J., Garcia-Morchon, O., and K. Wehrle,, "Privacy in the Internet of Things: Threats and Challenges", Security and Communication Networks - Special Issue on Security in a Completely Interconnected World , 2013.

Authors' Addresses

Oscar Garcia-Morchon
Philips IP&S
High Tech Campus 5
Eindhoven, 5656 AA
The Netherlands

Email: oscar.garcia-morchon@philips.com

Sandeep S. Kumar
Philips Research
High Tech Campus
Eindhoven, 5656 AA
The Netherlands

Email: sandeep.kumar@philips.com

Mohit Sethi
Ericsson
Hirsalantie 11
Jorvas, 02420
Finland

Email: mohit@piuha.net

Network Working Group O. Garcia-Morchon
Internet-Draft Philips IP&S
Intended status: Informational S. Kumar
Expires: June 16, 2019 Philips Research
M. Sethi
Ericsson
December 13, 2018

State-of-the-Art and Challenges for the Internet of Things Security
draft-irtf-t2trg-iot-seccons-16

Abstract

The Internet of Things (IoT) concept refers to the usage of standard Internet protocols to allow for human-to-thing and thing-to-thing communication. The security needs for IoT systems are well-recognized and many standardization steps to provide security have been taken, for example, the specification of Constrained Application Protocol (CoAP) secured with Datagram Transport Layer Security (DTLS). However, security challenges still exist, not only because there are some use cases that lack a suitable solution, but also because many IoT devices and systems have been designed and deployed with very limited security capabilities. In this document, we first discuss the various stages in the lifecycle of a thing. Next, we document the security threats to a thing and the challenges that one might face to protect against these threats. Lastly, we discuss the next steps needed to facilitate the deployment of secure IoT systems. This document can be used by implementors and authors of IoT specifications as a reference for details about security considerations while documenting their specific security challenges, threat models, and mitigations.

This document is a product of the IRTF Thing-to-Thing Research Group (T2TRG).

Status of This Memo

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 June 16, 2019.

Copyright Notice

Table of Contents

1. Introduction

The Internet of Things (IoT) denotes the interconnection of highly heterogeneous networked entities and networks that follow a number of different communication patterns such as: human-to-human (H2H), human-to-thing (H2T), thing-to-thing (T2T), or thing-to-things (T2Ts). The term IoT was first coined by the Auto-ID center [AUTO-ID] in 1999 which had envisioned a world where every physical object is tagged with a radio-frequency identification (RFID) tag having a globally unique identifier. This would not only allow tracking of objects in real-time but also allow querying of data about them over the Internet. However, since then, the meaning of the Internet of Things has expanded and now encompasses a wide variety of technologies, objects and protocols. It is not surprising that the IoT has received significant attention from the research community to (re)design, apply, and use standard Internet technology and protocols for the IoT.

The things that are part of the Internet of Things are computing devices that understand and react to the environment they reside in. These things are also often referred to as smart objects or smart devices. The introduction of IPv6 [RFC6568] and CoAP [RFC7252] as fundamental building blocks for IoT applications allows connecting IoT hosts to the Internet. This brings several advantages including: (i) a homogeneous protocol ecosystem that allows simple integration with other Internet hosts; (ii) simplified development for devices that significantly vary in their capabilities; (iii) a unified interface for applications, removing the need for application-level proxies. These building blocks greatly simplify the deployment of the envisioned scenarios which range from building automation to production environments and personal area networks.

This document presents an overview of important security aspects for the Internet of Things. We begin by discussing the lifecycle of a thing in Section 2. In Section 3, we discuss security threats for the IoT and methodologies for managing these threats when designing a secure system. Section 4 reviews existing IP-based (security) protocols for the IoT and briefly summarizes existing guidelines and regulations. Section 5 identifies remaining challenges for a secure IoT and discusses potential solutions. Section 6 includes final remarks and conclusions. This document can be used by IoT standards specifications as a reference for details about security considerations applying to the specified system or protocol.

The first draft version of this document was submitted in March 2011. Initial draft versions of this document were presented and discussed during the CORE meetings at IETF 80 and later. Discussions on security lifecycle at IETF 92 (March 2015) evolved into more general security considerations. Thus, the draft was selected to address the T2TRG work item on the security considerations and challenges for the Internet of Things. Further updates of the draft were presented and discussed during the T2TRG meetings at IETF 96 (July 2016) and IETF 97 (November 2016) and at the joint interim in Amsterdam (March 2017). This document has been reviewed by, commented on, and discussed extensively for a period of nearly six years by a vast majority of T2TRG and related group members; the number of which certainly exceeds 100 individuals. It is the consensus of T2TRG that the security considerations described in this document should be published in the IRTF Stream of the RFC series. This document does not constitute a standard.

2. The Thing Lifecycle

The lifecycle of a thing refers to the operational phases of a thing in the context of a given application or use case. Figure 1 shows the generic phases of the lifecycle of a thing. This generic lifecycle is applicable to very different IoT applications and scenarios. For instance, [RFC7744] provides an overview of relevant IoT use cases.

In this document, we consider a Building Automation and Control (BAC) system to illustrate the lifecycle and the meaning of these different phases. A BAC system consists of a network of interconnected nodes that performs various functions in the domains of HVAC (Heating, Ventilating, and Air Conditioning), lighting, safety, etc. The nodes vary in functionality and a large majority of them represent resource-constrained devices such as sensors and luminaries. Some devices may be battery operated or may rely on energy harvesting. This requires us to also consider devices that sleep during their operation to save energy. In our BAC scenario, the life of a thing starts when it is manufactured. Due to the different application areas (i.e., HVAC, lighting, or safety) nodes/things are tailored to a specific task. It is therefore unlikely that one single manufacturer will create all nodes in a building. Hence, interoperability as well as trust bootstrapping between nodes of different vendors is important.

The thing is later installed and commissioned within a network by an installer during the bootstrapping phase. Specifically, the device identity and the secret keys used during normal operation may be provided to the device during this phase. Different subcontractors may install different IoT devices for different purposes. Furthermore, the installation and bootstrapping procedures may not be a discrete event and may stretch over an extended period. After being bootstrapped, the device and the system of things are in operational mode and execute the functions of the BAC system. During this operational phase, the device is under the control of the system owner and used by multiple system users. For devices with lifetimes spanning several years, occasional maintenance cycles may be required. During each maintenance phase, the software on the device can be upgraded or applications running on the device can be reconfigured. The maintenance tasks can be performed either locally or from a backend system. Depending on the operational changes to the device, it may be required to re-bootstrap at the end of a maintenance cycle. The device continues to loop through the operational phase and the eventual maintenance phases until the device is decommissioned at the end of its lifecycle. However, the end-of-life of a device does not necessarily mean that it is defective and rather denotes a need to replace and upgrade the network to the next-generation devices for additional functionality. Therefore, the device can be removed and re-commissioned to be used in a different system under a different owner thereby starting the lifecycle all over again.

We note that the presented lifecycle represents to some extent a simplified model. For instance, it is possible to argue that the lifecycle does not start when a tangible device is manufactured but rather when the oldest bit of code that ends up in the device - maybe from an open source project or from the used operating system - was written. Similarly, the lifecycle could also include an on-the-shelf phase where the device is in the supply-chain before an owner/user purchases and installs it. Another phase could involve the device being re-badged by some vendor who is not the original manufacturer. Such phases can significantly complicate other phases such as maintenance and bootstrapping. Finally, other potential end-states can be, e.g., a vendor that no longer supports a device type because it is at end-of-life or a situation in which a device is simply forgotten but remains functional.

```
    _Manufactured           _SW update          _Decommissioned
   /                       /                   /
   |   _Installed          |   _ Application   |   _Removed &
   |  /                    |  / reconfigured   |  /  replaced
   |  |   _Commissioned    |  |                |  |
   |  |  /                 |  |                |  |   _Reownership &
   |  |  |    _Application |  |   _Application |  |  / recommissioned
   |  |  |   /   running   |  |  / running     |  |  |
   |  |  |   |             |  |  |             |  |  |             \\
   +##+##+###+#############+##+##+#############+##+##+##############>>>
       \/  \_______________/ \/  \_____________/ \___/         time //
       /           /         \          \          \
   Bootstrapping  /      Maintenance &   \     Maintenance &
                 /      re-bootstrapping  \   re-bootstrapping
           Operational               Operational
```

Figure 1: The lifecycle of a thing in the Internet of Things

Security is a key requirement in any communication system. However, security is an even more critical requirement in real-world IoT deployments for several reasons. First, compromised IoT systems can not only endanger the privacy and security of a user, but can also cause physical harm. This is because IoT systems often comprise sensors, actuators and other connected devices in the physical environment of the user which could adversely affect the user if they are compromised. Second, a vulnerable IoT system means that an attacker can alter the functionality of a device from a given manufacturer. This not only affects the manufacturer's brand image, but can also leak information that is very valuable for the manufacturer (such as proprietary algorithms). Third, the impact of attacking an IoT system goes beyond a specific device or an isolated system since compromised IoT systems can be misused at scale. For example, they may be used to perform a Distributed Denial of Service (DDoS) attack that limits the availability of other networks and services. The fact that many IoT systems rely on standard IP protocols allows for easier system integration, but this also makes attacks on standard IP protocols widely applicable in other environments. This results in new requirements regarding the implementation of security.

The term security subsumes a wide range of primitives, protocols, and procedures. For instance, the term security includes services such as confidentiality, authentication, integrity, authorization, source authentication, and availability. The term security often also includes augmented services such as duplicate detection and detection of stale packets (timeliness). These security services can be implemented through a combination of cryptographic mechanisms such as block ciphers, hash functions, and signature algorithms; as well as non-cryptographic mechanisms that implement authorization and other security policy enforcement aspects. For ensuring security in IoT networks, one should not only focus on the required security services, but also pay special attention to how the services are realized in the overall system.

3. Security Threats and Managing Risk

Security threats in related IP protocols have been analyzed in multiple documents including Hypertext Transfer Protocol (HTTP) over Transport Layer Security (TLS) (HTTPS) [RFC2818], Constrained Application Protocol (COAP) [RFC7252], IPv6 over Low-Power Wireless Personal Area Networks (6LoWPAN) [RFC4919], Access Node Control Protocol (ANCP) [RFC5713], Domain Name System (DNS) [RFC3833], IPv6 Neighbor Discovery (ND) [RFC3756], and Protocol for Carrying Authentication and Network Access (PANA) [RFC4016]. In this section, we specifically discuss the threats that could compromise an individual thing or the network as a whole. Some of these threats might go beyond the scope of Internet protocols but we gather them here for the sake of completeness. The threats in the following list are not in any particular order and some threats might be more critical than others depending on the deployment scenario under consideration:

1. Vulnerable Software/Code: Things in the Internet of Things rely on software that might contain severe bugs and/or bad design choices. This makes the things vulnerable to many different types of attacks, depending on the criticality of the bugs, e.g., buffer overflows or lack of authentication. This can be considered as one of the most important security threat. The large-scale distributed denial-of-service (DDoS) attack, popularly known as the Mirai botnet [mirai], was caused by things that had well-known or easy-to-guess passwords for configuration.

2. Privacy threat: The tracking of a thing's location and usage may pose a privacy risk to people around it. For instance, an attacker can infer privacy sensitive information from the data gathered and communicated by individual things. Such information may subsequently be sold to interested parties for marketing purposes and targeted advertising. In extreme cases, such information might be used to track dissidents in oppressive regimes. Unlawful surveillance and interception of traffic to/ from a thing by intelligence agencies is also a privacy threat.

3. Cloning of things: During the manufacturing process of a thing, an untrusted factory can easily clone the physical characteristics, firmware/software, or security configuration of

the thing. Deployed things might also be compromised and their software reverse engineered allowing for cloning or software modifications. Such a cloned thing may be sold at a cheaper price in the market, and yet can function normally as a genuine thing. For example, two cloned devices can still be associated and work with each other. In the worst-case scenario, a cloned device can be used to control a genuine device or perform an attack. One should note here, that an untrusted factory may also change functionality of the cloned thing, resulting in degraded functionality with respect to the genuine thing (thereby, inflicting potential damage to the reputation of the original thing manufacturer). Moreover, additional functionality can be introduced in the cloned thing. An example of such functionality is a backdoor.

4. Malicious substitution of things: During the installation of a thing, a genuine thing may be substituted with a similar variant (of lower quality) without being detected. The main motivation may be cost savings, where the installation of lower-quality things (for example, non-certified products) may significantly reduce the installation and operational costs. The installers can subsequently resell the genuine things to gain further financial benefits. Another motivation may be to inflict damage to the reputation of a competitor's offerings.

5. Eavesdropping attack: During the commissioning of a thing into a network, it may be susceptible to eavesdropping, especially if operational keying materials, security parameters, or configuration settings, are exchanged in clear using a wireless medium or if used cryptographic algorithms are not suitable for the envisioned lifetime of the device and the system. After obtaining the keying material, the attacker might be able to recover the secret keys established between the communicating entities, thereby compromising the authenticity and confidentiality of the communication channel, as well as the authenticity of commands and other traffic exchanged over this communication channel. When the network is in operation, T2T communication can be eavesdropped if the communication channel is not sufficiently protected or if a session key is compromised due to protocol weaknesses. An adversary may also be able to eavesdrop if keys are not renewed or updated appropriately. Lastly, messages can also be recorded and decrypted offline at a later point of time. The Venona project [venona-project] is one such example where messages were recorded for offline decryption.

6. Man-in-the-middle attack: Both the commissioning phase and operational phases may also be vulnerable to man-in-the-middle

attacks. For example, when keying material between communicating entities is exchanged in the clear and the security of the key establishment protocol depends on the tacit assumption that no third party can eavesdrop during the execution of this protocol. Additionally, device authentication or device authorization may be non-trivial, or may need support of a human decision process, since things usually do not have a-priori knowledge about each other and cannot always differentiate friends and foes via completely automated mechanisms.

7. Firmware attacks: When a thing is in operation or maintenance phase, its firmware or software may be updated to allow for new functionality or new features. An attacker may be able to exploit such a firmware upgrade by maliciously replacing the thing's firware, thereby influencing its operational behavior. For example, an attacker could add a piece of malicious code to the firmware that will cause it to periodically report the energy usage of the thing to a data repository for analysis. The attacker can then use this information to determine when a home or enterprise (where the thing is installed) is unoccupied and break in. Similarly, devices whose software has not been properly maintained and updated might contain vulnerabilities that might be exploited by attackers to replace the firmware on the device.

8. Extraction of private information: IoT devices (such as sensors, actuators, etc.) are often physically unprotected in their ambient environment and they could easily be captured by an attacker. An attacker with physical access may then attempt to extract private information such as keys (for example, device's key, private-key, group key), sensed data (for example, healthcare status of a user), configuration parameters (for example, the Wi-Fi key), or proprietary algorithms (for example, algorithm performing some data analytics task). Even when the data originating from a thing is encrypted, attackers can perform traffic analysis to deduce meaningful information which might compromise the privacy of the thing's owner and/or user.

9. Routing attack: As highlighted in [ID-Daniel], routing information in IoT networks can be spoofed, altered, or replayed, in order to create routing loops, attract/repel network traffic, extend/shorten source routes, etc. A non-exhaustive list of routing attacks includes 1) Sinkhole attack (or blackhole attack), where an attacker declares himself to have a high-quality route/path to the base station, thus allowing him to do manipulate all packets passing through it. 2) Selective forwarding, where an attacker may selectively forward

packets or simply drop a packet. 3) Wormhole attack, where an attacker may record packets at one location in the network and tunnel them to another location, thereby influencing perceived network behavior and potentially distorting statistics, thus greatly impacting the functionality of routing. 4) Sybil attack, whereby an attacker presents multiple identities to other things in the network. We refer to [ID-Daniel] for further router attacks and a more detailed description.

10. Elevation of privilege: An attacker with low privileges can misuse additional flaws in the implemented authentication and authorization mechanisms of a thing to gain more privileged access to the thing and its data.

11. Denial-of-Service (DoS) attack: Often things have very limited memory and computation capabilities. Therefore, they are vulnerable to resource exhaustion attack. Attackers can continuously send requests to specific things so as to deplete their resources. This is especially dangerous in the Internet of Things since an attacker might be located in the backend and target resource-constrained devices that are part of a constrained node network [RFC7228]. DoS attack can also be launched by physically jamming the communication channel. Network availability can also be disrupted by flooding the network with a large number of packets. On the other hand, things compromised by attackers can be used to disrupt the operation of other networks or systems by means of a Distributed DoS (DDoS) attack.

To deal with above threats it is required to find and apply suitable security mitigations. However, new threats and exploits appear on a daily basis and products are deployed in different environments prone to different types of threats. Thus, ensuring a proper level of security in an IoT system at any point of time is challenging. To address this challenge, some of the following methodologies can be used:

1. A Business Impact Analysis (BIA) assesses the consequences of the loss of basic security attributes: confidentiality, integrity and availability in an IoT system. These consequences might include the impact from lost data, reduced sales, increased expenses, regulatory fines, customer dissatisfaction, etc. Performing a business impact analysis allows a business to determine the relevance of having a proper security design.

2. A Risk Assessment (RA) analyzes security threats to an IoT system while considering their likelihood and impact. It also includes categorizing each of them with a risk level. Risks classified as

moderate or high must be mitigated, i.e., the security architecture should be able to deal with those threat.

3. A privacy impact assessment (PIA) aims at assessing the Personally Identifiable Information (PII) that is collected, processed, or used in an IoT system. By doing so, the goal is to fulfill applicable legal requirements, determine risks and effects of manipulation and loss of PII.

4. Procedures for incident reporting and mitigation refer to the methodologies that allow becoming aware of any security issues that affect an IoT system. Furthermore, this includes steps towards the actual deployment of patches that mitigate the identified vulnerabilities.

BIA, RA, and PIA should generally be realized during the creation of a new IoT system or when deploying significant system/feature upgrades. In general, it is recommended to re-assess them on a regular basis taking into account new use cases and/or threats. The way a BIA, RA, PIA are performed depends on the environment and the industry. More information can be found in NIST documents such as [NISTSP800-34r1], [NISTSP800-30r1], and [NISTSP800-122].

4. State-of-the-Art

This section is organized as follows. Section 4.1 summarizes state-of-the-art on IP-based IoT systems, within IETF and in other standardization bodies. Section 4.2 summarizes state-of-the-art on IP-based security protocols and their usage. Section 4.3 discusses guidelines and regulations for securing IoT as proposed by other bodies. Note that the references included in this section are a representative of the state-of-the-art at the point of writing and they are by no means exhaustive. The references are also at varying levels of maturity, and thus, it is advisable to review their specific status.

4.1. IP-based IoT Protocols and Standards

Nowadays, there exists a multitude of control protocols for IoT. For BAC systems, the ZigBee standard [ZB], BACNet [BACNET], and DALI [DALI] play key roles. Recent trends, however, focus on an all-IP approach for system control.

In this setting, a number of IETF working groups are designing new protocols for resource-constrained networks of smart things. The 6LoWPAN working group [WG-6LoWPAN] for example has defined methods and protocols for the efficient transmission and adaptation of IPv6 packets over IEEE 802.15.4 networks [RFC4944].

The CoRE working group [WG-CoRE] has specified the Constrained Application Protocol (CoAP) [RFC7252]. CoAP is a RESTful protocol for constrained devices that is modeled after HTTP and typically runs over UDP to enable efficient application-level communication for things.

In many smart object networks, the smart objects are dispersed and have intermittent reachability either because of network outages or because they sleep during their operational phase to save energy. In such scenarios, direct discovery of resources hosted on the constrained server might not be possible. To overcome this barrier, the CoRE working group is specifying the concept of a Resource Directory (RD) [ID-rd]. The Resource Directory hosts descriptions of resources which are located on other nodes. These resource descriptions are specified as CoRE link format [RFC6690].

While CoAP defines a standard communication protocol, a format for representing sensor measurements and parameters over CoAP is required. The Sensor Measurement Lists (SenML) [RFC8428] is a specification that defines media types for simple sensor measurements and parameters. It has a minimalistic design so that constrained devices with limited computational capabilities can easily encode their measurements and, at the same time, servers can efficiently collect large number of measurements.

In many IoT deployments, the resource-constrained smart objects are connected to the Internet via a gateway that is directly reachable. For example, an IEEE 802.11 Access Point (AP) typically connects the client devices to the Internet over just one wireless hop. However, some deployments of smart object networks require routing between the smart objects themselves. The IETF has therefore defined the IPv6 Routing Protocol for Low-Power and Lossy Networks (RPL) [RFC6550]. RPL provides support for multipoint-to-point traffic from resource-constrained smart objects towards a more resourceful central control point, as well as point-to-multipoint traffic in the reverse direction. It also supports point-to-point traffic between the resource-constrained devices. A set of routing metrics and constraints for path calculation in RPL are also specified [RFC6551].

The IPv6 over Networks of Resource-constrained Nodes (6lo) [WG-6lo] working group of the IETF has specified how IPv6 packets can be transmitted over various link layer protocols that are commonly employed for resource-constrained smart object networks. There is also ongoing work to specify IPv6 connectivity for a Non-Broadcast Multi-Access (NBMA) mesh network that is formed by IEEE 802.15.4 TimeSlotted Channel Hopping (TSCH} links [ID-6tisch]. Other link layer protocols for which IETF has specified or is currently specifying IPv6 support include Bluetooth [RFC7668], Digital Enhanced

Cordless Telecommunications (DECT) Ultra Low Energy (ULE) air interface [RFC8105], and Near Field Communication (NFC) [ID-6lonfc].

Baker and Meyer [RFC6272] identify which IP protocols can be used in smart grid environments. They give advice to smart grid network designers on how they can decide on a profile of the Internet protocol suite for smart grid networks.

The Low Power Wide-Area Network (LPWAN) working [WG-LPWAN] group is analyzing features, requirements, and solutions to adapt IP-based protocols to networks such as LORA [lora], SigFox [sigfox], NB-IoT [nbiot], etc. These networking technologies enable a smart thing to run for years on a single coin-cell by relying on a star network topology and using optimized radio modulation with frame sizes in the order of tens of bytes. Such networks bring new security challenges since most existing security mechanism do not work well with such resource constraints.

