

Network Working Group
Internet-Draft
Intended status: Informational
Expires: 30 August 2022

A. Keranen
Ericsson
M. Kovatsch
Huawei Technologies
K. Hartke
26 February 2022

Guidance on RESTful Design for Internet of Things Systems
draft-irtf-t2trg-rest-iot-09

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 30 August 2022.

Copyright Notice

Copyright (c) 2022 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

Table of Contents

1. Introduction	3
2. Terminology	4
3. Basics	7
3.1. Architecture	8
3.2. System design	10
3.3. Uniform Resource Identifiers (URIs)	11
3.4. Representations	12
3.5. HTTP/CoAP Methods	13
3.5.1. GET	14
3.5.2. POST	14
3.5.3. PUT	15
3.5.4. DELETE	15
3.5.5. FETCH	15
3.5.6. PATCH	16
3.6. HTTP/CoAP Status/Response Codes	16
4. REST Constraints	16
4.1. Client-Server	17
4.2. Stateless	17
4.3. Cache	18
4.4. Uniform Interface	18
4.5. Layered System	19
4.6. Code-on-Demand	20
5. Hypermedia-driven Applications	20
5.1. Motivation	21
5.2. Knowledge	21
5.3. Interaction	22
5.4. Hypermedia-driven Design Guidance	22
6. Design Patterns	23
6.1. Collections	23
6.2. Calling a Procedure	23
6.2.1. Instantly Returning Procedures	24
6.2.2. Long-running Procedures	24
6.2.3. Conversion	24
6.2.4. Events as State	25
6.3. Server Push	26
7. Security Considerations	27
8. Acknowledgement	28
9. References	28
9.1. Normative References	28
9.2. Informative References	31
Authors' Addresses	34

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 lightweight and enable loose coupling of system components, they are a good fit for various Internet of Things (IoT) applications, which in general aim at interconnecting the physical world with the virtual world. 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:

- * accommodating for constrained devices [RFC7228], 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, e.g., in energy consumption or implementation complexity)
- * facilitating efficient transfer over (often) constrained networks and lightweight processing in constrained nodes through compact and simple data formats
- * minimizing or preferably avoiding the need for human interaction through machine-understandable data formats and interaction patterns
- * enabling the system to evolve gradually in the field, as the usually large number of endpoints can not be updated simultaneously
- * having endpoints that are both clients and servers

2. Terminology

This section explains selected terminology that is commonly used in the context of RESTful design for IoT systems. For terminology of constrained nodes and networks, see [RFC7228]. Terminology on modeling of Things and their affordances (Properties, Actions, and Events) was taken from [I-D.ietf-asdf-sdf].

Action: An affordance that can potentially be used to perform a named operation on a Thing.

Action Result: A representation sent as a response by a server that does not represent resource state, but the result of the interaction with the originally addressed resource.

Affordance: An element of an interface offered for interaction, defining its possible uses or making clear how it can or should be used. The term is used here for the digital interfaces of a Thing only; it might also have physical affordances such as buttons, dials, and displays.

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; it therefore has the initiative to interact. In RESTful IoT systems it is common for nodes to have more than one role (i.e., to be 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 resources are highly dynamic and 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 Type: A string that carries the media type plus potential parameters for the representation format such as "text/plain; charset=UTF-8".

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 interact with the resource of a URI.

Dereferenceable URI: A URI that can be dereferenced, i.e., interaction with the identified resource is possible. Not all HTTP or CoAP URIs are dereferenceable, e.g., when the target resource does not exist.

Event: An affordance that can potentially be used to (recurrently) obtain information about what happened to a Thing, e.g., through server push.

Form: A hypermedia control that enables a client to construct more complex requests, e.g., to change the state of a resource or perform specific queries.

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: Information provided by a server on how to use its RESTful API; usually a URI and instructions on how to dereference it for a specific interaction. Hypermedia Controls are the serialized/encoded affordances of hypermedia systems.

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 [RFC8288]).

Media Type: An IANA-registered 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 that prefers to accept the media type "application/senml+json".

Property: An affordance that can potentially be used to read, write, and/or observe state on a Thing.