JavaScript Object Notation (JSON) is a lightweight text representation format for structured data [RFC8259]. It is often used for transmitting serialized structured data over the network. IETF has defined specifications for encoding cryptographic keys, encrypted content, signed content, and claims to be transferred between two parties as JSON objects. They are referred to as JSON Web Keys (JWK) [RFC7517], JSON Web Encryption (JWE) [RFC7516], JSON Web Signatures (JWS) [RFC7515] and JSON Web Token (JWT) [RFC7519].

An alternative to JSON, Concise Binary Object Representation (CBOR) [RFC7049] is a concise binary data format that is used for serialization of structured data. It is designed for resource-constrained nodes and therefore it aims to provide a fairly small message size with minimal implementation code, and extensibility without the need for version negotiation. CBOR Object Signing and Encryption (COSE) [RFC8152] specifies how to encode cryptographic keys, message authentication codes, encrypted content, and signatures with CBOR.

The Light-Weight Implementation Guidance (LWIG) working group [WG-LWIG] is collecting experiences from implementers of IP stacks in constrained devices. The working group has already produced documents such as RFC7815 [RFC7815] which defines how a minimal Internet Key Exchange Version 2 (IKEv2) initiator can be implemented.

The Thing-2-Thing Research Group (T2TRG) [RG-T2TRG] is investigating the remaining research issues that need to be addressed to quickly turn the vision of IoT into a reality where resource-constrained nodes can communicate with each other and with other more capable nodes on the Internet.

Additionally, industry alliances and other standardization bodies are creating constrained IP protocol stacks based on the IETF work. Some important examples of this include:

1. Thread [Thread]: Specifies the Thread protocol that is intended for a variety of IoT devices. It is an IPv6-based network protocol that runs over IEEE 802.15.4.

2. Industrial Internet Consortium [IIoT]: The consortium defines reference architectures and security frameworks for development, adoption and widespread use of Industrial Internet technologies based on existing IETF standards.

3. Internet Protocol for Smart Objects IPSO [IPSO]: The alliance specifies a common object model that enables application software on any device to interoperate with other conforming devices.

4. OneM2M [OneM2M]: The standards body defines technical and API specifications for IoT devices. It aims to create a service layer that can run on any IoT device hardware and software.

5. Open Connectivity Foundation (OCF) [OCF]: The foundation develops standards and certifications primarily for IoT devices that use Constrained Application Protocol (CoAP) as the application layer protocol.

6. Fairhair Alliance [Fairhair]: Specifies an IoT middleware to enable a common IP network infrastructure between different application standards used in building automation and lighting systems such as BACnet, KNX and ZigBee.

7. OMA LWM2M [LWM2M]: OMA Lightweight M2M is a standard from the Open Mobile Alliance for M2M and IoT device management. LWM2M relies on CoAP as the application layer protocol and uses a RESTful architecture for remote management of IoT devices.

4.2. Existing IP-based Security Protocols and Solutions

There are three main security objectives for IoT networks: 1. protecting the IoT network from attackers. 2. protecting IoT applications and thus, the things and users. 3. protecting the rest of the Internet and other things from attacks that use compromised things as an attack platform.

In the context of the IP-based IoT deployments, consideration of existing Internet security protocols is important. There are a wide range of specialized as well as general-purpose security solutions for the Internet domain such as IKEv2/IPsec [RFC7296], Transport

Layer Security (TLS) [RFC8446], Datagram Transport Layer Security (DTLS) [RFC6347], Host Identity Protocol (HIP) [RFC7401], PANA [RFC5191], Kerberos ([RFC4120]), Simple Authentication and Security Layer (SASL) [RFC4422], and Extensible Authentication Protocol (EAP) [RFC3748].

TLS provides security for TCP and requires a reliable transport. DTLS secures and uses datagram-oriented protocols such as UDP. Both protocols are intentionally kept similar and share the same ideology and cipher suites. The CoAP base specification [RFC7252] provides a description of how DTLS can be used for securing CoAP. It proposes three different modes for using DTLS: the PreSharedKey mode, where nodes have pre-provisioned keys for initiating a DTLS session with another node, RawPublicKey mode, where nodes have asymmetric-key pairs but no certificates to verify the ownership, and Certificate mode, where public keys are certified by a certification authority. An IoT implementation profile [RFC7925] is defined for TLS version 1.2 and DTLS version 1.2 that offers communication security for resource-constrained nodes.

There is ongoing work to define an authorization and access-control framework for resource-constrained nodes. The Authentication and Authorization for Constrained Environments (ACE) [WG-ACE] working group is defining a solution to allow only authorized access to resources that are hosted on a smart object server and are identified by a URI. The current proposal [ID-aceoauth] is based on the OAuth 2.0 framework [RFC6749] and it comes with profiles intended for different communication scenarios, e.g. DTLS Profile for Authentication and Authorization for Constrained Environments [ID-acedtls].

OSCORE [ID-OSCORE] is a proposal that protects CoAP messages by wrapping them in the CBOR Object Signing and Encryption (COSE) [RFC8152] format. Thus, OSCORE falls in the category of object security and it can be applied wherever CoAP can be used. The advantage of OSCORE over DTLS is that it provides some more flexibility when dealing with end-to-end security. Section 5.1.3 discusses this further.

The Automated Certificate Management Environment (ACME) [WG-ACME] working group is specifying conventions for automated X.509 certificate management. This includes automatic validation of certificate issuance, certificate renewal, and certificate revocation. While the initial focus of working group is on domain name certificates (as used by web servers), other uses in some IoT deployments is possible.

The Internet Key Exchange (IKEv2)/IPsec - as well as the less used Host Identity protocol (HIP) - reside at or above the network layer in the OSI model. Both protocols are able to perform an authenticated key exchange and set up the IPsec for secure payload delivery. Currently, there are also ongoing efforts to create a HIP variant coined Diet HIP [ID-HIP-DEX] that takes constrained networks and nodes into account at the authentication and key exchange level.

Migault et al. [ID-dietesp] are working on a compressed version of IPsec so that it can easily be used by resource-constrained IoT devices. They rely on the Internet Key Exchange Protocol version 2 (IKEv2) for negotiating the compression format.

The Extensible Authentication Protocol (EAP) [RFC3748] is an authentication framework supporting multiple authentication methods. EAP runs directly over the data link layer and, thus, does not require the deployment of IP. It supports duplicate detection and retransmission, but does not allow for packet fragmentation. The Protocol for Carrying Authentication for Network Access (PANA) is a network-layer transport for EAP that enables network access authentication between clients and the network infrastructure. In EAP terms, PANA is a UDP-based EAP lower layer that runs between the EAP peer and the EAP authenticator.

4.3. IoT Security Guidelines

Attacks on and from IoT devices have become common in the last years, for instance, large scale Denial of Service (DoS) attacks on the Internet Infrastructure from compromised IoT devices. This fact has prompted many different standards bodies and consortia to provide guidelines for developers and the Internet community at large to build secure IoT devices and services. A subset of the different guidelines and ongoing projects are as follows:

1. Global System for Mobile Communications (GSM) Association (GSMA) IoT security guidelines [GSMAsecurity]: GSMA has published a set of security guidelines for the benefit of new IoT product and service providers. The guidelines are aimed at device manufacturers, service providers, developers and network operators. An enterprise can complete an IoT Security Self-Assessment to demonstrate that its products and services are aligned with the security guidelines of the GSMA.

2. Broadband Internet Technical Advisory Group (BITAG) IoT Security and Privacy Recommendations [BITAG]: BITAG has published recommendations for ensuring security and privacy of IoT device users. BITAG observes that many IoT devices are shipped from the factory with software that is already outdated and

vulnerable. The report also states that many devices with vulnerabilities will not be fixed either because the manufacturer does not provide updates or because the user does not apply them. The recommendations include that IoT devices should function without cloud and Internet connectivity, and that all IoT devices should have methods for automatic secure software updates.

3. United Kingdom Department for Digital, Culture, Media and Sport (DCMS) [DCMS]: UK DCMS has released a report that includes a list of 13 steps for improving IoT security. These steps, for example, highlight the need for implementing a vulnerability disclosure policy and keeping software updated. The report is aimed at device manufacturers, IoT service providers, mobile application developers and retailers.

4. Cloud Security Alliance (CSA) New Security Guidance for Early Adopters of the IoT [CSA]: CSA recommendations for early adopters of IoT encourages enterprises to implement security at different layers of the protocol stack. It also recommends implementation of an authentication/authorization framework for IoT deployments. A complete list of recommendations is available in the report [CSA].

5. United States Department of Homeland Security [DHS]: DHS has put forth six strategic principles that would enable IoT developers, manufacturers, service providers and consumers to maintain security as they develop, manufacture, implement or use network-connected IoT devices.

6. National Institute of Standards and Technology (NIST) [NIST-Guide]: The NIST special publication urges enterprise and US federal agencies to address security throughout the systems engineering process. The publication builds upon the International Organization for Standardization (ISO)/International Electrotechnical Commission (IEC) 15288 standard and augments each process in the system lifecycle with security enhancements.

7. National Institute of Standards and Technology (NIST) [nist-lightweight-project]: NIST is running a project on lightweight cryptography with the purpose of: (i) identifying application areas for which standard cryptographic algorithms are too heavy, classifying them according to some application profiles to be determined; (ii) determining limitations in those existing cryptographic standards; and (iii) standardizing lightweight algorithms that can be used in specific application profiles.

8. Open Web Application Security Project (OWASP) [OWASP]: OWASP provides security guidance for IoT manufactures, developers and consumers. OWASP also includes guidelines for those who intend to test and analyze IoT devices and applications.

9. IoT Security foundation [IoTSecFoundation]: IoT security foundation has published a document that enlists various considerations that need to be taken into account when developing IoT applications. For example, the document states that IoT devices could use hardware-root of trust to ensure that only authorized software runs on the devices.

10. National Highway Traffic Safety Administration (NHTSA) [NHTSA]: The US NHTSA provides guidance to the automotive industry for improving the cyber security of vehicles. While some of the guidelines are general, the document provides specific recommendations for the automotive industry such as how various automotive manufacturer can share cyber security vulnerabilities discovered.

11. Best Current Practices (BCP) for IoT devices [ID-Moore]: This document provides a list of minimum requirements that vendors of Internet of Things (IoT) devices should to take into account while developing applications, services and firmware updates in order to reduce the frequency and severity of security incidents that arise from compromised IoT devices.

12. European Union Agency for Network and Information Security (ENISA) [ENISA-ICS]: ENISA published a document on communication network dependencies for Industrial Control Systems (ICS)/Supervisory Control And Data Acquisition (SCADA) systems in which security vulnerabilities, guidelines and general recommendations are summarized.

13. Internet Society Online Trust Alliance [ISOC-OTA]: The Internet Society's IoT Trust Framework identifies the core requirements manufacturers, service providers, distributors, purchasers and policymakers need to understand, assess and embrace for effective security and privacy as part of the Internet of Things.

Other guideline and recommendation documents may exist or may later be published. This list should be considered non-exhaustive. Despite the acknowledgment that security in the Internet is needed and the existence of multiple guidelines, the fact is that many IoT devices and systems have very limited security. There are multiple reasons for this. For instance, some manufactures focus on delivering a product without paying enough attention to security.

This may be because of lack of expertise or limited budget. However, the deployment of such insecure devices poses a severe threat on the privacy and safety of users. The vast amount of devices and their inherent mobile nature also implies that an initially secure system can become insecure if a compromised device gains access to the system at some point in time. Even if all other devices in a given environment are secure, this does not prevent external attacks caused by insecure devices. Recently the Federal Communications Commission (FCC) [FCC] has stated the need for additional regulation of IoT systems. It is possible that we may see other such regional regulations in the future.

5. Challenges for a Secure IoT

In this section, we take a closer look at the various security challenges in the operational and technical features of IoT and then discuss how existing Internet security protocols cope with these technical and conceptual challenges through the lifecycle of a thing. This discussion should neither be understood as a comprehensive evaluation of all protocols, nor can it cover all possible aspects of IoT security. Yet, it aims at showing concrete limitations and challenges in some IoT design areas rather than giving an abstract discussion. In this regard, the discussion handles issues that are most important from the authors' perspectives.

5.1. Constraints and Heterogeneous Communication

Coupling resource-constrained networks and the powerful Internet is a challenge because the resulting heterogeneity of both networks complicates protocol design and system operation. In the following we briefly discuss the resource constraints of IoT devices and the consequences for the use of Internet Protocols in the IoT domain.

5.1.1. Resource Constraints

IoT deployments are often characterized by lossy and low-bandwidth communication channels. IoT devices are also often constrained in terms of CPU, memory, and energy budget available [RFC7228]. These characteristics directly impact the design of protocols for the IoT domain. For instance, small packet size limits at the physical layer (127 Bytes in IEEE 802.15.4) can lead to (i) hop-by-hop fragmentation and reassembly or (ii) small IP-layer maximum transmission unit (MTU). In the first case, excessive fragmentation of large packets that are often required by security protocols may open new attack vectors for state exhaustion attacks. The second case might lead to more fragmentation at the IP layer which commonly downgrades the overall system performance due to packet loss and the need for retransmission.

The size and number of messages should be minimized to reduce memory requirements and optimize bandwidth usage. In this context, layered approaches involving a number of protocols might lead to worse performance in resource-constrained devices since they combine the headers of the different protocols. In some settings, protocol negotiation can increase the number of exchanged messages. To improve performance during basic procedures such as, for example, bootstrapping, it might be a good strategy to perform those procedures at a lower layer.

Small CPUs and scarce memory limit the usage of resource-expensive cryptographic primitives such as public-key cryptography as used in most Internet security standards. This is especially true if the basic cryptographic blocks need to be frequently used or the underlying application demands low delay.

There are ongoing efforts to reduce the resource consumption of security protocols by using more efficient underlying cryptographic primitives such as Elliptic Curve Cryptography [RFC8446]. The specification of elliptic curve X25519 [ecc25519], stream ciphers such as ChaCha [ChaCha], Diet HIP [ID-HIP-DEX], and ECC goups for IKEv2 [RFC5903] are all examples of efforts to make security protocols more resource efficient. Additionally, most modern security protocols have been revised in the last few years to enable cryptographic agility, making cryptographic primitives interchangeable. However, these improvements are only a first step in reducing the computation and communication overhead of Internet protocols. The question remains if other approaches can be applied to leverage key agreement in these heavily resource-constrained environments.

A further fundamental need refers to the limited energy budget available to IoT nodes. Careful protocol (re)design and usage is required to reduce not only the energy consumption during normal operation, but also under DoS attacks. Since the energy consumption of IoT devices differs from other device classes, judgments on the energy consumption of a particular protocol cannot be made without tailor-made IoT implementations.

5.1.2. Denial-of-Service Resistance

The tight memory and processing constraints of things naturally alleviate resource exhaustion attacks. Especially in unattended T2T communication, such attacks are difficult to notice before the service becomes unavailable (for example, because of battery or memory exhaustion). As a DoS countermeasure, DTLS, IKEv2, HIP, and Diet HIP implement return routability checks based on a cookie mechanism to delay the establishment of state at the responding host until the address of the initiating host is verified. The effectiveness of these defenses strongly depend on the routing topology of the network. Return routability checks are particularly effective if hosts cannot receive packets addressed to other hosts and if IP addresses present meaningful information as is the case in today's Internet. However, they are less effective in broadcast media or when attackers can influence the routing and addressing of hosts (for example, if hosts contribute to the routing infrastructure in ad-hoc networks and meshes).

In addition, HIP implements a puzzle mechanism that can force the initiator of a connection (and potential attacker) to solve cryptographic puzzles with variable difficulties. Puzzle-based defense mechanisms are less dependent on the network topology but perform poorly if CPU resources in the network are heterogeneous (for example, if a powerful Internet host attacks a thing). Increasing the puzzle difficulty under attack conditions can easily lead to situations where a powerful attacker can still solve the puzzle while weak IoT clients cannot and are excluded from communicating with the victim. Still, puzzle-based approaches are a viable option for sheltering IoT devices against unintended overload caused by misconfiguration or malfunctioning things.

5.1.3. End-to-end security, protocol translation, and the role of middleboxes

The term end-to-end security often has multiple interpretations. Here, we consider end-to-end security in the context end-to-end IP connectivity, from a sender to a receiver. Services such as confidentiality and integrity protection on packet data, message authentication codes or encryption are typically used to provide end-to-end security. These protection methods render the protected parts of the packets immutable as rewriting is either not possible because a) the relevant information is encrypted and inaccessible to the gateway or b) rewriting integrity-protected parts of the packet would invalidate the end-to-end integrity protection.

Protocols for constrained IoT networks are not exactly identical to their larger Internet counterparts for efficiency and performance reasons. Hence, more or less subtle differences between protocols for constrained IoT networks and Internet protocols will remain. While these differences can be bridged with protocol translators at middleboxes, they may become major obstacles if end-to-end security measures between IoT devices and Internet hosts are needed.

If access to data or messages by the middleboxes is required or acceptable, then a diverse set of approaches for handling such a scenario are available. Note that some of these approaches affect the meaning of end-to-end security in terms of integrity and confidentiality since the middleboxes will be able to either decrypt or modify partially the exchanged messages:

1. Sharing credentials with middleboxes enables them to transform (for example, decompress, convert, etc.) packets and re-apply the security measures after transformation. This method abandons end-to-end security and is only applicable to simple scenarios with a rudimentary security model.

2. Reusing the Internet wire format for IoT makes conversion between IoT and Internet protocols unnecessary. However, it can lead to poor performance in some use cases because IoT specific optimizations (for example, stateful or stateless compression) are not possible.

3. Selectively protecting vital and immutable packet parts with a message authentication code or with encryption requires a careful balance between performance and security. Otherwise this approach might either result in poor performance or poor security depending on which parts are selected for protection, where they are located in the original packet, and how they are processed. [ID-OSCORE] proposes a solution in this direction by encrypting and integrity protecting most of the message fields except those parts that a middlebox needs to read or change.

4. Homomorphic encryption techniques can be used in the middlebox to perform certain operations. However, this is limited to data processing involving arithmetic operations. Furthermore, performance of existing libraries, for example, SEAL [SEAL] is still too limited and homomorphic encryption techniques are not widely applicable yet.

5. Message authentication codes that sustain transformation can be realized by considering the order of transformation and protection (for example, by creating a signature before compression so that the gateway can decompress the packet without recalculating the signature). Such an approach enables IoT specific optimizations but is more complex and may require application-specific transformations before security is applied. Moreover, the usage of encrypted or integrity-protected data prevents middleboxes from transforming packets.

6. Mechanisms based on object security can bridge the protocol worlds, but still require that the two worlds use the same object security formats. Currently the object security format based on CBOR Object Signing and Encryption (COSE) [RFC8152] is different from JSON Object Signing and Encryption (JOSE) [RFC7520] or

Cryptographic Message Syntax (CMS) [RFC5652]. Legacy devices relying on traditional Internet protocols will need to update to the newer protocols for constrained environments to enable real end-to-end security. Furthermore, middleboxes do not have any access to the data and this approach does not prevent an attacker who is capable of modifying relevant message header fields that are not protected.

To the best of our knowledge, none of the mentioned security approaches that focus on the confidentiality and integrity of the communication exchange between two IP end-points provide the perfect solution in this problem space.

5.1.4. New network architectures and paradigm

There is a multitude of new link layer protocols that aim to address the resource-constrained nature of IoT devices. For example, the IEEE 802.11 ah [IEEE802ah] has been specified for extended range and lower energy consumption to support Internet of Things (IoT) devices. Similarly, Low-Power Wide-Area Network (LPWAN) protocols such as LoRa [lora], Sigfox [sigfox], NarrowBand IoT (NB-IoT) [nbiot] are all designed for resource-constrained devices that require long range and low bit rates. [RFC8376] provides an informational overview of the set of LPWAN technologies being considered by the IETF. It also identifies the potential gaps that exist between the needs of those technologies and the goal of running IP in such networks. While these protocols allow IoT devices to conserve energy and operate efficiently, they also add additional security challenges. For example, the relatively small MTU can make security handshakes with large X509 certificates a significant overhead. At the same time, new communication paradigms also allow IoT devices to communicate directly amongst themselves with or without support from the network. This communication paradigm is also referred to as Device-to-Device (D2D) or Machine-to-Machine (M2M) or Thing-to-Thing (T2T) communication and it is motivated by a number of features such as improved network performance, lower latency and lower energy requirements.

5.2. Bootstrapping of a Security Domain

Creating a security domain from a set of previously unassociated IoT devices is a key operation in the lifecycle of a thing in an IoT network. This aspect is further elaborated and discussed in the T2TRG draft on bootstrapping [ID-bootstrap].

5.3. Operational Challenges

After the bootstrapping phase, the system enters the operational phase. During the operational phase, things can use the state information created during the bootstrapping phase in order to exchange information securely. In this section, we discuss the security challenges during the operational phase. Note that many of the challenges discussed in Section 5.1 apply during the operational phase.

5.3.1. Group Membership and Security

Group key negotiation is an important security service for IoT communication patterns in which a thing sends some data to multiple things or data flows from multiple things towards a thing. All discussed protocols only cover unicast communication and therefore, do not focus on group-key establishment. This applies in particular to (D)TLS and IKEv2. Thus, a solution is required in this area. A potential solution might be to use the Diffie-Hellman keys - that are used in IKEv2 and HIP to setup a secure unicast link - for group Diffie-Hellman key-negotiations. However, Diffie-Hellman is a relatively heavy solution, especially if the group is large.

Symmetric and asymmetric keys can be used in group communication. Asymmetric keys have the advantage that they can provide source authentication. However, doing broadcast encryption with a single public/private key pair is also not feasible. Although a single symmetric key can be used to encrypt the communication or compute a message authentication code, it has inherent risks since the capture of a single node can compromise the key shared throughout the network. The usage of symmetric-keys also does not provide source authentication. Another factor to consider is that asymmetric cryptography is more resource-intensive than symmetric key solutions. Thus, the security risks and performance trade-offs of applying either symmetric or asymmetric keys to a given IoT use case need to be well-analyzed according to risk and usability assessments. [ID-multicast] is looking at a combination of symmetric (for encryption) and asymmetric (for authentication) in the same packet.

Conceptually, solutions that provide secure group communication at the network layer (IPsec/IKEv2, HIP/Diet HIP) may have an advantage in terms of the cryptographic overhead when compared to application-focused security solutions (TLS/ DTLS). This is due to the fact that application-focused solutions require cryptographic operations per group application, whereas network layer approaches may allow sharing secure group associations between multiple applications (for example, for neighbor discovery and routing or service discovery). Hence, implementing shared features lower in the communication stack can avoid redundant security measures. However, it is important to note that sharing security contexts among different applications involves potential security threats, e.g., if one of the applications is malicious and monitors exchanged messages or injects fake messages. In the case of OSCORE, it provides security for CoAP group communication as defined in RFC7390, i.e., based on multicast IP. If the same security association is reused for each application, then this solution does not seem to have more cryptographic overhead compared to IPsec.

Several group key solutions have been developed by the MSEC working group [WG-MSEC] of the IETF. The MIKEY architecture [RFC4738] is one example. While these solutions are specifically tailored for multicast and group broadcast applications in the Internet, they should also be considered as candidate solutions for group key agreement in IoT. The MIKEY architecture for example describes a coordinator entity that disseminates symmetric keys over pair-wise end-to-end secured channels. However, such a centralized approach may not be applicable in a distributed IoT environment, where the choice of one or several coordinators and the management of the group key is not trivial.

5.3.2. Mobility and IP Network Dynamics

It is expected that many things (for example, wearable sensors, and user devices) will be mobile in the sense that they are attached to different networks during the lifetime of a security association. Built-in mobility signaling can greatly reduce the overhead of the cryptographic protocols because unnecessary and costly re-establishments of the session (possibly including handshake and key agreement) can be avoided. IKEv2 supports host mobility with the MOBIKE [RFC4555] and [RFC4621] extension. MOBIKE refrains from applying heavyweight cryptographic extensions for mobility. However, MOBIKE mandates the use of IPsec tunnel mode which requires the transmission of an additional IP header in each packet.

HIP offers a simple yet effective mobility management by allowing hosts to signal changes to their associations [RFC8046]. However, slight adjustments might be necessary to reduce the cryptographic costs, for example, by making the public-key signatures in the mobility messages optional. Diet HIP does not define mobility yet but it is sufficiently similar to HIP and can use the same mechanisms. DTLS provides some mobility support by relying on a connection ID (CID). The use of connection IDs can provide all the mobility functionality described in [ID-Williams], except, sending the updated location. The specific need for IP-layer mobility mainly depends on the scenario in which the nodes operate. In many cases, mobility supported by means of a mobile gateway may suffice to enable mobile IoT networks, such as body sensor networks. Using message based application-layer security solutions such as OSCORE [ID-OSCORE] can also alleviate the problem of re-establishing lower-layer sessions for mobile nodes.

5.4. Secure software update and cryptographic agility

IoT devices are often expected to stay functional for several years and decades even though they might operate unattended with direct Internet connectivity. Software updates for IoT devices are therefore not only required for new functionality, but also to eliminate security vulnerabilities due to software bugs, design flaws, or deprecated algorithms. Software bugs might remain even after careful code review. Implementations of security protocols might contain (design) flaws. Cryptographic algorithms can also become insecure due to advances in cryptanalysis. Therefore, it is necessary that devices which are incapable of verifying a cryptographic signature are not exposed to the Internet (even indirectly).

Schneier [SchneierSecurity] in his essay highlights several challenges that hinder mechanisms for secure software update of IoT devices. First, there is a lack of incentives for manufactures, vendors and others on the supply chain to issue updates for their devices. Second, parts of the software running on IoT devices is simply a binary blob without any source code available. Since the complete source code is not available, no patches can be written for that piece of code. Lastly Schneier points out that even when updates are available, users generally have to manually download and install them. However, users are never alerted about security updates and at many times do not have the necessary expertise to manually administer the required updates.

The FTC staff report on Internet of Things - Privacy & Security in a Connected World [FTCreport] and the Article 29 Working Party Opinion 8/2014 on the Recent Developments on the Internet of Things [Article29] also document the challenges for secure remote software update of IoT devices. They note that even providing such a software update capability may add new vulnerabilities for constrained devices. For example, a buffer overflow vulnerability in the implementation of a software update protocol (TR69) [TR69] and an expired certificate in a hub device [wink] demonstrate how the software update process itself can introduce vulnerabilities.

Powerful IoT devices that run general purpose operating systems can make use of sophisticated software update mechanisms known from the desktop world. However, resource-constrained devices typically do not have any operating system and are often not equipped with a memory management unit or similar tools. Therefore, they might require more specialized solutions.

An important requirement for secure software and firmware updates is source authentication. Source authentication requires the resource-constrained things to implement public-key signature verification algorithms. As stated in Section 5.1.1, resource-constrained things have limited amount of computational capabilities and energy supply available which can hinder the amount and frequency of cryptographic processing that they can perform. In addition to source authentication, software updates might require confidential delivery over a secure (encrypted) channel. The complexity of broadcast encryption can force the usage of point-to-point secure links - however, this increases the duration of a software update in a large system. Alternatively, it may force the usage of solutions in which the software update is delivered to a gateway, and then distributed to the rest of the system with a network key. Sending large amounts of data that later needs to be assembled and verified over a secure channel can consume a lot of energy and computational resources. Correct scheduling of the software updates is also a crucial design challenge. For example, a user of connected light bulbs would not want them to update and restart at night. More importantly, the user would not want all the lights to update at the same time.

Software updates in IoT systems are also needed to update old and insecure cryptographic primitives. However, many IoT systems, some of which are already deployed, are not designed with provisions for cryptographic agility. For example, many devices come with a wireless radio that has an AES128 hardware co-processor. These devices solely rely on the co-processor for encrypting and authenticating messages. A software update adding support for new cryptographic algorithms implemented solely in software might not fit on these devices due to limited memory, or might drastically hinder its operational performance. This can lead to the use of old and insecure software. Therefore, it is important to account for the fact that cryptographic algorithms would need to be updated and consider the following when planning for cryptographic agility:

1. Would it be secure to use the existing cryptographic algorithms available on the device for updating with new cryptographic algorithms that are more secure?

2. Will the new software-based implementation fit on the device given the limited resources?

3. Would the normal operation of existing IoT applications on the device be severely hindered by the update?

Finally, we would like to highlight the previous and ongoing work in the area of secure software and firmware updates at the IETF. [RFC4108] describes how Cryptographic Message Syntax (CMS) [RFC5652] can be used to protect firmware packages. The IAB has also organized a workshop to understand the challenges for secure software update of IoT devices. A summary of the recommendations to the standards community derived from the discussions during that workshop have been documented [RFC8240]. A working group called Software Updates for Internet of Things (suit) [WG-SUIT] is currently working on a new version [RFC4108] to reflect the best current practices for firmware update based on experience from IoT deployments. It is specifically working on describing an IoT firmware update architecture and specifying a manifest format that contains meta-data about the firmware update package. Finally, the Trusted Execution Environment Provisioning working group [WG-TEEP] aims at developing a protocol for lifecycle management of trusted applications running on the secure area of a processor (Trusted Execution Enviornment (TEE)).

5.5. End-of-Life

Like all commercial devices, IoT devices have a given useful lifetime. The term end-of-life (EOL) is used by vendors or network operators to indicate the point of time in which they limit or end support for the IoT device. This may be planned or unplanned (for example when the manufacturer goes bankrupt, when the vendor just decides to abandon a product, or when a network operator moves to a different type of networking technology). A user should still be able to use and perhaps even update the device. This requires for some form of authorization handover.

Although this may seem far-fetched given the commercial interests and market dynamics, we have examples from the mobile world where the devices have been functional and up-to-date long after the original vendor stopped supporting the device. CyanogenMod for Android devices, and OpenWrt for home routers are two such instances where users have been able to use and update their devices even after the official EOL. Admittedly it is not easy for an average user to install and configure their devices on their own. With the deployment of millions of IoT devices, simpler mechanisms are needed to allow users to add new root-of-trusts and install software and firmware from other sources once the device is EOL.

5.6. Verifying device behavior

Users using new IoT appliances such as Internet-connected smart televisions, speakers and cameras are often unaware that these devices can undermine their privacy. Recent revelations have shown that many IoT device vendors have been collecting sensitive private data through these connected appliances with or without appropriate user warnings [cctv].

An IoT device user/owner would like to monitor and verify its operational behavior. For instance, the user might want to know if the device is connecting to the server of the manufacturer for any reason. This feature - connecting to the manufacturer's server - may be necessary in some scenarios, such as during the initial configuration of the device. However, the user should be kept aware of the data that the device is sending back to the vendor. For example, the user might want to know if his/her TV is sending data when he/she inserts a new USB stick.

Providing such information to the users in an understandable fashion is challenging. This is because IoT devices are not only resource-constrained in terms of their computational capability, but also in terms of the user interface available. Also, the network infrastructure where these devices are deployed will vary significantly from one user environment to another. Therefore, where and how this monitoring feature is implemented still remains an open question.

Manufacturer Usage Description (MUD) files [ID-MUD] are perhaps a first step towards implementation of such a monitoring service. The idea behind MUD files is relatively simple: IoT devices would disclose the location of their MUD file to the network during installation. The network can then retrieve those files, and learn about the intended behavior of the devices stated by the device manufacturer. A network monitoring service could then warn the user/owner of devices if they don't behave as expected.

Many devices and software services that automatically learn and monitor the behavior of different IoT devices in a given network are commercially available. Such monitoring devices/services can be configured by the user to limit network traffic and trigger alarms when unexpected operation of IoT devices is detected.

5.7. Testing: bug hunting and vulnerabilities

Given that IoT devices often have inadvertent vulnerabilities, both users and developers would want to perform extensive testing on their IoT devices, networks, and systems. Nonetheless, since the devices are resource-constrained and manufactured by multiple vendors, some of them very small, devices might be shipped with very limited testing, so that bugs can remain and can be exploited at a later stage. This leads to two main types of challenges:

1. It remains to be seen how the software testing and quality assurance mechanisms used from the desktop and mobile world will be applied to IoT devices to give end users the confidence that the purchased devices are robust. Bodies such as the European Cyber Security Organization (ECSO) [ECSO] are working on processes for security certification of IoT devices.

2. It is also an open question how the combination of devices from multiple vendors might actually lead to dangerous network configurations. For example, if combination of specific devices can trigger unexpected behavior. It is needless to say that the security of the whole system is limited by its weakest point.

5.8. Quantum-resistance

Many IoT systems that are being deployed today will remain operational for many years. With the advancements made in the field of quantum computers, it is possible that large-scale quantum computers are available in the future for performing cryptanalysis on existing cryptographic algorithms and cipher suites. If this happens, it will have two consequences. First, functionalities enabled by means of primitives such as RSA or ECC - namely key exchange, public-key encryption and signature - would not be secure anymore due to Shor's algorithm. Second, the security level of symmetric algorithms will decrease, for example, the security of a block cipher with a key size of b bits will only offer b/2 bits of security due to Grover's algorithm.

The above scenario becomes more urgent when we consider the so called "harvest and decrypt" attack in which an attacker can start to harvest (store) encrypted data today, before a quantum-computer is available, and decrypt it years later, once a quantum computer is available. Such "harvest and decrypt" attacks are not new and were used in the Venona project [venona-project]. Many IoT devices that are being deployed today will remain operational for a decade or even longer. During this time, digital signatures used to sign software updates might become obsolete making the secure update of IoT devices challenging.

This situation would require us to move to quantum-resistant alternatives, in particular, for those functionalities involving key exchange, public-key encryption and signatures. [ID-c2pq] describes when quantum computers may become widely available and what steps are necessary for transition to cryptographic algorithms that provide security even in presence of quantum computers. While future planning is hard, it may be a necessity in certain critical IoT deployments which are expected to last decades or more. Although increasing the key-size of the different algorithms is definitely an option, it would also incur additional computational overhead and network traffic. This would be undesirable in most scenarios. There have been recent advancements in quantum-resistant cryptography. We refer to [ETSI-GR-QSC-001] for an extensive overview of existing quantum-resistant cryptography and [RFC7696] provides guidelines for cryptographic algorithm agility.

5.9. Privacy protection

People will eventually be surrounded by hundreds of connected IoT devices. Even if the communication links are encrypted and protected, information about people might still be collected or processed for different purposes. The fact that IoT devices in the vicinity of people might enable more pervasive monitoring can negatively impact their privacy. For instance, imagine the scenario where a static presence sensor emits a packet due to the presence or absence of people in its vicinity. In such a scenario, anyone who can observe the packet, can gather critical privacy-sensitive information.

Such information about people is referred to as personal data in the European Union (EU) or Personally identifiable information (PII) in the United States (US), In particular, the General Data Protection Regulation (GDPR) [GDPR] defines personal data as: 'any information relating to an identified or identifiable natural person ('data subject'); an identifiable natural person is one who can be identified, directly or indirectly, in particular by reference to an identifier such as a name, an identification number, location data, an online identifier or to one or more factors specific to the physical, physiological, genetic, mental, economic, cultural or social identity of that natural person'.

Ziegeldorf [Ziegeldorf] defines privacy in IoT as a threefold guarantee:

1. Awareness of the privacy risks imposed by IoT devices and services. This awareness is achieved by means of transparent practices by the data controller, i.e., the entity that is providing IoT devices and/or services.

2. Individual control over the collection and processing of personal information by IoT devices and services.

3. Awareness and control of the subsequent use and dissemination of personal information by data controllers to any entity outside the subject's personal control sphere. This point implies that the data controller must be accountable for its actions on the personal information.

Based on this definition, several threats to the privacy of users have been documented [Ziegeldorf] and [RFC6973], in particular considering the IoT environment and its lifecycle:

1. Identification - refers to the identification of the users, their IoT devices, and generated data.

2. Localization - relates to the capability of locating a user and even tracking them, e.g., by tracking MAC addresses in Wi-Fi or Bluetooth.

3. Profiling - is about creating a profile of the user and their preferences.

4. Interaction - occurs when a user has been profiled and a given interaction is preferred, presenting (for example, visually) some information that discloses private information.

5. Lifecycle transitions - take place when devices are, for example, sold without properly removing private data.

6. Inventory attacks - happen if specific information about IoT devices in possession of a user is disclosed.

7. Linkage - is about when information of two of more IoT systems (or other data sets) is combined so that a broader view of the personal data captured can be created.

When IoT systems are deployed, the above issues should be considered to ensure that private data remains private. These issues are particularly challenging in environments in which multiple users with different privacy preferences interact with the same IoT devices. For example, an IoT device controlled by user A (low privacy settings) might leak private information about another user B (high privacy settings). How to deal with these threats in practice is an area of ongoing research.

5.10. Reverse engineering considerations

Many IoT devices are resource-constrained and often deployed in unattended environments. Some of these devices can also be purchased off-the-shelf or online without any credential-provisioning process. Therefore, an attacker can have direct access to the device and apply advanced techniques to retrieve information that a traditional black box model does not consider. Example of those techniques are side-channel attacks or code disassembly. By doing this, the attacker can try to retrieve data such as:

1. long term keys. These long term keys can be extracted by means of a side-channel attack or reverse engineering. If these keys are exposed, then they might be used to perform attacks on devices deployed in other locations.

2. source code. Extraction of source code might allow the attacker to determine bugs or find exploits to perform other types of attacks. The attacker might also just sell the source code.

3. proprietary algorithms. The attacker can analyze these algorithms gaining valuable know-how. The attacker can also create copies of the product (based on those proprietary algorithms) or modify the algorithms to perform more advanced attacks.

4. configuration or personal data. The attacker might be able to read personal data, e.g., healthcare data, that has been stored on a device.

One existing solution to prevent such data leaks is the use of a secure element, a tamper-resistant device that is capable of securely hosting applications and their confidential data. Another potential solution is the usage of of Physical Unclonable Function (PUFs) that serves as unique digital fingerprint of a hardware device. PUFs can also enable other functionalities such as secure key storage. Protection against such data leakage patterns is non-trivial since devices are inherently resource-constrained. An open question is whether there are any viable techniques to protect IoT devices and the data in the devices in such an adversarial model.

5.11. Trustworthy IoT Operation

Flaws in the design and implementation of IoT devices and networks can lead to security vulnerabilities. A common flaw is the use of well-known or easy-to-guess passwords for configuration of IoT devices. Many such compromised IoT devices can be found on the Internet by means of tools such as Shodan [shodan]. Once discovered, these compromised devices can be exploited at scale, for example, to launch DDoS attacks. Dyn, a major DNS , was attacked by means of a DDoS attack originating from a large IoT botnet composed of thousands of compromised IP-cameras [dyn-attack]. There are several open research questions in this area:

1. How to avoid vulnerabilities in IoT devices that can lead to large-scale attacks?

2. How to detect sophisticated attacks against IoT devices?

3. How to prevent attackers from exploiting known vulnerabilities at a large scale?

Some ideas are being explored to address this issue. One of the approaches relies on the use of Manufacturer Usage Description (MUD) files [ID-MUD]. As explained earlier, this proposal requires IoT devices to disclose the location of their MUD file to the network during installation. The network can then (i) retrieve those files, (ii) learn from the manufacturers the intended usage of the devices, for example, which services they need to access, and then (iii) create suitable filters and firewall rules.

6. Conclusions and Next Steps

This Internet Draft provides IoT security researchers, system designers and implementers with an overview of security requirements in the IP-based Internet of Things. We discuss the security threats, state-of-the-art, and challenges.

Although plenty of steps have been realized during the last few years (summarized in Section 4.1) and many organizations are publishing general recommendations (Section 4.3) describing how IoT should be secured, there are many challenges ahead that require further attention. Challenges of particular importance are bootstrapping of security, group security, secure software updates, long-term security and quantum-resistance, privacy protection, data leakage prevention - where data could be cryptographic keys, personal data, or even algorithms - and ensuring trustworthy IoT operation.

Authors of new IoT specifications and implementors need to consider how all the security challenges discussed in this draft (and those that emerge later) affect their work. The authors of IoT specifications not only need to put in a real effort towards addressing the security challenges, but also clearly documenting how the security challenges are addressed. This would reduce the chances of security vulnerabilities in the code written by implementors of those specifications.

7. Security Considerations

This entire memo deals with security issues.

8. IANA Considerations

This document contains no request to IANA.

9. Acknowledgments

We gratefully acknowledge feedback and fruitful discussion with Tobias Heer, Robert Moskowitz, Thorsten Dahm, Hannes Tschofenig, Carsten Bormann, Barry Raveendran, Ari Keranen, Goran Selander, Fred Baker, Vicent Roca, Thomas Fossati and Eliot Lear. We acknowledge the additional authors of the previous version of this document Sye Loong Keoh, Rene Hummen and Rene Struik.

10. Informative References

[Article29]
"Opinion 8/2014 on the on Recent Developments on the Internet of Things", Web http://ec.europa.eu/justice/data-protection/article-29/documentation/opinion-recommendation/files/2014/wp223_en.pdf, n.d..

[AUTO-ID] "AUTO-ID LABS", Web http://www.autoidlabs.org/, September 2010.

[BACNET] "BACnet", Web http://www.bacnet.org/, February 2011.

[BITAG] "Internet of Things (IoT) Security and Privacy Recommendations", Web http://www.bitag.org/report-internet-of-things-security-privacy-recommendations.php, n.d..

[cctv] "Backdoor In MVPower DVR Firmware Sends CCTV Stills To an Email Address In China", Web https://hardware.slashdot.org/story/16/02/17/0422259/backdoor-in-mvpower-dvr-firmware-sends-cctv-stills-to-an-email-address-in-china, n.d..

[ChaCha] Bernstein, D., "ChaCha, a variant of Salsa20", Web http://cr.yp.to/chacha/chacha-20080128.pdf, n.d..

[CSA] "Security Guidance for Early Adopters of the Internet of Things (IoT)", Web https://downloads.cloudsecurityalliance.org/whitepapers/Security_Guidance_for_Early_Adopters_of_the_Internet_of_Things.pdf, n.d..

[DALI] "DALI", Web http://www.dalibydesign.us/dali.html, February 2011.

[DCMS] "Secure by Design: Improving the cyber security of consumer Internet of Things Report", Web https://www.gov.uk/government/publications/secure-by-design, n.d..

[DHS] "Strategic Principles For Securing the Internet of Things (IoT)", Web https://www.dhs.gov/sites/default/files/publications/Strategic_Principles_for_Securing_the_Internet_of_Things-2016-1115-FINAL....pdf, n.d..

[dyn-attack] "Dyn Analysis Summary Of Friday October 21 Attack", Web https://dyn.com/blog/dyn-analysis-summary-of-friday-october-21-attack/, n.d..

[ecc25519] Bernstein, D., "Curve25519: new Diffie-Hellman speed records", Web https://cr.yp.to/ecdh/curve25519-20060209.pdf, n.d..

[ECSO] "European Cyber Security Organization", Web https://www.ecs-org.eu/, n.d..

[ENISA-ICS] "Communication network dependencies for ICS/SCADA Systems", European Union Agency For Network And Information Security , February 2017.

[ETSI-GR-QSC-001] "Quantum-Safe Cryptography (QSC);Quantum-safe algorithmic framework", European Telecommunications Standards Institute (ETSI) , June 2016.

[Fairhair] "Fairhair Alliance", Web https://www.fairhair-alliance.org/, n.d..

[FCC] "Federal Communications Comssion Response 12-05-2016", FCC , February 2016.

[FTCreport] "FTC Report on Internet of Things Urges Companies to Adopt Best Practices to Address Consumer Privacy and Security Risks", Web https://www.ftc.gov/news-events/press-releases/2015/01/ftc-report-internet-things-urges-companies-adopt-best-practices, n.d..

[GDPR] "The EU General Data Protection Regulation", Web https://www.eugdpr.org/, n.d..

[GSMAsecurity]
"GSMA IoT Security Guidelines", Web http://www.gsma.com/connectedliving/future-iot-networks/iot-security-guidelines/, n.d..

[ID-6lonfc]
Choi, Y., Hong, Y., Youn, J., Kim, D., and J. Choi, "Transmission of IPv6 Packets over Near Field Communication", draft-ietf-6lo-nfc-12 (work in progress), November 2018.

[ID-6tisch]
Thubert, P., "An Architecture for IPv6 over the TSCH mode of IEEE 802.15.4", draft-ietf-6tisch-architecture-18 (work in progress), December 2018.

[ID-acedtls]
Gerdes, S., Bergmann, O., Bormann, C., Selander, G., and L. Seitz, "Datagram Transport Layer Security (DTLS) Profile for Authentication and Authorization for Constrained Environments (ACE)", draft-ietf-ace-dtls-authorize-05 (work in progress), October 2018.

[ID-aceoauth]
Seitz, L., Selander, G., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "Authentication and Authorization for Constrained Environments (ACE) using the OAuth 2.0 Framework (ACE-OAuth)", draft-ietf-ace-oauth-authz-17 (work in progress), November 2018.

[ID-bootstrap]
Sarikaya, B., Sethi, M., and D. Garcia-Carillo, "Secure IoT Bootstrapping: A Survey", draft-sarikaya-t2trg-sbootstrapping-05 (work in progress), September 2018.

[ID-c2pq] Hoffman, P., "The Transition from Classical to Post-Quantum Cryptography", draft-hoffman-c2pq-04 (work in progress), August 2018.

[ID-Daniel]
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.

[ID-dietesp]
Migault, D., Guggemos, T., and C. Bormann, "Diet-ESP: a flexible and compressed format for IPsec/ESP", draft-mglt-6lo-diet-esp-02 (work in progress), July 2016.

[ID-HIP-DEX]
Moskowitz, R., "HIP Diet EXchange (DEX)", draft-moskowitz-hip-rg-dex-06 (work in progress), May 2012.

[ID-Moore]
Moore, K., Barnes, R., and H. Tschofenig, "Best Current Practices for Securing Internet of Things (IoT) Devices", draft-moore-iot-security-bcp-01 (work in progress), July 2017.

[ID-MUD] Lear, E., Droms, R., and D. Romascanu, "Manufacturer Usage Description Specification", draft-ietf-opsawg-mud-25 (work in progress), June 2018.

[ID-multicast]
Tiloca, M., Selander, G., Palombini, F., and J. Park, "Group OSCORE - Secure Group Communication for CoAP", draft-ietf-core-oscore-groupcomm-03 (work in progress), October 2018.

[ID-OSCORE]
Selander, G., Mattsson, J., Palombini, F., and L. Seitz, "Object Security for Constrained RESTful Environments (OSCORE)", draft-ietf-core-object-security-15 (work in progress), August 2018.

[ID-rd] Shelby, Z., Koster, M., Bormann, C., Stok, P., and C. Amsuess, "CoRE Resource Directory", draft-ietf-core-resource-directory-17 (work in progress), October 2018.

[ID-Williams]
Williams, M. and J. Barrett, "Mobile DTLS", draft-barrett-mobile-dtls-00 (work in progress), March 2009.

[IEEE802ah]
"Status of Project IEEE 802.11ah, IEEE P802.11- Task Group AH-Meeting Update.", Web http://www.ieee802.org/11/Reports/tgah_update.htm, n.d..

[IIoT] "Industrial Internet Consortium", Web http://www.iiconsortium.org/, n.d..

[IoTSecFoundation]
"Establishing Principles for Internet of Things Security", Web https://iotsecurityfoundation.org/establishing-principles-for-internet-of-things-security/, n.d..

[IPSO] "IPSO Alliance", Web http://www.ipso-alliance.org, n.d..

[ISOC-OTA]
"Internet Society’s Online Trust Alliance (OTA)", Web https://www.internetsociety.org/ota/, n.d..

[lora] "LoRa - Wide Area Networks for IoT", Web https://www.lora-alliance.org/, n.d..

[LWM2M] "OMA LWM2M", Web http://openmobilealliance.org/iot/lightweight-m2m-lwm2m, n.d..

[mirai] Kolias, C., Kambourakis, G., Stavrou, A., and J. Voas,, "DDoS in the IoT: Mirai and Other Botnets", IEEE Computer , 2017.

[nbiot] "NarrowBand IoT", Web http://www.3gpp.org/ftp/tsg_ran/TSG_RAN/TSGR_69/Docs/RP-151621.zip, n.d..

[NHTSA] "Cybersecurity Best Practices for Modern Vehicles", Web https://www.nhtsa.gov/staticfiles/nvs/pdf/812333_CybersecurityForModernVehicles.pdf, n.d..

[NIST-Guide]
Ross, R., McEvilley, M., and J. Oren, "Systems Security Engineering", Web http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-160.pdf, n.d..

[nist-lightweight-project]
"NIST lightweight Project", Web www.nist.gov/programs-projects/lightweight-cryptography, www.nist.gov/sites/default/files/documents/2016/10/17/sonmez-turan-presentation-lwc2016.pdf, n.d..

[NISTSP800-122]
Erika McCallister, ., Tim Grance, ., and . Karen Scarfone, "NIST SP800-122 - Guide to Protecting the Confidentiality of Personally Identifiable Information", Web https://nvlpubs.nist.gov/nistpubs/legacy/sp/nistspecialpublication800-122.pdf, n.d..

[NISTSP800-30r1]
"NIST SP 800-30r1 - Guide for Conducting Risk Assessments", Web https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-30r1.pdf, n.d..

[NISTSP800-34r1]
Marianne Swanson, ., Pauline Bowen, ., Amy Wohl Phillips, ., Dean Gallup, ., and . David Lynes, "NIST SP800-34r1 - Contingency Planning Guide for Federal Information Systems", Web https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-34r1.pdf, n.d..

[OCF] "Open Connectivity Foundation", Web https://openconnectivity.org/, n.d..

[OneM2M] "OneM2M", Web http://www.onem2m.org/, n.d..

[OWASP] "IoT Security Guidance", Web https://www.owasp.org/index.php/IoT_Security_Guidance, n.d..

[RFC2818] Rescorla, E., "HTTP Over TLS", RFC 2818, DOI 10.17487/RFC2818, May 2000, <https://www.rfc-editor.org/info/rfc2818>.

[RFC3748] Aboba, B., Blunk, L., Vollbrecht, J., Carlson, J., and H. Levkowetz, Ed., "Extensible Authentication Protocol (EAP)", RFC 3748, DOI 10.17487/RFC3748, June 2004, <https://www.rfc-editor.org/info/rfc3748>.

[RFC3756] Nikander, P., Ed., Kempf, J., and E. Nordmark, "IPv6 Neighbor Discovery (ND) Trust Models and Threats", RFC 3756, DOI 10.17487/RFC3756, May 2004, <https://www.rfc-editor.org/info/rfc3756>.

[RFC3833] Atkins, D. and R. Austein, "Threat Analysis of the Domain Name System (DNS)", RFC 3833, DOI 10.17487/RFC3833, August 2004, <https://www.rfc-editor.org/info/rfc3833>.

[RFC4016] Parthasarathy, M., "Protocol for Carrying Authentication and Network Access (PANA) Threat Analysis and Security Requirements", RFC 4016, DOI 10.17487/RFC4016, March 2005, <https://www.rfc-editor.org/info/rfc4016>.

[RFC4108] Housley, R., "Using Cryptographic Message Syntax (CMS) to Protect Firmware Packages", RFC 4108, DOI 10.17487/RFC4108, August 2005, <https://www.rfc-editor.org/info/rfc4108>.

[RFC4120] Neuman, C., Yu, T., Hartman, S., and K. Raeburn, "The Kerberos Network Authentication Service (V5)", RFC 4120, DOI 10.17487/RFC4120, July 2005, <https://www.rfc-editor.org/info/rfc4120>.

[RFC4422] Melnikov, A., Ed. and K. Zeilenga, Ed., "Simple Authentication and Security Layer (SASL)", RFC 4422, DOI 10.17487/RFC4422, June 2006, <https://www.rfc-editor.org/info/rfc4422>.

[RFC4555] Eronen, P., "IKEv2 Mobility and Multihoming Protocol (MOBIKE)", RFC 4555, DOI 10.17487/RFC4555, June 2006, <https://www.rfc-editor.org/info/rfc4555>.

[RFC4621] Kivinen, T. and H. Tschofenig, "Design of the IKEv2 Mobility and Multihoming (MOBIKE) Protocol", RFC 4621, DOI 10.17487/RFC4621, August 2006, <https://www.rfc-editor.org/info/rfc4621>.

[RFC4738] Ignjatic, D., Dondeti, L., Audet, F., and P. Lin, "MIKEY-RSA-R: An Additional Mode of Key Distribution in Multimedia Internet KEYing (MIKEY)", RFC 4738, DOI 10.17487/RFC4738, November 2006, <https://www.rfc-editor.org/info/rfc4738>.

[RFC4919] Kushalnagar, N., Montenegro, G., and C. Schumacher, "IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs): Overview, Assumptions, Problem Statement, and Goals", RFC 4919, DOI 10.17487/RFC4919, August 2007, <https://www.rfc-editor.org/info/rfc4919>.

[RFC4944] Montenegro, G., Kushalnagar, N., Hui, J., and D. Culler, "Transmission of IPv6 Packets over IEEE 802.15.4 Networks", RFC 4944, DOI 10.17487/RFC4944, September 2007, <https://www.rfc-editor.org/info/rfc4944>.

[RFC5191] Forsberg, D., Ohba, Y., Ed., Patil, B., Tschofenig, H., and A. Yegin, "Protocol for Carrying Authentication for Network Access (PANA)", RFC 5191, DOI 10.17487/RFC5191, May 2008, <https://www.rfc-editor.org/info/rfc5191>.

[RFC5652] Housley, R., "Cryptographic Message Syntax (CMS)", STD 70, RFC 5652, DOI 10.17487/RFC5652, September 2009, <https://www.rfc-editor.org/info/rfc5652>.

[RFC5713] Moustafa, H., Tschofenig, H., and S. De Cnodder, "Security Threats and Security Requirements for the Access Node Control Protocol (ANCP)", RFC 5713, DOI 10.17487/RFC5713, January 2010, <https://www.rfc-editor.org/info/rfc5713>.

[RFC5903] Fu, D. and J. Solinas, "Elliptic Curve Groups modulo a Prime (ECP Groups) for IKE and IKEv2", RFC 5903, DOI 10.17487/RFC5903, June 2010, <https://www.rfc-editor.org/info/rfc5903>.

[RFC6272] Baker, F. and D. Meyer, "Internet Protocols for the Smart Grid", RFC 6272, DOI 10.17487/RFC6272, June 2011, <https://www.rfc-editor.org/info/rfc6272>.

[RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, DOI 10.17487/RFC6347, January 2012, <https://www.rfc-editor.org/info/rfc6347>.

[RFC6550] Winter, T., Ed., Thubert, P., Ed., Brandt, A., Hui, J., Kelsey, R., Levis, P., Pister, K., Struik, R., Vasseur, JP., and R. Alexander, "RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks", RFC 6550, DOI 10.17487/RFC6550, March 2012, <https://www.rfc-editor.org/info/rfc6550>.

[RFC6551] Vasseur, JP., Ed., Kim, M., Ed., Pister, K., Dejean, N., and D. Barthel, "Routing Metrics Used for Path Calculation in Low-Power and Lossy Networks", RFC 6551, DOI 10.17487/RFC6551, March 2012, <https://www.rfc-editor.org/info/rfc6551>.

[RFC6568] Kim, E., Kaspar, D., and JP. Vasseur, "Design and Application Spaces for IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs)", RFC 6568, DOI 10.17487/RFC6568, April 2012, <https://www.rfc-editor.org/info/rfc6568>.

[RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, DOI 10.17487/RFC6690, August 2012, <https://www.rfc-editor.org/info/rfc6690>.

[RFC6749] Hardt, D., Ed., "The OAuth 2.0 Authorization Framework", RFC 6749, DOI 10.17487/RFC6749, October 2012, <https://www.rfc-editor.org/info/rfc6749>.

[RFC6973] Cooper, A., Tschofenig, H., Aboba, B., Peterson, J., Morris, J., Hansen, M., and R. Smith, "Privacy Considerations for Internet Protocols", RFC 6973, DOI 10.17487/RFC6973, July 2013, <https://www.rfc-editor.org/info/rfc6973>.

[RFC7049] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", RFC 7049, DOI 10.17487/RFC7049, October 2013, <https://www.rfc-editor.org/info/rfc7049>.

[RFC7228] Bormann, C., Ersue, M., and A. Keranen, "Terminology for Constrained-Node Networks", RFC 7228, DOI 10.17487/RFC7228, May 2014, <https://www.rfc-editor.org/info/rfc7228>.

[RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <https://www.rfc-editor.org/info/rfc7252>.

[RFC7296] Kaufman, C., Hoffman, P., Nir, Y., Eronen, P., and T. Kivinen, "Internet Key Exchange Protocol Version 2 (IKEv2)", STD 79, RFC 7296, DOI 10.17487/RFC7296, October 2014, <https://www.rfc-editor.org/info/rfc7296>.

[RFC7401] Moskowitz, R., Ed., Heer, T., Jokela, P., and T. Henderson, "Host Identity Protocol Version 2 (HIPv2)", RFC 7401, DOI 10.17487/RFC7401, April 2015, <https://www.rfc-editor.org/info/rfc7401>.

[RFC7515] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Signature (JWS)", RFC 7515, DOI 10.17487/RFC7515, May 2015, <https://www.rfc-editor.org/info/rfc7515>.

[RFC7516] Jones, M. and J. Hildebrand, "JSON Web Encryption (JWE)", RFC 7516, DOI 10.17487/RFC7516, May 2015, <https://www.rfc-editor.org/info/rfc7516>.

[RFC7517] Jones, M., "JSON Web Key (JWK)", RFC 7517, DOI 10.17487/RFC7517, May 2015, <https://www.rfc-editor.org/info/rfc7517>.

[RFC7519] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Token (JWT)", RFC 7519, DOI 10.17487/RFC7519, May 2015, <https://www.rfc-editor.org/info/rfc7519>.

[RFC7520] Miller, M., "Examples of Protecting Content Using JSON Object Signing and Encryption (JOSE)", RFC 7520, DOI 10.17487/RFC7520, May 2015, <https://www.rfc-editor.org/info/rfc7520>.

[RFC7668] Nieminen, J., Savolainen, T., Isomaki, M., Patil, B., Shelby, Z., and C. Gomez, "IPv6 over BLUETOOTH(R) Low Energy", RFC 7668, DOI 10.17487/RFC7668, October 2015, <https://www.rfc-editor.org/info/rfc7668>.

[RFC7696] Housley, R., "Guidelines for Cryptographic Algorithm Agility and Selecting Mandatory-to-Implement Algorithms", BCP 201, RFC 7696, DOI 10.17487/RFC7696, November 2015, <https://www.rfc-editor.org/info/rfc7696>.

[RFC7744] Seitz, L., Ed., Gerdes, S., Ed., Selander, G., Mani, M., and S. Kumar, "Use Cases for Authentication and Authorization in Constrained Environments", RFC 7744, DOI 10.17487/RFC7744, January 2016, <https://www.rfc-editor.org/info/rfc7744>.

[RFC7815] Kivinen, T., "Minimal Internet Key Exchange Version 2 (IKEv2) Initiator Implementation", RFC 7815, DOI 10.17487/RFC7815, March 2016, <https://www.rfc-editor.org/info/rfc7815>.

[RFC7925] Tschofenig, H., Ed. and T. Fossati, "Transport Layer Security (TLS) / Datagram Transport Layer Security (DTLS) Profiles for the Internet of Things", RFC 7925, DOI 10.17487/RFC7925, July 2016, <https://www.rfc-editor.org/info/rfc7925>.

[RFC8046] Henderson, T., Ed., Vogt, C., and J. Arkko, "Host Mobility with the Host Identity Protocol", RFC 8046, DOI 10.17487/RFC8046, February 2017, <https://www.rfc-editor.org/info/rfc8046>.

[RFC8105] Mariager, P., Petersen, J., Ed., Shelby, Z., Van de Logt, M., and D. Barthel, "Transmission of IPv6 Packets over Digital Enhanced Cordless Telecommunications (DECT) Ultra Low Energy (ULE)", RFC 8105, DOI 10.17487/RFC8105, May 2017, <https://www.rfc-editor.org/info/rfc8105>.

[RFC8152] Schaad, J., "CBOR Object Signing and Encryption (COSE)", RFC 8152, DOI 10.17487/RFC8152, July 2017, <https://www.rfc-editor.org/info/rfc8152>.

[RFC8240] Tschofenig, H. and S. Farrell, "Report from the Internet of Things Software Update (IoTSU) Workshop 2016", RFC 8240, DOI 10.17487/RFC8240, September 2017, <https://www.rfc-editor.org/info/rfc8240>.

[RFC8259] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", STD 90, RFC 8259, DOI 10.17487/RFC8259, December 2017, <https://www.rfc-editor.org/info/rfc8259>.

[RFC8376] Farrell, S., Ed., "Low-Power Wide Area Network (LPWAN) Overview", RFC 8376, DOI 10.17487/RFC8376, May 2018, <https://www.rfc-editor.org/info/rfc8376>.

[RFC8428] Jennings, C., Shelby, Z., Arkko, J., Keranen, A., and C. Bormann, "Sensor Measurement Lists (SenML)", RFC 8428, DOI 10.17487/RFC8428, August 2018, <https://www.rfc-editor.org/info/rfc8428>.

[RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <https://www.rfc-editor.org/info/rfc8446>.

[RG-T2TRG] "IRTF Thing-to-Thing (T2TRG) Research Group", Web https://datatracker.ietf.org/rg/t2trg/charter/, n.d..

[SchneierSecurity] "The Internet of Things Is Wildly Insecure--And Often Unpatchable", Web https://www.schneier.com/essays/archives/2014/01/the_internet_of_thin.html, n.d..

[SEAL] "Simple Encrypted Arithmetic Library - SEAL", Web https://www.microsoft.com/en-us/research/publication/simple-encrypted-arithmetic-library-seal-v2-0/, n.d..

[shodan] "Shodan", Web https://www.shodan.io/, n.d..

[sigfox] "Sigfox - The Global Communications Service Provider for the Internet of Things (IoT)", Web https://www.sigfox.com/, n.d..

[Thread] "Thread Group", Web http://threadgroup.org/, n.d..

[TR69] "Too Many Cooks - Exploiting the Internet-of-TR-069-Things", Web https://media.ccc.de/v/31c3_-_6166_-_en_-_saal_6_-_201412282145_-_too_many_cooks_-_exploiting_the_internet-of-tr-069-things_-_lior_oppenheim_-_shahar_tal, n.d..

[venona-project] "Venona Project", Web https://www.nsa.gov/news-features/declassified-documents/venona/index.shtml, n.d..

[WG-6lo] "IETF IPv6 over Networks of Resource-constrained Nodes (6lo) Working Group", Web https://datatracker.ietf.org/wg/6lo/charter/, n.d..

[WG-6LoWPAN] "IETF IPv6 over Low power WPAN (6lowpan) Working Group", Web http://tools.ietf.org/wg/6lowpan/, n.d..

[WG-ACE] "IETF Authentication and Authorization for Constrained Environments (ACE) Working Group", Web https://datatracker.ietf.org/wg/ace/charter/, n.d..

[WG-ACME] "Automated Certificate Management Environment Working Group", Web https://datatracker.ietf.org/wg/acme/about/, n.d..

[WG-CoRE] "IETF Constrained RESTful Environment (CoRE) Working Group", Web https://datatracker.ietf.org/wg/core/charter/, n.d..

[WG-LPWAN] "IETF Low Power Wide-Area Networks Working Group", Web https://datatracker.ietf.org/wg/lpwan/, n.d..

[WG-LWIG] "IETF Light-Weight Implementation Guidance (LWIG) Working Group", Web https://datatracker.ietf.org/wg/lwig/charter/, n.d..

[WG-MSEC] "IETF MSEC Working Group", Web https://datatracker.ietf.org/wg/msec/, n.d..

[WG-SUIT] "IETF Software Updates for Internet of Things (suit)", Web https://datatracker.ietf.org/group/suit/about/, n.d..

[WG-TEEP] "IETF Trusted Execution Environment Provisioning (teep)", Web https://datatracker.ietf.org/wg/teep/about/, n.d..

[wink] "Wink's Outage Shows Us How Frustrating Smart Homes Could Be", Web http://www.wired.com/2015/04/smart-home-headaches/, n.d..

[ZB] "ZigBee Alliance", Web http://www.zigbee.org/, February 2011.

[Ziegeldorf]
Ziegeldorf, J., Garcia-Morchon, O., and K. Wehrle,, "Privacy in the Internet of Things: Threats and Challenges", Security and Communication Networks - Special Issue on Security in a Completely Interconnected World , 2013.

Authors' Addresses

Oscar Garcia-Morchon
Philips IP&S
High Tech Campus 5
Eindhoven, 5656 AA
The Netherlands

Email: oscar.garcia-morchon@philips.com

Sandeep S. Kumar
Philips Research
High Tech Campus
Eindhoven, 5656 AA
The Netherlands

Email: sandeep.kumar@philips.com

Mohit Sethi
Ericsson
Hirsalantie 11
Jorvas, 02420
Finland

Email: mohit@piuha.net

Network Working Group A. Keranen
Internet-Draft Ericsson
Intended status: Informational M. Kovatsch
Expires: November 5, 2018 ETH Zurich
K. Hartke
Universitaet Bremen TZI
May 04, 2018

RESTful Design for Internet of Things Systems
draft-irtf-t2trg-rest-iot-01

Abstract

This document gives guidance for designing Internet of Things (IoT) systems that follow the principles of the Representational State Transfer (REST) architectural style.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

The Representational State Transfer (REST) architectural style [REST] is a set of guidelines and best practices for building distributed hypermedia systems. At its core is a set of constraints, which when fulfilled enable desirable properties for distributed software systems such as scalability and modifiability. When REST principles are applied to the design of a system, the result is often called RESTful and in particular an API following these principles is called a RESTful API.

Different protocols can be used with RESTful systems, but at the time of writing the most common protocols are HTTP [RFC7230] and CoAP [RFC7252]. Since RESTful APIs are often simple and lightweight, they are a good fit for various IoT applications. The goal of this document is to give basic guidance for designing RESTful systems and APIs for IoT applications and give pointers for more information. Design of a good RESTful IoT system has naturally many commonalities with other Web systems. Compared to other systems, the key characteristics of many IoT systems include:

- data formats, interaction patterns, and other mechanisms that minimize, or preferably avoid, the need for human interaction

- preference for compact and simple data formats to facilitate efficient transfer over (often) constrained networks and lightweight processing in constrained nodes

2. Terminology

This section explains some of the common terminology that is used in the context of RESTful design for IoT systems. For terminology of constrained nodes and networks, see [RFC7228].

Cache: A local store of response messages and the subsystem that controls storage, retrieval, and deletion of messages in it.

Client: A node that sends requests to servers and receives responses. In RESTful IoT systems it’s common for nodes to have more than one role (e.g., both server and client; see Section 3.1).

Client State: The state kept by a client between requests. This typically includes the currently processed representation, the set of active requests, the history of requests, bookmarks (URIs stored for later retrieval), and application-specific state (e.g., local variables). (Note that this is called "Application State" in [REST], which has some ambiguity in modern (IoT) systems where the overall state of the distributed application (i.e., application state) is reflected in the union of all Client States and Resource States of all clients and servers involved.)

Content Negotiation: The practice of determining the "best" representation for a client when examining the current state of a resource. The most common forms of content negotiation are Proactive Content Negotiation and Reactive Content Negotiation.

Form: A hypermedia control that enables a client to change the state of a resource or to construct a query locally.

Forward Proxy: An intermediary that is selected by a client, usually via local configuration rules, and that can be tasked to make requests on behalf of the client. This may be useful, for example, when the client lacks the capability to make the request itself or to service the response from a cache in order to reduce response time, network bandwidth, and energy consumption.

Gateway: A reverse proxy that provides an interface to a non-RESTful system such as legacy systems or alternative technologies such as Bluetooth ATT/GATT. See also "Reverse Proxy".

Hypermedia Control: A component, such as a link or a form, embedded in a representation that identifies a resource for future hypermedia interactions. If the client engages in an interaction with the identified resource, the result may be a change to resource state and/or client state.

Idempotent Method: A method where multiple identical requests with that method lead to the same visible resource state as a single such request.

Link: A hypermedia control that enables a client to navigate between resources and thereby change the client state.

Link Relation Type: An identifier that describes how the link target resource relates to the current resource (see [RFC5988]).

Media Type: A string such as "text/html" or "application/json" that is used to label representations so that it is known how the representation should be interpreted and how it is encoded.

Method: An operation associated with a resource. Common methods include GET, PUT, POST, and DELETE (see Section 3.5 for details).

Origin Server: A server that is the definitive source for representations of its resources and the ultimate recipient of any request that intends to modify its resources. In contrast, intermediaries (such as proxies caching a representation) can assume the role of a server, but are not the source for representations as these are acquired from the origin server.

Proactive Content Negotiation: A content negotiation mechanism where the server selects a representation based on the expressed preference of the client. For example, an IoT application could send a request to a sensor with preferred media type "application/senml+json".

Reactive Content Negotiation: A content negotiation mechanism where the client selects a representation from a list of available representations. The list may, for example, be included by a server in an initial response. If the user agent is not satisfied by the initial response representation, it can request one or more of the alternative representations, selected based on metadata (e.g., available media types) included in the response.

Representation: A serialization that represents the current or intended state of a resource and that can be transferred between clients and servers. REST requires representations to be self-describing, meaning that there must be metadata that allows peers to understand which representation format is used. Depending on the protocol needs and capabilities, there can be additional metadata that is transmitted along with the representation.

Representation Format: A set of rules for serializing resource state. On the Web, the most prevalent representation format is HTML. Other common formats include plain text and formats based on JSON [RFC7159], XML, or RDF. Within IoT systems, often compact formats based on JSON, CBOR [RFC7049], and EXI [W3C.REC-exi-20110310] are used.

Representational State Transfer (REST): An architectural style for Internet-scale distributed hypermedia systems.

Resource: An item of interest identified by a URI. Anything that can be named can be a resource. A resource often encapsulates a piece of state in a system. Typical resources in an IoT system can be, e.g., a sensor, the current value of a sensor, the location of a device, or the current state of an actuator.

Resource State: A model of a resource’s possible states that is represented in a supported representation type, typically a media type. Resources can change state because of REST interactions with them, or they can change state for reasons outside of the REST model.

Resource Type: An identifier that annotates the application-semantics of a resource (see Section 3.1 of [RFC6690]).

Reverse Proxy: An intermediary that appears as a server towards the client but satisfies the requests by forwarding them to the actual server (possibly via one or more other intermediaries). A reverse proxy is often used to encapsulate legacy services, to improve server performance through caching, and to enable load balancing across multiple machines.

Safe Method: A method that does not result in any state change on the origin server when applied to a resource.

Server: A node that listens for requests, performs the requested operation and sends responses back to the clients.

Uniform Resource Identifier (URI): A global identifier for resources. See Section 3.3 for more details.

3. Basics

3.1. Architecture

The components of a RESTful system are assigned one or both of two roles: client or server. Note that the terms "client" and "server" refer only to the roles that the nodes assume for a particular message exchange. The same node might act as a client in some communications and a server in others. Classic user agents (e.g., Web browsers) are always in the client role and have the initiative to issue requests. Origin servers always have the server role and govern over the resources they host.

```
 ________                       _________
|        |                     |         |
| User  (C)-------------------(S) Origin |
| Agent  |                     |  Server |
|________|                     |_________|
(Browser)                      (Web Server)
```

Figure 1: Client-Server Communication

Intermediaries (such as forward proxies, reverse proxies, and gateways) implement both roles, but only forward requests to other intermediaries or origin servers. They can also translate requests to different protocols, for instance, as CoAP-HTTP cross-proxies.

```
        ________       __________                        _________
       |        |     |          |                      |         |
       | User  (C)---(S) Inter- (C)--------------------(S) Origin |
       | Agent  |     |  mediary |                      |  Server |
       |________|     |__________|                      |_________|
       (Browser)     (Forward Proxy)                    (Web Server)
```

Figure 2: Communication with Forward Proxy

Reverse proxies are usually imposed by the origin server. In addition to the features of a forward proxy, they can also provide an interface for non-RESTful services such as legacy systems or alternative technologies such as Bluetooth ATT/GATT. In this case, reverse proxies are usually called gateways. This property is enabled by the Layered System constraint of REST, which says that a client cannot see beyond the server it is connected to (i.e., it is left unaware of the protocol/paradigm change).

```
       ________                        __________       _________
      |        |                      |          |     |         |
      | User  (C)--------------------(S) Inter- (x)---(x) Origin |
      | Agent  |                      |  mediary |     |  Server |
      |________|                      |__________|     |_________|
      (Browser)                        (Gateway)      (Legacy System)
```

Figure 3: Communication with Reverse Proxy

Nodes in IoT systems often implement both roles. Unlike intermediaries, however, they can take the initiative as a client (e.g., to register with a directory, such as CoRE Resource Directory [I-D.ietf-core-resource-directory], or to interact with another thing) and act as origin server at the same time (e.g., to serve sensor values or provide an actuator interface).

```
      ________                                         _________
     |        |                                       |         |
     | Thing (C)-------------------------------------(S) Origin |
     |       (S)                                      |  Server |
     |________| \                                     |_________|
      (Sensor)   \   ________                     (Resource Directory)
                  \ |        |
                   (C) Thing |
                    |________|
                   (Controller)
```

Figure 4: Constrained RESTful environments

3.2. System design

When designing a RESTful system, the primary effort goes into modeling the state of the distributed application and assigning it to the different components (i.e., clients and servers). How clients can navigate through the resources and modify state to achieve their goals is defined through hypermedia controls, that is, links and forms. Hypermedia controls span a kind of a state machine where the nodes are resources and the transitions are links or forms. Clients run this state machine (i.e., the application) by retrieving representations, processing the data, and following the included hypermedia controls. In REST, remote state is changed by submitting forms. This is usually done by retrieving the current state, modifying the state on the client side, and transferring the new state to the server in the form of new representations - rather than calling a service and modifying the state on the server side.

Client state encompasses the current state of the described state machine and the possible next transitions derived from the hypermedia controls within the currently processed representation (see Section 2). Furthermore, clients can have part of the state of the distributed application in local variables.

Resource state includes the more persistent data of an application (i.e., independent of individual clients). This can be static data such as device descriptions, persistent data such as system configurations, but also dynamic data such as the current value of a sensor on a thing.

It is important to distinguish between "client state" and "resource state" and keep them separate. Following the Stateless constraint, the client state must be kept only on clients. That is, there is no establishment of shared information about past and future interactions between client and server (usually called a session). On the one hand, this makes requests a bit more verbose since every request must contain all the information necessary to process it. On the other hand, this makes servers efficient and scalable, since they do not have to keep any state about their clients. Requests can easily be distributed over multiple worker threads or server instances. For IoT systems, this constraint lowers the memory requirements for server implementations, which is particularly important for constrained servers (e.g., sensor nodes) and servers serving large amount of clients (e.g., Resource Directory).

3.3. Uniform Resource Identifiers (URIs)

An important part of RESTful API design is to model the system as a set of resources whose state can be retrieved and/or modified and where resources can be potentially also created and/or deleted.

Uniform Resource Identifiers (URIs) are used to indicate a resource for interaction, to reference a resource from another resource, to advertise or bookmark a resource, or to index a resource by search engines.

```
  foo://example.com:8042/over/there?name=ferret#nose
  \_/   \______________/\_________/ \_________/ \__/
   |           |            |            |        |
scheme     authority       path        query   fragment
```

A URI is a sequence of characters that matches the syntax defined in [RFC3986]. It consists of a hierarchical sequence of five components: scheme, authority, path, query, and fragment (from most significant to least significant). A scheme creates a namespace for resources and defines how the following components identify a resource within that namespace. The authority identifies an entity that governs part of the namespace, such as the server "www.example.org" in the "http" scheme. A host name (e.g., a fully qualified domain name) or an IP address, potentially followed by a transport layer port number, are usually used in the authority component for the "http" and "coap" schemes. The path and query contain data to identify a resource within the scope of the URI's scheme and naming authority. The fragment allows to refer to some portion of the resource, such as a Record in a SenML Pack. However, fragments are processed only at client side and not sent on the wire. [RFC7320] provides more details on URI design and ownership with best current practices for establishing URI structures, conventions, and formats.

For RESTful IoT applications, typical schemes include "https", "coaps", "http", and "coap". These refer to HTTP and CoAP, with and without Transport Layer Security (TLS) [RFC5246]. (CoAP uses Datagram TLS (DTLS) [RFC6347], the variant of TLS for UDP.) These four schemes also provide means for locating the resource; using the HTTP protocol for "http" and "https", and with the CoAP protocol for "coap" and "coaps". If the scheme is different for two URIs (e.g., "coap" vs. "coaps"), it is important to note that even if the rest of the URI is identical, these are two different resources, in two distinct namespaces.

The query parameters can be used to parametrize the resource. For example, a GET request may use query parameters to request the server to send only certain kind data of the resource (i.e., filtering the response). Query parameters in PUT and POST requests do not have such established semantics and are not commonly used. Whether the order of the query parameters matters in URIs is unspecified and they can be re-ordered e.g., by proxies. Therefore applications should not rely on their order; see Section 3.3 of [RFC6943] for more details.

3.4. Representations

Clients can retrieve the resource state from an origin server or manipulate resource state on the origin server by transferring resource representations. Resource representations have a media type that tells how the representation should be interpreted by identifying the representation format used.

Typical media types for IoT systems include:

- o "text/plain" for simple UTF-8 text
- o "application/octet-stream" for arbitrary binary data
- o "application/json" for the JSON format [RFC7159]
- o "application/senml+json" [I-D.ietf-core-senml] for Sensor Markup Language (SenML) formatted data
- o "application/cbor" for CBOR [RFC7049]
- o "application/exi" for EXI [W3C.REC-exi-20110310]

A full list of registered Internet Media Types is available at the IANA registry [IANA-media-types] and numerical media types registered for use with CoAP are listed at CoAP Content-Formats IANA registry [IANA-CoAP-media].

3.5. HTTP/CoAP Methods

Section 4.3 of [RFC7231] defines the set of methods in HTTP; Section 5.8 of [RFC7252] defines the set of methods in CoAP. As part of the Uniform Interface constraint, each method can have certain properties that give guarantees to clients.

Safe methods do not cause any state change on the origin server when applied to a resource. For example, the GET method only returns a representation of the resource state but does not change the resource. Thus, it is always safe for a client to retrieve a representation without affecting server-side state.

Idempotent methods can be applied multiple times to the same resource while causing the same visible resource state as a single such request. For example, the PUT method replaces the state of a resource with a new state; replacing the state multiple times with the same new state still results in the same state for the resource. However, the response from the server can be different when the same idempotent method is used multiple times. For example when DELETE is used twice on an existing resource, the first request would remove the association and return success acknowledgement whereas the second request would likely result in error response due to non-existing resource.

The following lists the most relevant methods and gives a short explanation of their semantics.

3.5.1. GET

The GET method requests a current representation for the target resource, while the origin server must ensure that there are no side-effects on the resource state. Only the origin server needs to know how each of its resource identifiers corresponds to an implementation and how each implementation manages to select and send a current representation of the target resource in a response to GET.

A payload within a GET request message has no defined semantics.

The GET method is safe and idempotent.

3.5.2. POST

The POST method requests that the target resource process the representation enclosed in the request according to the resource's own specific semantics.

If one or more resources has been created on the origin server as a result of successfully processing a POST request, the origin server sends a 201 (Created) response containing a Location header field (with HTTP) or Location-Path and/or Location-Query Options (with CoAP) that provide an identifier for the resource created. The server also includes a representation that describes the status of the request while referring to the new resource(s).

The POST method is not safe nor idempotent.

3.5.3. PUT

The PUT method requests that the state of the target resource be created or replaced with the state defined by the representation enclosed in the request message payload. A successful PUT of a given representation would suggest that a subsequent GET on that same target resource will result in an equivalent representation being sent.

The fundamental difference between the POST and PUT methods is highlighted by the different intent for the enclosed representation. The target resource in a POST request is intended to handle the enclosed representation according to the resource's own semantics, whereas the enclosed representation in a PUT request is defined as replacing the state of the target resource. Hence, the intent of PUT is idempotent and visible to intermediaries, even though the exact effect is only known by the origin server.

The PUT method is not safe, but is idempotent.

3.5.4. DELETE

The DELETE method requests that the origin server remove the association between the target resource and its current functionality.

If the target resource has one or more current representations, they might or might not be destroyed by the origin server, and the associated storage might or might not be reclaimed, depending entirely on the nature of the resource and its implementation by the origin server.

The DELETE method is not safe, but is idempotent.

3.6. HTTP/CoAP Status/Response Codes

Section 6 of [RFC7231] defines a set of Status Codes in HTTP that are used by application to indicate whether a request was understood and satisfied, and how to interpret the answer. Similarly, Section 5.9 of [RFC7252] defines the set of Response Codes in CoAP.

The status codes consist of three digits (e.g., "404" with HTTP or "4.04" with CoAP) where the first digit expresses the class of the code. Implementations do not need to understand all status codes, but the class of the code must be understood. Codes starting with 1 are informational; the request was received and being processed. Codes starting with 2 indicate a successful request. Codes starting with 3 indicate redirection; further action is needed to complete the request. Codes stating with 4 and 5 indicate errors. The codes starting with 4 mean client error (e.g., bad syntax in the request) whereas codes starting with 5 mean server error; there was no apparent problem with the request, but server was not able to fulfill the request.

Responses may be stored in a cache to satisfy future, equivalent requests. HTTP and CoAP use two different patterns to decide what responses are cacheable. In HTTP, the cacheability of a response depends on the request method (e.g., responses returned in reply to a GET request are cacheable). In CoAP, the cacheability of a response depends on the response code (e.g., responses with code 2.04 are cacheable). This difference also leads to slightly different semantics for the codes starting with 2; for example, CoAP does not have a 2.00 response code whereas 200 ("OK") is commonly used with HTTP.

4. REST Constraints

The REST architectural style defines a set of constraints for the system design. When all constraints are applied correctly, REST enables architectural properties of key interest [REST]:

o Performance

o Scalability

o Reliability

o Simplicity

o Modifiability

o Visibility

o Portability

The following sub-sections briefly summarize the REST constraints and explain how they enable the listed properties.

4.1. Client-Server

As explained in the Architecture section, RESTful system components have clear roles in every interaction. Clients have the initiative to issue requests, intermediaries can only forward requests, and servers respond requests, while origin servers are the ultimate recipient of requests that intent to modify resource state.

This improves simplicity and visibility, as it is clear which component started an interaction. Furthermore, it improves modifiability through a clear separation of concerns.

4.2. Stateless

The Stateless constraint requires messages to be self-contained. They must contain all the information to process it, independent from previous messages. This allows to strictly separate the client state from the resource state.

This improves scalability and reliability, since servers or worker threads can be replicated. It also improves visibility because message traces contain all the information to understand the logged interactions.

Furthermore, the Stateless constraint enables caching.

4.3. Cache

This constraint requires responses to have implicit or explicit cache-control metadata. This enables clients and intermediary to store responses and re-use them to locally answer future requests. The cache-control metadata is necessary to decide whether the information in the cached response is still fresh or stale and needs to be discarded.

Cache improves performance, as less data needs to be transferred and response times can be reduced significantly. Less transfers also improves scalability, as origin servers can be protected from too many requests. Local caches furthermore improve reliability, since requests can be answered even if the origin server is temporarily not available.

4.4. Uniform Interface

All RESTful APIs use the same, uniform interface independent of the application. This simple interaction model is enabled by exchanging representations and modifying state locally, which simplifies the interface between clients and servers to a small set of methods to retrieve, update, and delete state - which applies to all applications.

In contrast, in a service-oriented RPC approach, all required ways to modify state need to be modeled explicitly in the interface resulting in a large set of methods - which differs from application to application. Moreover, it is also likely that different parties come up with different ways how to modify state, including the naming of the procedures, while the state within an application is a bit easier to agree on.

A REST interface is fully defined by:

- o URIs to identify resources
- o representation formats to represent (and retrieve and manipulate) resource state
- o self-descriptive messages with a standard set of methods (e.g., GET, POST, PUT, DELETE with their guaranteed properties)
- o hypermedia controls within representations

The concept of hypermedia controls is also known as HATEOAS: Hypermedia As The Engine Of Application State. The origin server embeds controls for the interface into its representations and thereby informs the client about possible next requests. The mostly used control for RESTful systems is Web Linking [RFC5590]. Hypermedia forms are more powerful controls that describe how to construct more complex requests, including representations to modify resource state.

While this is the most complex constraints (in particular the hypermedia controls), it improves many different key properties. It improves simplicity, as uniform interfaces are easier to understand. The self-descriptive messages improve visibility. The limitation to a known set of representation formats fosters portability. Most of all, however, this constraint is the key to modifiability, as hypermedia-driven, uniform interfaces allow clients and servers to evolve independently, and hence enable a system to evolve.

4.5. Layered System

This constraint enforces that a client cannot see beyond the server with which it is interacting.

A layered system is easier to modify, as topology changes become transparent. Furthermore, this helps scalability, as intermediaries such as load balancers can be introduced without changing the client side. The clean separation of concerns helps with simplicity.

4.6. Code-on-Demand

This principle enables origin servers to ship code to clients.

Code-on-Demand improves modifiability, since new features can be deployed during runtime (e.g., support for a new representation format). It also improves performance, as the server can provide code for local pre-processing before transferring the data.

5. Hypermedia-driven Applications

Hypermedia-driven applications take advantage of hypermedia controls, i.e., links and forms, embedded in the resource representations. A hypermedia client is a client that is capable of processing these hypermedia controls. Hypermedia links can be used to give additional information about a resource representation (e.g., the source URI of the representation) or pointing to other resources. The forms can be used to describe the structure of the data that can be sent (e.g., with a POST or PUT method) to a server, or how a data retrieval (e.g., GET) request for a resource should be formed. In a hypermedia-driven application the client interacts with the server using only the hypermedia controls, instead of selecting methods and/or constructing URIs based on out-of-band information, such as API documentation.

5.1. Motivation

The advantage of this approach is increased evolvability and extensibility. This is important in scenarios where servers exhibit a range of feature variations, where it's expensive to keep evolving client knowledge and server knowledge in sync all the time, or where there are many different client and server implementations. Hypermedia controls serve as indicators in capability negotiation. In particular, they describe available resources and possible operations on these resources using links and forms, respectively.

There are multiple reasons why a server might introduce new links or forms:

- o The server implements a newer version of the application. Older clients ignore the new links and forms, while newer clients are able to take advantage of the new features by following the new links and submitting the new forms.

- o The server offers links and forms depending on the current state. The server can tell the client which operations are currently valid and thus help the client navigate the application state machine. The client does not have to have knowledge which operations are allowed in the current state or make a request just to find out that the operation is not valid.

- o The server offers links and forms depending on the client's access control rights. If the client is unauthorized to perform a certain operation, then the server can simply omit the links and forms for that operation.

5.2. Knowledge

A client needs to have knowledge of a couple of things for successful interaction with a server. This includes what resources are available, what representations of resource states are available, what each representation describes, how to retrieve a representation, what state changing operations on a resource are possible, how to perform these operations, and so on.

Some part of this knowledge, such as how to retrieve the representation of a resource state, is typically hard-coded in the client software. For other parts, a choice can often be made between hard-coding the knowledge or acquiring it on-demand. The key to success in either case is the use in-band information for identifying the knowledge that is required. This enables the client to verify that is has all required knowledge and to acquire missing knowledge on-demand.

A hypermedia-driven application typically uses the following identifiers:

- o URI schemes that identify communication protocols,
- o Internet Media Types that identify representation formats,
- o link relation types or resource types that identify link semantics,
- o form relation types that identify form semantics,
- o variable names that identify the semantics of variables in templated links, and
- o form field names that identify the semantics of form fields in forms.

The knowledge about these identifiers as well as matching implementations have to be shared a priori in a RESTful system.

5.3. Interaction

A client begins interacting with an application through a GET request on an entry point URI. The entry point URI is the only URI a client is expected to know before interacting with an application. From there, the client is expected to make all requests by following links and submitting forms that are provided in previous responses. The entry point URI can be obtained, for example, by manual configuration or some discovery process (e.g., DNS-SD [RFC6763] or Resource Directory [I-D.ietf-core-resource-directory]). For Constrained RESTful environments "/.well-known/core" relative URI is defined as a default entry point for requesting the links hosted by servers with known or discovered addresses [RFC6690].

5.4. Hypermedia-driven Design Guidance

Assuming self-describing representation formats (i.e., human-readable with carefully chosen terms or processible by a formatting tool) and a client supporting the URI scheme used, a good rule of thumb for a good hypermedia-driven design is the following: A developer should only need an entry point URI to drive the application. All further information how to navigate through the application (links) and how to construct more complex requests (forms) are published by the server(s). There must be no need for additional, out-of-band information (e.g., API specification).

For machines, a well-chosen set of information needs to be shared a priori to agree on machine-understandable semantics. Agreeing on the exact semantics of terms for relation types and data elements will of course also help the developer.

6. Design Patterns

Certain kinds of design problems are often recurring in variety of domains, and often re-usable design patterns can be applied to them. Also some interactions with a RESTful IoT system are straightforward to design; a classic example of reading a temperature from a thermometer device is almost always implemented as a GET request to a resource that represents the current value of the thermometer. However, certain interactions, for example data conversions or event handling, do not have as straightforward and well established ways to represent the logic with resources and REST methods.

The following sections describe how common design problems such as different interactions can be modeled with REST and what are the benefits of different approaches.

6.1. Collections

A common pattern in RESTful systems across different domains is the collection. A collection can be used to combine multiple resources together by providing resources that consist of set of (often partial) representations of resources, called items, and links to resources. The collection resource also defines hypermedia controls for managing and searching the items in the collection.

Examples of the collection pattern in RESTful IoT systems are the CoRE Resource Directory [I-D.ietf-core-resource-directory], CoAP pub/sub broker [I-D.ietf-core-coap-pubsub], and resource discovery via ".well-known/core". Collection+JSON [CollectionJSON] is an example of a generic collection Media Type.

6.2. Calling a Procedure

To modify resource state, clients usually use GET to retrieve a representation from the server, modify that locally, and transfer the resulting state back to the server with a PUT (see Section 4.4). Sometimes, however, the state can only be modified on the server side, for instance, because representations would be too large to transfer or part of the required information shall not be accessible to clients. In this case, resource state is modified by calling a procedure (or "function"). This is usually modeled with a POST request, as this method leaves the behavior semantics completely to the server. Procedure calls can be divided into two different classes based on how long they are expected to execute: "instantly" returning and long-running.

6.2.1. Instantly Returning Procedures

When the procedure can return within the expected response time of the system, the result can be directly returned in the response. The result can either be actual content or just a confirmation that the call was successful. In either case, the response does not contain a representation of the resource, but a so-called action result. Action results can still have hypermedia controls to provide the possible transitions in the application state machine.

6.2.2. Long-running Procedures

When the procedure takes longer than the expected response time of the system, or even longer than the response timeout, it is a good pattern to create a new resource to track the "task" execution. The server would respond instantly with a "Created" status (HTTP code 201 or CoAP 2.01) and indicate the location of the task resource in the corresponding header field (or CoAP option) or as a link in the action result. The created resource can be used to monitor the progress, to potentially modify queued tasks or cancel tasks, and to eventually retrieve the result.

Monitoring information would be modeled as state of the task resource, and hence be retrievable as representation. The result - when available - can be embedded in the representation or given as a link to another sub-resource. Modifying tasks can be modeled with forms that either update sub-resources via PUT or do a partial write using PATCH or POST. Canceling a task would be modeled with a form that uses DELETE to remove the task resource.

6.2.3. Conversion

A conversion service is a good example where REST resources need to behave more like a procedure call. The knowledge of converting from one representation to another is located only at the server to relieve clients from high processing or storing lots of data. There are different approaches that all depend on the particular conversion problem.

As mentioned in the previous sections, POST request are a good way to model functionality that does not necessarily affect resource state. When the input data for the conversion is small and the conversion result is deterministic, however, it can be better to use a GET request with the input data in the URI query part. The query is parameterizing the conversion resource, so that it acts like a look-up table. The benefit is that results can be cached also for HTTP (where responses to POST are not cacheable). In CoAP, cacheability depends on the response code, so that also a response to a POST request can be made cacheable through a 2.05 Content code.

When the input data is large or has a binary encoding, it is better to use POST requests with a proper Media Type for the input representation. A POST request is also more suitable, when the result is time-dependent and the latest result is expected (e.g., exchange rates).

6.2.4. Events as State

In event-centric paradigms such as pub/sub, events are usually represented by an incoming message that might even be identical for each occurrence. Since the messages are queued, the receiver is aware of each occurrence of the event and can react accordingly. For instance, in an event-centric system, ringing a door bell would result in a message being sent that represents the event that it was rung.

In resource-oriented paradigms such as REST, messages usually carry the current state of the remote resource, independent from the changes (i.e., events) that have lead to that state. In a naive yet natural design, a door bell could be modeled as a resource that can have the states unpressed and pressed. There are, however, a few issues with this approach. Polling is not an option, as it is highly unlikely to be able to observe the pressed state with any realistic polling interval. When using CoAP Observe with Confirmable notifications, the server will usually send two notifications for the event that the door bell was pressed: notification for changing from unpressed to pressed and another one for changing back to unpressed. If the time between the state changes is very short, the server might drop the first notification, as Observe only guarantees only eventual consistency (see Section 1.3 of [RFC7641]).

The solution is to pick a state model that fits better to the application. In the case of the door bell - and many other event-driven resources - the solution could be a counter that counts how often the bell was pressed. The corresponding action is taken each time the client observes a change in the received representation.

In the case of a network outage, this could lead to a ringing sound 10 minutes after the bell was rung. Also including a timestamp of the last counter increment in the state can help to suppress ringing a sound when the event has become obsolete.

6.3. Server Push

Overall, a universal mechanism for server push, that is, change-of-state notifications and stand-alone event notifications, is still an open issue that is being discussed in the Thing-to-Thing Research Group. It is connected to the state-event duality problem and custody transfer, that is, the transfer of the responsibility that a message (e.g., event) is delivered successfully.

A proficient mechanism for change-of-state notifications is currently only available for CoAP: Observing resources [RFC7641]. It offers enventual consistency, which guarantees "that if the resource does not undergo a new change in state, eventually all registered observers will have a current representation of the latest resource state". It intrinsically deals with the challenges of lossy networks, where notifications might be lost, and constrained networks, where there might not be enough bandwidth to propagate all changes.

For stand-alone event notifications, that is, where every single notification contains an identifiable event that must not be lost, observing resources is not a good fit. A better strategy is to model each event as a new resource, whose existence is notified through change-of-state notifications of an index resource (cf. Collection pattern). Large numbers of events will cause the notification to grow large, as it needs to contain a large number of Web links. Blockwise transfers [RFC7959] can help here. When the links are ordered by freshness of the events, the first block can already contain all links to new events. Then, observers do not need to retrieve the remaining blocks from the server, but only the representations of the new event resources.

An alternative pattern is to exploit the dual roles of IoT devices, in particular when using CoAP: they are usually client and server at the same time. A client observer would subscribe to events by registering a callback URI at the origin server, e.g., using a POST request and receiving the location of a temporary subscription resource as handle. The origin server would then publish events by sending POST requests containing the event to the observer. The cancellation can be modeled through deleting the subscription resource. This pattern makes the origin server responsible for delivering the event notifications. This goes beyond retransmissions of messages; the origin server is usually supposed to queue all undelivered events and to retry until successful delivery or explicit cancellation. In HTTP, this pattern is known as REST Hooks.

In HTTP, there exist a number of workarounds to enable server push, e.g., long polling and streaming [RFC6202] or server-sent events [W3C.REC-html5-20141028]. Long polling as an extension that both server and client need to be aware of. In IoT systems, long polling can introduce a considerable overhead, as the request has to be repeated for each notification. Streaming and server-sent events (in fact an evolved version of streaming) are more efficient, as only one request is sent. However, there is only one response header and subsequent notifications can only have content. There are no means for individual status and metadata, and hence no means for proficient error handling (e.g., when the resource is deleted).

7. Security Considerations

This document does not define new functionality and therefore does not introduce new security concerns. We assume that system designers apply classic Web security on top of the basic RESTful guidance given in this document. Thus, security protocols and considerations from related specifications apply to RESTful IoT design. These include:

- Transport Layer Security (TLS): [RFC5246] and [RFC6347]

- Internet X.509 Public Key Infrastructure: [RFC5280]

- o HTTP security: Section 9 of [RFC7230], Section 9 of [RFC7231], etc.

- o CoAP security: Section 11 of [RFC7252]

- o URI security: Section 7 of [RFC3986]

IoT-specific security is mainly work in progress at the time of writing. First specifications include:

- o (D)TLS Profiles for the Internet of Things: [RFC7925]

Further IoT security considerations are available in [I-D.irtf-t2trg-iot-seccons].

8. Acknowledgement

The authors would like to thank Mert Ocak, Heidi-Maria Back, Tero Kauppinen, Michael Koster, Robby Simpson, Ravi Subramaniam, Dave Thaler, Erik Wilde, and Niklas Widell for the reviews and feedback.

9. References

9.1. Normative References

[I-D.ietf-core-object-security]
Selander, G., Mattsson, J., Palombini, F., and L. Seitz, "Object Security for Constrained RESTful Environments (OSCORE)", draft-ietf-core-object-security-12 (work in progress), March 2018.

[I-D.ietf-core-resource-directory]
Shelby, Z., Koster, M., Bormann, C., Stok, P., and C. Amsuess, "CoRE Resource Directory", draft-ietf-core-resource-directory-13 (work in progress), March 2018.

[REST] Fielding, R., "Architectural Styles and the Design of Network-based Software Architectures", Ph.D. Dissertation, University of California, Irvine , 2000.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <https://www.rfc-editor.org/info/rfc3986>.

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

[RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, DOI 10.17487/RFC5280, May 2008, <https://www.rfc-editor.org/info/rfc5280>.

[RFC5590] Harrington, D. and J. Schoenwaelder, "Transport Subsystem for the Simple Network Management Protocol (SNMP)", STD 78, RFC 5590, DOI 10.17487/RFC5590, June 2009, <https://www.rfc-editor.org/info/rfc5590>.

[RFC5988] Nottingham, M., "Web Linking", RFC 5988, DOI 10.17487/RFC5988, October 2010, <https://www.rfc-editor.org/info/rfc5988>.

[RFC6202] Loreto, S., Saint-Andre, P., Salsano, S., and G. Wilkins, "Known Issues and Best Practices for the Use of Long Polling and Streaming in Bidirectional HTTP", RFC 6202, DOI 10.17487/RFC6202, April 2011, <https://www.rfc-editor.org/info/rfc6202>.

[RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, DOI 10.17487/RFC6347, January 2012, <https://www.rfc-editor.org/info/rfc6347>.

[RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, DOI 10.17487/RFC6690, August 2012, <https://www.rfc-editor.org/info/rfc6690>.

[RFC7049] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", RFC 7049, DOI 10.17487/RFC7049, October 2013, <https://www.rfc-editor.org/info/rfc7049>.

[RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <https://www.rfc-editor.org/info/rfc7230>.

[RFC7231] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", RFC 7231, DOI 10.17487/RFC7231, June 2014, <https://www.rfc-editor.org/info/rfc7231>.

[RFC7641] Hartke, K., "Observing Resources in the Constrained Application Protocol (CoAP)", RFC 7641, DOI 10.17487/RFC7641, September 2015, <https://www.rfc-editor.org/info/rfc7641>.

[RFC7959] Bormann, C. and Z. Shelby, Ed., "Block-Wise Transfers in the Constrained Application Protocol (CoAP)", RFC 7959, DOI 10.17487/RFC7959, August 2016, <https://www.rfc-editor.org/info/rfc7959>.

[W3C.REC-exi-20110310]
Schneider, J. and T. Kamiya, "Efficient XML Interchange (EXI) Format 1.0", World Wide Web Consortium Recommendation REC-exi-20110310, March 2011, <http://www.w3.org/TR/2011/REC-exi-20110310>.

[W3C.REC-html5-20141028]
Hickson, I., Berjon, R., Faulkner, S., Leithead, T., Navara, E., O'Connor, T., and S. Pfeiffer, "HTML5", World Wide Web Consortium Recommendation REC-html5-20141028, October 2014, <http://www.w3.org/TR/2014/REC-html5-20141028>.

9.2. Informative References

[CollectionJSON]
Amundsen, M., "Collection+JSON - Document Format", February 2013, <http://amundsen.com/media-types/collection/format/>.

[I-D.ietf-core-coap-pubsub]
Koster, M., Keranen, A., and J. Jimenez, "Publish-Subscribe Broker for the Constrained Application Protocol (CoAP)", draft-ietf-core-coap-pubsub-04 (work in progress), March 2018.

[I-D.ietf-core-senml]
Jennings, C., Shelby, Z., Arkko, J., Keranen, A., and C. Bormann, "Media Types for Sensor Measurement Lists (SenML)", draft-ietf-core-senml-14 (work in progress), April 2018.

[I-D.irtf-t2trg-iot-seccons]
Garcia-Morchon, O., Kumar, S., and M. Sethi, "State-of-the-Art and Challenges for the Internet of Things Security", draft-irtf-t2trg-iot-seccons-14 (work in progress), April 2018.

[IANA-CoAP-media]
"CoAP Content-Formats", n.d., <http://www.iana.org/assignments/core-parameters/core-parameters.xhtml#content-formats>.

[IANA-media-types]
"Media Types", n.d., <http://www.iana.org/assignments/media-types/media-types.xhtml>.

[RFC6763] Cheshire, S. and M. Krochmal, "DNS-Based Service Discovery", RFC 6763, DOI 10.17487/RFC6763, February 2013, <https://www.rfc-editor.org/info/rfc6763>.

[RFC6943] Thaler, D., Ed., "Issues in Identifier Comparison for Security Purposes", RFC 6943, DOI 10.17487/RFC6943, May 2013, <https://www.rfc-editor.org/info/rfc6943>.

[RFC7159] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7159, DOI 10.17487/RFC7159, March 2014, <https://www.rfc-editor.org/info/rfc7159>.

[RFC7228] Bormann, C., Ersue, M., and A. Keranen, "Terminology for Constrained-Node Networks", RFC 7228, DOI 10.17487/RFC7228, May 2014, <https://www.rfc-editor.org/info/rfc7228>.

[RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <https://www.rfc-editor.org/info/rfc7252>.

[RFC7320] Nottingham, M., "URI Design and Ownership", BCP 190, RFC 7320, DOI 10.17487/RFC7320, July 2014, <https://www.rfc-editor.org/info/rfc7320>.

[RFC7925] Tschofenig, H., Ed. and T. Fossati, "Transport Layer Security (TLS) / Datagram Transport Layer Security (DTLS) Profiles for the Internet of Things", RFC 7925, DOI 10.17487/RFC7925, July 2016, <https://www.rfc-editor.org/info/rfc7925>.

Appendix A. Future Work

- o Interface semantics: shared knowledge among system components (URI schemes, media types, relation types, well-known locations; see core-apps)

- o Unreliable (best effort) communication, robust communication in network with high packet loss, 3-way commit
- o Discuss directories, such as CoAP Resource Directory
- o More information on how to design resources; choosing what is modeled as a resource, etc.

Authors' Addresses

Ari Keranen
Ericsson
Jorvas 02420
Finland

Email: ari.keranen@ericsson.com

Matthias Kovatsch
ETH Zurich
Universitaetstrasse 6
Zurich CH-8092
Switzerland

Email: kovatsch@inf.ethz.ch

Klaus Hartke
Universitaet Bremen TZI
Postfach 330440
Bremen D-28359
Germany

Email: hartke@tzi.org

Network Working Group
Internet-Draft
Intended status: Informational
Expires: 12 November 2020

A. Keranen
Ericsson
M. Kovatsch
Siemens AG
K. Hartke
Ericsson

11 May 2020

RESTful Design for Internet of Things Systems
draft-irtf-t2trg-rest-iot-06

Abstract

This document gives guidance for designing Internet of Things (IoT) systems that follow the principles of the Representational State Transfer (REST) architectural style. This document is a product of the IRTF Thing-to-Thing Research Group (T2TRG).

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 https://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 12 November 2020.

Copyright Notice

Table of Contents

1. Introduction

The Representational State Transfer (REST) architectural style [REST] is a set of guidelines and best practices for building distributed hypermedia systems. At its core is a set of constraints, which when fulfilled enable desirable properties for distributed software systems such as scalability and modifiability. When REST principles are applied to the design of a system, the result is often called RESTful and in particular an API following these principles is called a RESTful API.

Different protocols can be used with RESTful systems, but at the time of writing the most common protocols are HTTP [RFC7230] and CoAP [RFC7252]. Since RESTful APIs are often simple and lightweight, they are a good fit for various IoT applications. The goal of this document is to give basic guidance for designing RESTful systems and APIs for IoT applications and give pointers for more information.

Design of a good RESTful IoT system has naturally many commonalities with other Web systems. Compared to other systems, the key characteristics of many RESTful IoT systems include:

* need to accommodate for constrained devices, so with IoT, REST is not only used for scaling out (large number of clients on a web server), but also for scaling down (efficient server on constrained node)

* data formats, interaction patterns, and other mechanisms that minimize, or preferably avoid, the need for human interaction

* for some classes [RFC7228] of server endpoints, significant constraints, e.g., in energy consumption, and thus implementation complexity, may apply

* endpoints are commonly both clients and servers

* preference for compact and simple data formats to facilitate efficient transfer over (often) constrained networks and lightweight processing in constrained nodes

* the usually large number of endpoints can not be updated simultaneously, yet the system needs to be able to evolve in the field without long downtimes

2. Terminology

This section explains some of the common terminology that is used in the context of RESTful design for IoT systems. For terminology of constrained nodes and networks, see [RFC7228].

Cache: A local store of response messages and the subsystem that controls storage, retrieval, and deletion of messages in it.

Client: A node that sends requests to servers and receives responses. In RESTful IoT systems it's common for nodes to have more than one role (e.g., both server and client; see Section 3.1).

Client State: The state kept by a client between requests. This typically includes the currently processed representation, the set of active requests, the history of requests, bookmarks (URIs stored for later retrieval), and application-specific state (e.g., local variables). (Note that this is called "Application State" in [REST], which has some ambiguity in modern (IoT) systems where the overall state of the distributed application (i.e., application state) is reflected in the union of all Client States and Resource States of all clients and servers involved.)

Content Negotiation: The practice of determining the "best" representation for a client when examining the current state of a resource. The most common forms of content negotiation are Proactive Content Negotiation and Reactive Content Negotiation.

Dereference: To use an access mechanism (e.g., HTTP or CoAP) to perform an action on a URI's resource.

Dereferencable URI: A URI that can be dereferenced, i.e., an action can be performed on the identified resource. Not all HTTP or CoAP URIs are dereferencable, e.g., when the target resource does not exist.

Form: A hypermedia control that enables a client to change the state of a resource or to construct a query locally.

Forward Proxy: An intermediary that is selected by a client, usually via local configuration rules, and that can be tasked to make requests on behalf of the client. This may be useful, for example, when the client lacks the capability to make the request itself or to service the response from a cache in order to reduce response time, network bandwidth, and energy consumption.

Gateway: A reverse proxy that provides an interface to a non-RESTful system such as legacy systems or alternative technologies such as Bluetooth Attribute Profile (ATT) or Generic Attribute Profile (GATT). See also "Reverse Proxy".

Hypermedia Control: A component, such as a link or a form, embedded in a representation that identifies a resource for future hypermedia interactions. If the client engages in an interaction with the identified resource, the result may be a change to resource state and/or client state.

Idempotent Method: A method where multiple identical requests with that method lead to the same visible resource state as a single such request.

Link: A hypermedia control that enables a client to navigate between resources and thereby change the client state.

Link Relation Type: An identifier that describes how the link target resource relates to the current resource (see [RFC5988]).

Media Type: A string such as "text/html" or "application/json" that is used to label representations so that it is known how the representation should be interpreted and how it is encoded.

Method: An operation associated with a resource. Common methods include GET, PUT, POST, and DELETE (see Section 3.5 for details).

Origin Server: A server that is the definitive source for representations of its resources and the ultimate recipient of any request that intends to modify its resources. In contrast, intermediaries (such as proxies caching a representation) can assume the role of a server, but are not the source for representations as these are acquired from the origin server.

Proactive Content Negotiation: A content negotiation mechanism where the server selects a representation based on the expressed preference of the client. For example, an IoT application could send a request to a sensor with preferred media type "application/senml+json".

Reactive Content Negotiation: A content negotiation mechanism where the client selects a representation from a list of available representations. The list may, for example, be included by a server in an initial response. If the user agent is not satisfied by the initial response representation, it can request one or more of the alternative representations, selected based on metadata (e.g., available media types) included in the response.

Representation: A serialization that represents the current or intended state of a resource and that can be transferred between clients and servers. REST requires representations to be self-describing, meaning that there must be metadata that allows peers to understand which representation format is used. Depending on the protocol needs and capabilities, there can be additional metadata that is transmitted along with the representation.

Representation Format: A set of rules for serializing resource state. On the Web, the most prevalent representation format is HTML. Other common formats include plain text and formats based on JSON [RFC8259], XML, or RDF. Within IoT systems, often compact formats based on JSON, CBOR [RFC7049], and EXI [W3C.REC-exi-20110310] are used.

Representational State Transfer (REST): An architectural style for Internet-scale distributed hypermedia systems.

Resource: An item of interest identified by a URI. Anything that can be named can be a resource. A resource often encapsulates a piece of state in a system. Typical resources in an IoT system can be, e.g., a sensor, the current value of a sensor, the location of a device, or the current state of an actuator.

Resource State: A model of a resource's possible states that is represented in a supported representation type, typically a media type. Resources can change state because of REST interactions with them, or they can change state for reasons outside of the REST model.

Resource Type: An identifier that annotates the application-semantics of a resource (see Section 3.1 of [RFC6690]).

Reverse Proxy: An intermediary that appears as a server towards the client but satisfies the requests by forwarding them to the actual server (possibly via one or more other intermediaries). A reverse proxy is often used to encapsulate legacy services, to improve server performance through caching, and to enable load balancing across multiple machines.

Safe Method: A method that does not result in any state change on the origin server when applied to a resource.

Server: A node that listens for requests, performs the requested operation and sends responses back to the clients.

Transfer protocols: In particular in IoT domain, protocols above transport layer that are used to transfer data objects and provide semantics for operations on the data.

Transfer layer: Re-usable part of the application layer used to transfer the application specific data items using a standard set of methods that can fulfill application-specific operations.

Uniform Resource Identifier (URI): A global identifier for resources. See Section 3.3 for more details.

3. Basics

3.1. Architecture

The components of a RESTful system are assigned one or both of two roles: client or server. Note that the terms "client" and "server" refer only to the roles that the nodes assume for a particular message exchange. The same node might act as a client in some communications and a server in others. Classic user agents (e.g., Web browsers) are always in the client role and have the initiative to issue requests. Origin servers always have the server role and govern over the resources they host. Simple IoT devices, such as sensors and actuators, are commonly acting as servers and exposing their physical world interaction capabilities (e.g., temperature measurement or door lock control capability) as resources. Typical IoT system client can be a cloud service that retrieves data from the sensors and commands the actuators based on the sensor information. Alternatively an IoT data storage system could work as a server where IoT sensor devices send data, in client role.

```
 ________                       __________
|        |                     |          |
| User  (C)-------------------(S) Origin  |
| Agent  |                     |  Server  |
|________|                     |__________|
(Browser)                      (Web Server)
```

Figure 1: Client-Server Communication

Intermediaries (such as forward proxies, reverse proxies, and gateways) implement both roles, but only forward requests to other intermediaries or origin servers. They can also translate requests to different protocols, for instance, as CoAP-HTTP cross-proxies [RFC8075].

```
        ________       __________                        _________
       |        |     |          |                      |         |
       | User  (C)---(S) Inter- (C)--------------------(S) Origin |
       | Agent  |     |  mediary |                      |  Server |
       |________|     |__________|                      |_________|
       (Browser)     (Forward Proxy)                    (Web Server)
```

Figure 2: Communication with Forward Proxy

Reverse proxies are usually imposed by the origin server. In addition to the features of a forward proxy, they can also provide an interface for non-RESTful services such as legacy systems or alternative technologies such as Bluetooth ATT/GATT. In this case, reverse proxies are usually called gateways. This property is enabled by the Layered System constraint of REST, which says that a client cannot see beyond the server it is connected to (i.e., it is left unaware of the protocol/paradigm change).

```
       ________                        __________       _________
      |        |                      |          |     |         |
      | User  (C)--------------------(S) Inter- (x)---(x) Origin |
      | Agent  |                      |  mediary |     |  Server |
      |________|                      |__________|     |_________|
      (Browser)                        (Gateway)     (Legacy System)
```

Figure 3: Communication with Reverse Proxy

Nodes in IoT systems often implement both roles. Unlike intermediaries, however, they can take the initiative as a client (e.g., to register with a directory, such as CoRE Resource Directory [I-D.ietf-core-resource-directory], or to interact with another thing) and act as origin server at the same time (e.g., to serve sensor values or provide an actuator interface).

```
      ________                                         _________
     |        |                                       |         |
     | Thing (C)-------------------------------------(S) Origin |
     |       (S)                                      |  Server |
     |________| \                                     |_________|
      (Sensor)   \   ________                     (Resource Directory)
                  \ |        |
                   (C) Thing |
                    |________|
                   (Controller)
```

Figure 4: Constrained RESTful environments

3.2. System design

When designing a RESTful system, the primary effort goes into modeling the state of the distributed application and assigning it to the different components (i.e., clients and servers). How clients can navigate through the resources and modify state to achieve their goals is defined through hypermedia controls, that is, links and forms. Hypermedia controls span a kind of a state machine where the nodes are resources and the transitions are links or forms. Clients run this state machine (i.e., the application) by retrieving representations, processing the data, and following the included hypermedia controls. In REST, remote state is changed by submitting forms. This is usually done by retrieving the current state, modifying the state on the client side, and transferring the new state to the server in the form of new representations - rather than calling a service and modifying the state on the server side.

Client state encompasses the current state of the described state machine and the possible next transitions derived from the hypermedia controls within the currently processed representation (see Section 2). Furthermore, clients can have part of the state of the distributed application in local variables.

Resource state includes the more persistent data of an application (i.e., independent of individual clients). This can be static data such as device descriptions, persistent data such as system configurations, but also dynamic data such as the current value of a sensor on a thing.

It is important to distinguish between "client state" and "resource state" and keep them separate. Following the Stateless constraint, the client state must be kept only on clients. That is, there is no establishment of shared information about past and future interactions between client and server (usually called a session). On the one hand, this makes requests a bit more verbose since every request must contain all the information necessary to process it. On the other hand, this makes servers efficient and scalable, since they do not have to keep any state about their clients. Requests can easily be distributed over multiple worker threads or server instances. For IoT systems, this constraint lowers the memory requirements for server implementations, which is particularly important for constrained servers (e.g., sensor nodes) and servers serving large amount of clients (e.g., Resource Directory).

3.3. Uniform Resource Identifiers (URIs)

An important part of RESTful API design is to model the system as a set of resources whose state can be retrieved and/or modified and where resources can be potentially also created and/or deleted.

Uniform Resource Identifiers (URIs) are used to indicate a resource for interaction, to reference a resource from another resource, to advertise or bookmark a resource, or to index a resource by search engines.

```
  foo://example.com:8042/over/there?name=ferret#nose
  \_/   \______________/\_________/ \_________/ \__/
   |           |            |            |        |
scheme     authority       path        query   fragment
```

A URI is a sequence of characters that matches the syntax defined in [RFC3986]. It consists of a hierarchical sequence of five components: scheme, authority, path, query, and fragment (from most significant to least significant). A scheme creates a namespace for resources and defines how the following components identify a resource within that namespace. The authority identifies an entity that governs part of the namespace, such as the server "www.example.org" in the "http" scheme. A host name (e.g., a fully qualified domain name) or an IP address, potentially followed by a transport layer port number, are usually used in the authority component for the "http" and "coap" schemes. The path and query contain data to identify a resource within the scope of the URI's scheme and naming authority. The fragment allows to refer to some portion of the resource, such as a Record in a SenML Pack (Section 9 of [RFC8428]). However, fragments are processed only at client side and not sent on the wire. [RFC7320] provides more details on URI design and ownership with best current practices for establishing URI structures, conventions, and formats.

For RESTful IoT applications, typical schemes include "https", "coaps", "http", and "coap". These refer to HTTP and CoAP, with and without Transport Layer Security (TLS, [RFC5246] for TLS 1.2 and [RFC8446] for TLS 1.3). (CoAP uses Datagram TLS (DTLS) [RFC6347], the variant of TLS for UDP.) These four schemes also provide means for locating the resource; using the HTTP protocol for "http" and "https", and with the CoAP protocol for "coap" and "coaps". If the scheme is different for two URIs (e.g., "coap" vs. "coaps"), it is important to note that even if the rest of the URI is identical, these are two different resources, in two distinct namespaces.

Some schemes are for URIs with main purpose as identifiers and hence are not dereferencable, e.g., the "urn" scheme can be used to construct unique names in registered namespaces. In particular the "urn:dev" [I-D.ietf-core-dev-urn] details multiple ways for generating and representing endpoint identifiers of IoT devices.

The query parameters can be used to parametrize the resource. For example, a GET request may use query parameters to request the server to send only certain kind data of the resource (i.e., filtering the response). Query parameters in PUT and POST requests do not have such established semantics and are not commonly used. Whether the order of the query parameters matters in URIs is unspecified and they can be re-ordered e.g., by proxies. Therefore applications should not rely on their order; see Section 3.3 of [RFC6943] for more details.

Due to the relatively complex processing rules and text representation format, URI handling can be difficult to implement correctly in constrained devices. Constrained Resource Identifiers [I-D.ietf-core-href] provide a CBOR-based format of URIs that is better suited also for resource constrained IoT devices.

3.4. Representations

Clients can retrieve the resource state from an origin server or manipulate resource state on the origin server by transferring resource representations. Resource representations have a content-type (media-type, optionally with parameters) that tells how the representation should be interpreted by identifying the representation format used.

Typical media-types for IoT systems include:

* "text/plain" for simple UTF-8 text

* "application/octet-stream" for arbitrary binary data

* "application/json" for the JSON format [RFC8259]

* "application/cbor" for CBOR [RFC7049]

* "application/exi" for EXI [W3C.REC-exi-20110310]

* "application/link-format" for CoRE Link Format [RFC6690]

* "application/senml+json" and "application/senml+cbor" for Sensor Measurement Lists (SenML) data [RFC8428]

A full list of registered Internet Media Types is available at the IANA registry [IANA-media-types] and numerical identifiers for media-types, parameters, and content-codings registered for use with CoAP are listed at CoAP Content-Formats IANA registry [IANA-CoAP-media].

The terms "media-type", "content-type", and "content-format" (short identifier of content-type and content-coding, abbreviated for historical reasons "ct") are often used when referring to representation formats used with CoAP. The differences between these terms are discussed in more detail in [I-D.bormann-core-media-content-type-format].

3.5. HTTP/CoAP Methods

Section 4.3 of [RFC7231] defines the set of methods in HTTP; Section 5.8 of [RFC7252] defines the set of methods in CoAP. As part of the Uniform Interface constraint, each method can have certain properties that give guarantees to clients.

Safe methods do not cause any state change on the origin server when applied to a resource. For example, the GET method only returns a representation of the resource state but does not change the resource. Thus, it is always safe for a client to retrieve a representation without affecting server-side state.

Idempotent methods can be applied multiple times to the same resource while causing the same visible resource state as a single such request. For example, the PUT method replaces the state of a resource with a new state; replacing the state multiple times with the same new state still results in the same state for the resource. However, the response from the server can be different when the same idempotent method is used multiple times. For example when DELETE is used twice on an existing resource, the first request would remove the association and return success acknowledgement whereas the second request would likely result in error response due to non-existing resource.

The following lists the most relevant methods and gives a short explanation of their semantics.

3.5.1. GET

The GET method requests a current representation for the target resource, while the origin server must ensure that there are no side-effects on the resource state. Only the origin server needs to know how each of its resource identifiers corresponds to an implementation and how each implementation manages to select and send a current representation of the target resource in a response to GET.

A payload within a GET request message has no defined semantics.

The GET method is safe and idempotent.

3.5.2. POST

The POST method requests that the target resource process the representation enclosed in the request according to the resource's own specific semantics.

If one or more resources has been created on the origin server as a result of successfully processing a POST request, the origin server sends a 201 (Created) response containing a Location header field (with HTTP) or Location-Path and/or Location-Query Options (with CoAP) that provide an identifier for the resource created. The server also includes a representation that describes the status of the request while referring to the new resource(s).

The POST method is not safe nor idempotent.

3.5.3. PUT

The PUT method requests that the state of the target resource be created or replaced with the state defined by the representation enclosed in the request message payload. A successful PUT of a given representation would suggest that a subsequent GET on that same target resource will result in an equivalent representation being sent.

The fundamental difference between the POST and PUT methods is highlighted by the different intent for the enclosed representation. The target resource in a POST request is intended to handle the enclosed representation according to the resource's own semantics, whereas the enclosed representation in a PUT request is defined as replacing the state of the target resource. Hence, the intent of PUT is idempotent and visible to intermediaries, even though the exact effect is only known by the origin server.

The PUT method is not safe, but is idempotent.

3.5.4. DELETE

The DELETE method requests that the origin server remove the association between the target resource and its current functionality.

If the target resource has one or more current representations, they might or might not be destroyed by the origin server, and the associated storage might or might not be reclaimed, depending entirely on the nature of the resource and its implementation by the origin server.

The DELETE method is not safe, but is idempotent.

3.5.5. FETCH

The CoAP-specific FETCH method [RFC8132] requests a representation of a resource parameterized by a representation enclosed in the request.

The fundamental difference between the GET and FETCH methods is that the request parameters are included as the payload of a FETCH request, while in a GET request they're typically part of the query string of the request URI.

The FETCH method is safe and idempotent.

3.5.6. PATCH

The PATCH method [RFC5789] [RFC8132] requests that a set of changes described in the request entity be applied to the target resource.

The PATCH method is not safe nor idempotent.

The CoAP-specific iPATCH method is a variant of the PATCH method that is not safe, but is idempotent.

3.6. HTTP/CoAP Status/Response Codes

Section 6 of [RFC7231] defines a set of Status Codes in HTTP that are used by application to indicate whether a request was understood and satisfied, and how to interpret the answer. Similarly, Section 5.9 of [RFC7252] defines the set of Response Codes in CoAP.

The status codes consist of three digits (e.g., "404" with HTTP or "4.04" with CoAP) where the first digit expresses the class of the code. Implementations do not need to understand all status codes, but the class of the code must be understood. Codes starting with 1 are informational; the request was received and being processed.

Codes starting with 2 indicate a successful request. Codes starting with 3 indicate redirection; further action is needed to complete the request. Codes stating with 4 and 5 indicate errors. The codes starting with 4 mean client error (e.g., bad syntax in the request) whereas codes starting with 5 mean server error; there was no apparent problem with the request, but server was not able to fulfill the request.

Responses may be stored in a cache to satisfy future, equivalent requests. HTTP and CoAP use two different patterns to decide what responses are cacheable. In HTTP, the cacheability of a response depends on the request method (e.g., responses returned in reply to a GET request are cacheable). In CoAP, the cacheability of a response depends on the response code (e.g., responses with code 2.04 are cacheable). This difference also leads to slightly different semantics for the codes starting with 2; for example, CoAP does not have a 2.00 response code whereas 200 ("OK") is commonly used with HTTP.

4. REST Constraints

The REST architectural style defines a set of constraints for the system design. When all constraints are applied correctly, REST enables architectural properties of key interest [REST]:

* Performance
* Scalability
* Reliability
* Simplicity
* Modifiability
* Visibility
* Portability

The following sub-sections briefly summarize the REST constraints and explain how they enable the listed properties.

4.1. Client-Server

As explained in the Architecture section, RESTful system components have clear roles in every interaction. Clients have the initiative to issue requests, intermediaries can only forward requests, and servers respond requests, while origin servers are the ultimate recipient of requests that intent to modify resource state.

This improves simplicity and visibility (also for digital forensics), as it is clear which component started an interaction. Furthermore, it improves modifiability through a clear separation of concerns.

In IoT systems, endpoints often assume both roles of client and (origin) server simultaneously. When an IoT device has initiative (because there is a user, e.g., pressing a button, or installed rules/policies), it acts as a client. When a device offers a service, it is in server role.

4.2. Stateless

The Stateless constraint requires messages to be self-contained. They must contain all the information to process it, independent from previous messages. This allows to strictly separate the client state from the resource state.

This improves scalability and reliability, since servers or worker threads can be replicated. It also improves visibility because message traces contain all the information to understand the logged interactions. Furthermore, the Stateless constraint enables caching.

For IoT, the scaling properties of REST become particularly important. Note that being self-contained does not necessarily mean that all information has to be inlined. Constrained IoT devices may choose to externalize metadata and hypermedia controls using Web linking, so that only the dynamic content needs to be sent and the static content such as schemas or controls can be cached.

4.3. Cache

This constraint requires responses to have implicit or explicit cache-control metadata. This enables clients and intermediary to store responses and re-use them to locally answer future requests. The cache-control metadata is necessary to decide whether the information in the cached response is still fresh or stale and needs to be discarded.

Cache improves performance, as less data needs to be transferred and response times can be reduced significantly. Less transfers also improves scalability, as origin servers can be protected from too many requests. Local caches furthermore improve reliability, since requests can be answered even if the origin server is temporarily not available.

Caching usually only makes sense when the data is used by multiple participants. In IoT systems, however, it might make sense to cache also individual data to protect constrained devices and networks from frequent requests of data that does not change often. Security often hinders the ability to cache responses. For IoT systems, object security [RFC8613] may be preferable over transport layer security, as it enables intermediaries to cache responses while preserving security.

4.4. Uniform Interface

All RESTful APIs use the same, uniform interface independent of the application. This simple interaction model is enabled by exchanging representations and modifying state locally, which simplifies the interface between clients and servers to a small set of methods to retrieve, update, and delete state - which applies to all applications.

In contrast, in a service-oriented RPC approach, all required ways to modify state need to be modeled explicitly in the interface resulting in a large set of methods - which differs from application to application. Moreover, it is also likely that different parties come up with different ways how to modify state, including the naming of the procedures, while the state within an application is a bit easier to agree on.

A REST interface is fully defined by:

* URIs to identify resources

* representation formats to represent and manipulate resource state

* self-descriptive messages with a standard set of methods (e.g., GET, POST, PUT, DELETE with their guaranteed properties)

* hypermedia controls within representations

The concept of hypermedia controls is also known as HATEOAS: Hypermedia As The Engine Of Application State. The origin server embeds controls for the interface into its representations and thereby informs the client about possible next requests. The most used control for RESTful systems today is Web Linking [RFC5988]. Hypermedia forms are more powerful controls that describe how to construct more complex requests, including representations to modify resource state.

While this is the most complex constraints (in particular the hypermedia controls), it improves many different key properties. It improves simplicity, as uniform interfaces are easier to understand. The self-descriptive messages improve visibility. The limitation to a known set of representation formats fosters portability. Most of all, however, this constraint is the key to modifiability, as hypermedia-driven, uniform interfaces allow clients and servers to evolve independently, and hence enable a system to evolve.

For a large number of IoT applications, the hypermedia controls are mainly used for the discovery of resources, as they often serve sensor data. Such resources are "dead ends", as they usually do not link any further and only have one form of interaction: fetching the sensor value. For IoT, the critical parts of the Uniform Interface constraint are the descriptions of messages and representation formats used. Simply using, for instance, "application/json" does not help machine clients to understand the semantics of the representation. Yet defining very precise media types limits the re-usability and interoperability. Representation formats such as SenML [RFC8428] try to find a good trade-off between precision and re-usability. Another approach is to combine a generic format such as JSON with syntactic as well as semantic annotations (see [I-D.handrews-json-schema-validation] and [W3C-TD], resp.).

4.5. Layered System

This constraint enforces that a client cannot see beyond the server with which it is interacting.

A layered system is easier to modify, as topology changes become transparent. Furthermore, this helps scalability, as intermediaries such as load balancers can be introduced without changing the client side. The clean separation of concerns helps with simplicity.

IoT systems greatly benefit from this constraint, as it allows to effectively shield constrained devices behind intermediaries and is also the basis for gateways, which are used to integrate other (IoT) ecosystems.

4.6. Code-on-Demand

This principle enables origin servers to ship code to clients.

Code-on-Demand improves modifiability, since new features can be deployed during runtime (e.g., support for a new representation format). It also improves performance, as the server can provide code for local pre-processing before transferring the data.

As of today, code-on-demand has not been explored much in IoT systems. Aspects to consider are that either one or both nodes are constrained and might not have the resources to host or dynamically fetch and execute such code. Moreover, the origin server often has no understanding of the actual application a mashup client realizes. Still, code-on-demand can be useful for small polyfills, e.g., to decode payloads, and potentially other features in the future.

5. Hypermedia-driven Applications

Hypermedia-driven applications take advantage of hypermedia controls, i.e., links and forms, embedded in the resource representations. A hypermedia client is a client that is capable of processing these hypermedia controls. Hypermedia links can be used to give additional information about a resource representation (e.g., the source URI of the representation) or pointing to other resources. The forms can be used to describe the structure of the data that can be sent (e.g., with a POST or PUT method) to a server, or how a data retrieval (e.g., GET) request for a resource should be formed. In a hypermedia-driven application the client interacts with the server using only the hypermedia controls, instead of selecting methods and/or constructing URIs based on out-of-band information, such as API documentation. The Constrained RESTful Application Language (CoRAL) [I-D.ietf-core-coral] provides a hypermedia-format that is suitable for constrained IoT environments.

5.1. Motivation

The advantage of this approach is increased evolvability and extensibility. This is important in scenarios where servers exhibit a range of feature variations, where it's expensive to keep evolving client knowledge and server knowledge in sync all the time, or where there are many different client and server implementations. Hypermedia controls serve as indicators in capability negotiation. In particular, they describe available resources and possible operations on these resources using links and forms, respectively.

There are multiple reasons why a server might introduce new links or forms:

* The server implements a newer version of the application. Older clients ignore the new links and forms, while newer clients are able to take advantage of the new features by following the new links and submitting the new forms.

* The server offers links and forms depending on the current state. The server can tell the client which operations are currently valid and thus help the client navigate the application state machine. The client does not have to have knowledge which operations are allowed in the current state or make a request just to find out that the operation is not valid.

* The server offers links and forms depending on the client's access control rights. If the client is unauthorized to perform a certain operation, then the server can simply omit the links and forms for that operation.

5.2. Knowledge

A client needs to have knowledge of a couple of things for successful interaction with a server. This includes what resources are available, what representations of resource states are available, what each representation describes, how to retrieve a representation, what state changing operations on a resource are possible, how to perform these operations, and so on.

Some part of this knowledge, such as how to retrieve the representation of a resource state, is typically hard-coded in the client software. For other parts, a choice can often be made between hard-coding the knowledge or acquiring it on-demand. The key to success in either case is the use of in-band information for identifying the knowledge that is required. This enables the client to verify that it has all the required knowledge or to acquire missing knowledge on-demand.

A hypermedia-driven application typically uses the following identifiers:

* URI schemes that identify communication protocols,

* Internet Media Types that identify representation formats,

* link relation types or resource types that identify link semantics,

* form relation types that identify form semantics,

* variable names that identify the semantics of variables in templated links, and

* form field names that identify the semantics of form fields in forms.

The knowledge about these identifiers as well as matching implementations have to be shared a priori in a RESTful system.

5.3. Interaction

A client begins interacting with an application through a GET request on an entry point URI. The entry point URI is the only URI a client is expected to know before interacting with an application. From there, the client is expected to make all requests by following links and submitting forms that are provided in previous responses. The entry point URI can be obtained, for example, by manual configuration or some discovery process (e.g., DNS-SD [RFC6763] or Resource Directory [I-D.ietf-core-resource-directory]). For Constrained RESTful environments "/.well-known/core" relative URI is defined as a default entry point for requesting the links hosted by servers with known or discovered addresses [RFC6690].

5.4. Hypermedia-driven Design Guidance

Assuming self-describing representation formats (i.e., human-readable with carefully chosen terms or processible by a formatting tool) and a client supporting the URI scheme used, a good rule of thumb for a good hypermedia-driven design is the following: A developer should only need an entry point URI to drive the application. All further information how to navigate through the application (links) and how to construct more complex requests (forms) are published by the server(s). There must be no need for additional, out-of-band information (e.g., API specification).

For machines, a well-chosen set of information needs to be shared a priori to agree on machine-understandable semantics. Agreeing on the exact semantics of terms for relation types and data elements will of course also help the developer. [I-D.hartke-core-apps] proposes a convention for specifying the set of information in a structured way.

6. Design Patterns

Certain kinds of design problems are often recurring in variety of domains, and often re-usable design patterns can be applied to them. Also some interactions with a RESTful IoT system are straightforward to design; a classic example of reading a temperature from a thermometer device is almost always implemented as a GET request to a resource that represents the current value of the thermometer. However, certain interactions, for example data conversions or event handling, do not have as straightforward and well established ways to represent the logic with resources and REST methods.

The following sections describe how common design problems such as different interactions can be modeled with REST and what are the benefits of different approaches.

6.1. Collections

A common pattern in RESTful systems across different domains is the collection. A collection can be used to combine multiple resources together by providing resources that consist of set of (often partial) representations of resources, called items, and links to resources. The collection resource also defines hypermedia controls for managing and searching the items in the collection.

Examples of the collection pattern in RESTful IoT systems are the CoRE Resource Directory [I-D.ietf-core-resource-directory], CoAP pub/sub broker [I-D.ietf-core-coap-pubsub], and resource discovery via ".well-known/core". Collection+JSON [CollectionJSON] is an example of a generic collection Media Type.

6.2. Calling a Procedure

To modify resource state, clients usually use GET to retrieve a representation from the server, modify that locally, and transfer the resulting state back to the server with a PUT (see Section 4.4). Sometimes, however, the state can only be modified on the server side, for instance, because representations would be too large to transfer or part of the required information shall not be accessible to clients. In this case, resource state is modified by calling a procedure (or "function"). This is usually modeled with a POST request, as this method leaves the behavior semantics completely to the server. Procedure calls can be divided into two different classes based on how long they are expected to execute: "instantly" returning and long-running.

6.2.1. Instantly Returning Procedures

When the procedure can return within the expected response time of the system, the result can be directly returned in the response. The result can either be actual content or just a confirmation that the call was successful. In either case, the response does not contain a representation of the resource, but a so-called action result. Action results can still have hypermedia controls to provide the possible transitions in the application state machine.

6.2.2. Long-running Procedures

When the procedure takes longer than the expected response time of the system, or even longer than the response timeout, it is a good pattern to create a new resource to track the "task" execution. The server would respond instantly with a "Created" status (HTTP code 201 or CoAP 2.01) and indicate the location of the task resource in the corresponding header field (or CoAP option) or as a link in the action result. The created resource can be used to monitor the progress, to potentially modify queued tasks or cancel tasks, and to eventually retrieve the result.

Monitoring information would be modeled as state of the task resource, and hence be retrievable as representation. The result - when available - can be embedded in the representation or given as a link to another sub-resource. Modifying tasks can be modeled with forms that either update sub-resources via PUT or do a partial write using PATCH or POST. Canceling a task would be modeled with a form that uses DELETE to remove the task resource.

6.2.3. Conversion

A conversion service is a good example where REST resources need to behave more like a procedure call. The knowledge of converting from one representation to another is located only at the server to relieve clients from high processing or storing lots of data. There are different approaches that all depend on the particular conversion problem.

As mentioned in the previous sections, POST request are a good way to model functionality that does not necessarily affect resource state. When the input data for the conversion is small and the conversion result is deterministic, however, it can be better to use a GET request with the input data in the URI query part. The query is parameterizing the conversion resource, so that it acts like a look-up table. The benefit is that results can be cached also for HTTP (where responses to POST are not cacheable). In CoAP, cacheability depends on the response code, so that also a response to a POST request can be made cacheable through a 2.05 Content code.

When the input data is large or has a binary encoding, it is better to use POST requests with a proper Media Type for the input representation. A POST request is also more suitable, when the result is time-dependent and the latest result is expected (e.g., exchange rates).

6.2.4. Events as State

In event-centric paradigms such as pub/sub, events are usually represented by an incoming message that might even be identical for each occurrence. Since the messages are queued, the receiver is aware of each occurrence of the event and can react accordingly. For instance, in an event-centric system, ringing a door bell would result in a message being sent that represents the event that it was rung.

In resource-oriented paradigms such as REST, messages usually carry the current state of the remote resource, independent from the changes (i.e., events) that have lead to that state. In a naive yet natural design, a door bell could be modeled as a resource that can have the states unpressed and pressed. There are, however, a few issues with this approach. Polling (i.e., periodically retrieving) the door bell resource state is not a good option, as the client is highly unlikely to be able to observe all the changes in the pressed state with any realistic polling interval. When using CoAP Observe with Confirmable notifications, the server will usually send two notifications for the event that the door bell was pressed: notification for changing from unpressed to pressed and another one for changing back to unpressed. If the time between the state changes is very short, the server might drop the first notification, as Observe only guarantees only eventual consistency (see Section 1.3 of [RFC7641]).

The solution is to pick a state model that fits better to the application. In the case of the door bell - and many other event-driven resources - the solution could be a counter that counts how often the bell was pressed. The corresponding action is taken each time the client observes a change in the received representation. In the case of a network outage, this could lead to a ringing sound long after the bell was rung. Also including a timestamp of the last counter increment in the state can help to suppress ringing a sound when the event has become obsolete. Another solution would be to change the client/server roles of the door bell button and the ringer, as described in Section 6.3.

6.3. Server Push

Overall, a universal mechanism for server push, that is, change-of-state notifications and stand-alone event notifications, is still an open issue that is being discussed in the Thing-to-Thing Research Group. It is connected to the state-event duality problem and custody transfer, that is, the transfer of the responsibility that a message (e.g., event) is delivered successfully.

A proficient mechanism for change-of-state notifications is currently only available for CoAP: Observing resources [RFC7641]. The CoAP Observe mechanism offers eventual consistency, which guarantees "that if the resource does not undergo a new change in state, eventually all registered observers will have a current representation of the latest resource state". It intrinsically deals with the challenges of lossy networks, where notifications might be lost, and constrained networks, where there might not be enough bandwidth to propagate all changes.

For stand-alone event notifications, that is, where every single notification contains an identifiable event that must not be lost, observing resources is not a good fit. A better strategy is to model each event as a new resource, whose existence is notified through change-of-state notifications of an index resource (cf. Collection pattern). Large numbers of events will cause the notification to grow large, as it needs to contain a large number of Web links. Block-wise transfers [RFC7959] can help here. When the links are ordered by freshness of the events, the first block can already contain all links to new events. Then, observers do not need to retrieve the remaining blocks from the server, but only the representations of the new event resources.

An alternative pattern is to exploit the dual roles of IoT devices, in particular when using CoAP: they are usually client and server at the same time. An endpoint interested in observing the events would subscribe to them by registering a callback URI at the origin server, e.g., using a POST request with the URI or a hypermedia document in the payload, and receiving the location of a temporary "subscription resource" as handle in the response. The origin server would then publish events by sending requests containing the event data to the observer’s callback URI; here POST can be used to add events to a collection located at the callback URI or PUT can be used when the event data is a new state that shall replace the outdated state at the callback URI. The cancellation can be modeled through deleting the subscription resource. This pattern makes the origin server responsible for delivering the event notifications. This goes beyond retransmissions of messages; the origin server is usually supposed to queue all undelivered events and to retry until successful delivery or explicit cancellation. In HTTP, this pattern is known as REST Hooks.

Methods for configuring server push and notification conditions with CoAP are provided by the CoRE Dynamic Resource Linking specification [I-D.ietf-core-dynlink].

In HTTP, there exist a number of workarounds to enable server push, e.g., long polling and streaming [RFC6202] or server-sent events [W3C.REC-html5-20141028]. In IoT systems, long polling can introduce a considerable overhead, as the request has to be repeated for each notification. Streaming and server-sent events (the latter is actually an evolution of the former) are more efficient, as only one request is sent. However, there is only one response header and subsequent notifications can only have content. Individual status and metadata needs to be included in the content message. This reduces HTTP again to a pure transport, as its status signaling and metadata capabilities cannot be used.

7. Security Considerations

This document does not define new functionality and therefore does not introduce new security concerns. We assume that system designers apply classic Web security on top of the basic RESTful guidance given in this document. Thus, security protocols and considerations from related specifications apply to RESTful IoT design. These include:

* Transport Layer Security (TLS): [RFC8446], [RFC5246], and [RFC6347]

* Internet X.509 Public Key Infrastructure: [RFC5280]

* HTTP security: Section 9 of [RFC7230], Section 9 of [RFC7231], etc.

* CoAP security: Section 11 of [RFC7252]

* URI security: Section 7 of [RFC3986]

IoT-specific security is active area of standardization at the time of writing. First finalized specifications include:

* (D)TLS Profiles for the Internet of Things: [RFC7925]

* CBOR Object Signing and Encryption (COSE) [RFC8152]

* CBOR Web Token [RFC8392]

* Proof-of-Possession Key Semantics for CBOR Web Tokens (CWTs) [RFC8747]

* Object Security for Constrained RESTful Environments (OSCORE) [RFC8613]

* Authentication and Authorization for Constrained Environments (ACE) using the OAuth 2.0 Framework [I-D.ietf-ace-oauth-authz]

* ACE profiles for DTLS [I-D.ietf-ace-dtls-authorize] and OSCORE [I-D.ietf-ace-oscore-profile]

Further IoT security considerations are available in [RFC8576].

8. Acknowledgement

The authors would like to thank Mike Amundsen, Heidi-Maria Back, Carsten Bormann, Tero Kauppinen, Michael Koster, Mert Ocak, Robby Simpson, Ravi Subramaniam, Dave Thaler, Niklas Widell, and Erik Wilde for the reviews and feedback.

9. References

9.1. Normative References

[I-D.ietf-core-coral]
Hartke, K., "The Constrained RESTful Application Language (CoRAL)", Work in Progress, Internet-Draft, draft-ietf-core-coral-03, 9 March 2020, <http://www.ietf.org/internet-drafts/draft-ietf-core-coral-03.txt>.

[I-D.ietf-core-dev-urn]
Arkko, J., Jennings, C., and Z. Shelby, "Uniform Resource Names for Device Identifiers", Work in Progress, Internet-Draft, draft-ietf-core-dev-urn-04, 13 December 2019, <http://www.ietf.org/internet-drafts/draft-ietf-core-dev-urn-04.txt>.

[I-D.ietf-core-dynlink]
Shelby, Z., Koster, M., Groves, C., Zhu, J., and B. Silverajan, "Dynamic Resource Linking for Constrained RESTful Environments", Work in Progress, Internet-Draft, draft-ietf-core-dynlink-10, 22 July 2019, <http://www.ietf.org/internet-drafts/draft-ietf-core-dynlink-10.txt>.

[I-D.ietf-core-href]
Hartke, K., "Constrained Resource Identifiers", Work in Progress, Internet-Draft, draft-ietf-core-href-03, 9 March 2020, <http://www.ietf.org/internet-drafts/draft-ietf-core-href-03.txt>.

[I-D.ietf-core-resource-directory]
Shelby, Z., Koster, M., Bormann, C., Stok, P., and C. Amsuess, "CoRE Resource Directory", Work in Progress, Internet-Draft, draft-ietf-core-resource-directory-24, 9 March 2020, <http://www.ietf.org/internet-drafts/draft-ietf-core-resource-directory-24.txt>.

[REST] Fielding, R., "Architectural Styles and the Design of Network-based Software Architectures", Ph.D. Dissertation, University of California, Irvine , 2000.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <https://www.rfc-editor.org/info/rfc3986>.

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

[RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, DOI 10.17487/RFC5280, May 2008, <https://www.rfc-editor.org/info/rfc5280>.

[RFC5988] Nottingham, M., "Web Linking", RFC 5988, DOI 10.17487/RFC5988, October 2010, <https://www.rfc-editor.org/info/rfc5988>.

[RFC6202] Loreto, S., Saint-Andre, P., Salsano, S., and G. Wilkins, "Known Issues and Best Practices for the Use of Long Polling and Streaming in Bidirectional HTTP", RFC 6202, DOI 10.17487/RFC6202, April 2011, <https://www.rfc-editor.org/info/rfc6202>.

[RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, DOI 10.17487/RFC6347, January 2012, <https://www.rfc-editor.org/info/rfc6347>.

[RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, DOI 10.17487/RFC6690, August 2012, <https://www.rfc-editor.org/info/rfc6690>.

[RFC7049] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", RFC 7049, DOI 10.17487/RFC7049, October 2013, <https://www.rfc-editor.org/info/rfc7049>.

[RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <https://www.rfc-editor.org/info/rfc7230>.

[RFC7231] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", RFC 7231, DOI 10.17487/RFC7231, June 2014, <https://www.rfc-editor.org/info/rfc7231>.

[RFC7641] Hartke, K., "Observing Resources in the Constrained Application Protocol (CoAP)", RFC 7641, DOI 10.17487/RFC7641, September 2015, <https://www.rfc-editor.org/info/rfc7641>.

[RFC7959] Bormann, C. and Z. Shelby, Ed., "Block-Wise Transfers in the Constrained Application Protocol (CoAP)", RFC 7959, DOI 10.17487/RFC7959, August 2016, <https://www.rfc-editor.org/info/rfc7959>.

[RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <https://www.rfc-editor.org/info/rfc8446>.

[RFC8613] Selander, G., Mattsson, J., Palombini, F., and L. Seitz, "Object Security for Constrained RESTful Environments (OSCORE)", RFC 8613, DOI 10.17487/RFC8613, July 2019, <https://www.rfc-editor.org/info/rfc8613>.

[W3C.REC-exi-20110310]
Schneider, J. and T. Kamiya, "Efficient XML Interchange (EXI) Format 1.0", World Wide Web Consortium Recommendation REC-exi-20110310, 10 March 2011, <http://www.w3.org/TR/2011/REC-exi-20110310>.

[W3C.REC-html5-20141028]
Hickson, I., Berjon, R., Faulkner, S., Leithead, T., Navara, E., O'Connor, T., and S. Pfeiffer, "HTML5", World Wide Web Consortium Recommendation REC-html5-20141028, 28 October 2014, <http://www.w3.org/TR/2014/REC-html5-20141028>.

9.2. Informative References

[CollectionJSON]
Amundsen, M., "Collection+JSON - Document Format", February 2013, <http://amundsen.com/media-types/collection/format/>.

[I-D.bormann-core-media-content-type-format]
Bormann, C., "On Media-Types, Content-Types, and related terminology", Work in Progress, Internet-Draft, draft-bormann-core-media-content-type-format-01, 8 July 2019, <http://www.ietf.org/internet-drafts/draft-bormann-core-media-content-type-format-01.txt>.

[I-D.handrews-json-schema-validation]
Wright, A., Andrews, H., and B. Hutton, "JSON Schema Validation: A Vocabulary for Structural Validation of JSON", Work in Progress, Internet-Draft, draft-handrews-json-schema-validation-02, 17 September 2019, <http://www.ietf.org/internet-drafts/draft-handrews-json-schema-validation-02.txt>.

[I-D.hartke-core-apps]
Hartke, K., "CoRE Applications", Work in Progress, Internet-Draft, draft-hartke-core-apps-08, 22 October 2018, <http://www.ietf.org/internet-drafts/draft-hartke-core-apps-08.txt>.

[I-D.ietf-ace-dtls-authorize]
Gerdes, S., Bergmann, O., Bormann, C., Selander, G., and L. Seitz, "Datagram Transport Layer Security (DTLS) Profile for Authentication and Authorization for Constrained Environments (ACE)", Work in Progress, Internet-Draft, draft-ietf-ace-dtls-authorize-09, 18 December 2019, <http://www.ietf.org/internet-drafts/draft-ietf-ace-dtls-authorize-09.txt>.

[I-D.ietf-ace-oauth-authz]
Seitz, L., Selander, G., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "Authentication and Authorization for Constrained Environments (ACE) using the OAuth 2.0 Framework (ACE-OAuth)", Work in Progress, Internet-Draft, draft-ietf-ace-oauth-authz-33, 6 February 2020, <http://www.ietf.org/internet-drafts/draft-ietf-ace-oauth-authz-33.txt>.

[I-D.ietf-ace-oscore-profile]
Palombini, F., Seitz, L., Selander, G., and M. Gunnarsson, "OSCORE profile of the Authentication and Authorization for Constrained Environments Framework", Work in Progress, Internet-Draft, draft-ietf-ace-oscore-profile-10, 9 March 2020, <http://www.ietf.org/internet-drafts/draft-ietf-ace-oscore-profile-10.txt>.

[I-D.ietf-core-coap-pubsub]
Koster, M., Keranen, A., and J. Jimenez, "Publish-Subscribe Broker for the Constrained Application Protocol (CoAP)", Work in Progress, Internet-Draft, draft-ietf-core-coap-pubsub-09, 30 September 2019, <http://www.ietf.org/internet-drafts/draft-ietf-core-coap-pubsub-09.txt>.

[IANA-CoAP-media]
"CoAP Content-Formats", n.d., <http://www.iana.org/assignments/core-parameters/core-parameters.xhtml#content-formats>.

[IANA-media-types]
"Media Types", n.d., <http://www.iana.org/assignments/media-types/media-types.xhtml>.

[RFC5789] Dusseault, L. and J. Snell, "PATCH Method for HTTP", RFC 5789, DOI 10.17487/RFC5789, March 2010, <https://www.rfc-editor.org/info/rfc5789>.

[RFC6763] Cheshire, S. and M. Krochmal, "DNS-Based Service Discovery", RFC 6763, DOI 10.17487/RFC6763, February 2013, <https://www.rfc-editor.org/info/rfc6763>.

[RFC6943] Thaler, D., Ed., "Issues in Identifier Comparison for Security Purposes", RFC 6943, DOI 10.17487/RFC6943, May 2013, <https://www.rfc-editor.org/info/rfc6943>.

[RFC7228] Bormann, C., Ersue, M., and A. Keranen, "Terminology for Constrained-Node Networks", RFC 7228, DOI 10.17487/RFC7228, May 2014, <https://www.rfc-editor.org/info/rfc7228>.

[RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <https://www.rfc-editor.org/info/rfc7252>.

[RFC7320] Nottingham, M., "URI Design and Ownership", BCP 190, RFC 7320, DOI 10.17487/RFC7320, July 2014, <https://www.rfc-editor.org/info/rfc7320>.

[RFC7925] Tschofenig, H., Ed. and T. Fossati, "Transport Layer Security (TLS) / Datagram Transport Layer Security (DTLS) Profiles for the Internet of Things", RFC 7925, DOI 10.17487/RFC7925, July 2016, <https://www.rfc-editor.org/info/rfc7925>.

[RFC8075] Castellani, A., Loreto, S., Rahman, A., Fossati, T., and E. Dijk, "Guidelines for Mapping Implementations: HTTP to the Constrained Application Protocol (CoAP)", RFC 8075, DOI 10.17487/RFC8075, February 2017, <https://www.rfc-editor.org/info/rfc8075>.

[RFC8132] van der Stok, P., Bormann, C., and A. Sehgal, "PATCH and FETCH Methods for the Constrained Application Protocol (CoAP)", RFC 8132, DOI 10.17487/RFC8132, April 2017, <https://www.rfc-editor.org/info/rfc8132>.

[RFC8152] Schaad, J., "CBOR Object Signing and Encryption (COSE)", RFC 8152, DOI 10.17487/RFC8152, July 2017, <https://www.rfc-editor.org/info/rfc8152>.

[RFC8259] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", STD 90, RFC 8259, DOI 10.17487/RFC8259, December 2017, <https://www.rfc-editor.org/info/rfc8259>.

[RFC8392] Jones, M., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "CBOR Web Token (CWT)", RFC 8392, DOI 10.17487/RFC8392, May 2018, <https://www.rfc-editor.org/info/rfc8392>.

[RFC8428] Jennings, C., Shelby, Z., Arkko, J., Keranen, A., and C. Bormann, "Sensor Measurement Lists (SenML)", RFC 8428, DOI 10.17487/RFC8428, August 2018, <https://www.rfc-editor.org/info/rfc8428>.

[RFC8576] Garcia-Morchon, O., Kumar, S., and M. Sethi, "Internet of Things (IoT) Security: State of the Art and Challenges", RFC 8576, DOI 10.17487/RFC8576, April 2019, <https://www.rfc-editor.org/info/rfc8576>.

[RFC8747] Jones, M., Seitz, L., Selander, G., Erdtman, S., and H. Tschofenig, "Proof-of-Possession Key Semantics for CBOR Web Tokens (CWTs)", RFC 8747, DOI 10.17487/RFC8747, March 2020, <https://www.rfc-editor.org/info/rfc8747>.

[W3C-TD] Kaebisch, S., Kamiya, T., McCool, M., Charpenay, V., and M. Kovatsch, "Web of Things (WoT) Thing Description", April 2020, <https://www.w3.org/TR/wot-thing-description/>.

Appendix A. Future Work

* Unreliable (best effort) communication, robust communication in network with high packet loss, 3-way commit

* Discuss directories, such as CoAP Resource Directory

* More information on how to design resources; choosing what is modeled as a resource, etc.

Authors' Addresses

Ari Keranen
Ericsson
FI-02420 Jorvas
Finland

Email: ari.keranen@ericsson.com

Matthias Kovatsch
Siemens AG
Otto-Hahn-Ring 6
D-81739 Munich
Germany

Email: matthias.kovatsch@siemens.com

Klaus Hartke
Ericsson
Torshamnsgatan 23
SE-16483 Stockholm
Sweden

Email: klaus.hartke@ericsson.com