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 client and server. 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 [RFC8949], 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 the possible states of a resource that is expressed in supported representation formats. Resources can change state because of REST interactions with them, or they can change state for reasons outside of the REST model, e.g., business logic implemented on the server side such as sampling a sensor.

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. In RESTful IoT systems it is common for nodes to have more than one role (i.e., to be both server and client; see Section 3.1).

Thing: A physical item that is made available in the Internet of Things, thereby enabling digital interaction with the physical world for humans, services, and/or other Things.

Transfer protocols: In particular in the IoT domain, protocols above the 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.

Which resources exist and how they can be used is expressed by the servers in so-called affordances, which is metadata that can be included in responses (e.g., the initial response from a well-known resource) or be made available out of band (e.g., through a W3C Thing Description document [W3C-TD] from a directory). In RESTful systems, affordances are encoded as hypermedia controls of which exist two types: links that allow to navigate between resources and forms that enable clients to formulate more complex requests (e.g., to modify a resource or perform a query).

A 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 their data in client role.

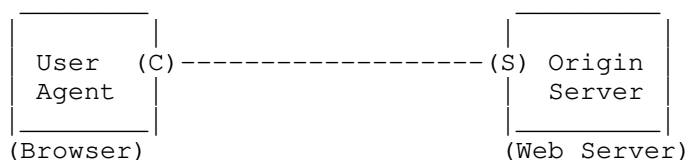


Figure 1: Client-Server Communication

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].

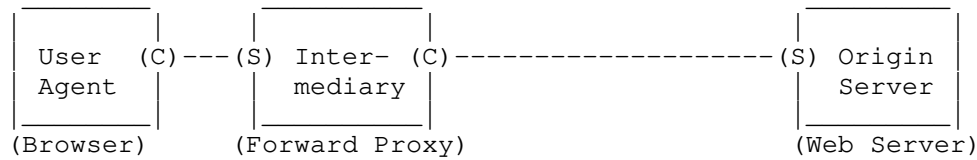


Figure 2: Communication with Forward Proxy

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).

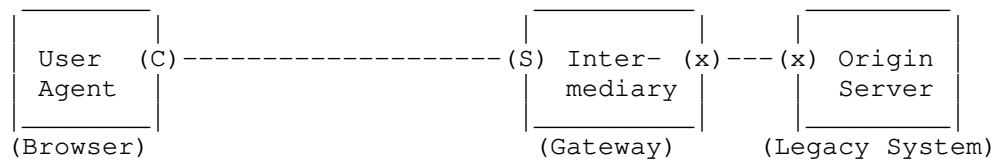


Figure 3: Communication with Reverse Proxy

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).

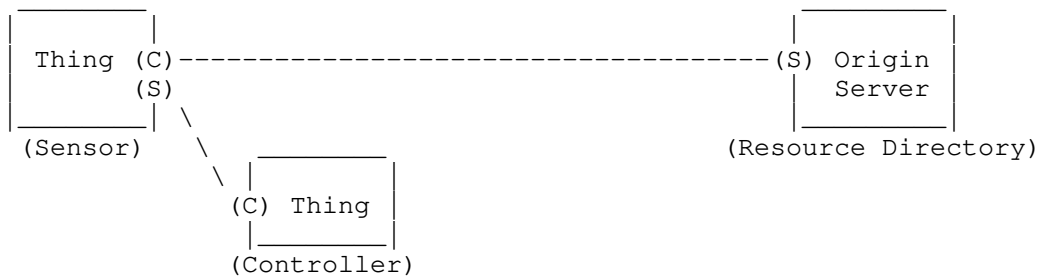


Figure 4: Constrained RESTful environments

3.2. System design

When designing a RESTful system, the primary effort goes into modeling the application as distributed state and assigning it to the different components (i.e., clients and servers). The secondary effort is then selecting or designing the necessary representation formats to exchange information and enable interaction between the components through resources.

How clients can navigate through the resource space and modify state to achieve their goals is encoded in hypermedia controls, that is, links and forms within the representations. The concept behind hypermedia controls is to provide machine-understandable "affordances" [HCI], which refer to the perceived and actual properties of a Thing and determine how it could possibly be used. A physical door may have a door knob as affordance, indicating that the door can be opened by twisting the knob; a keyhole may indicate that it can be locked. For Things in the IoT, these affordances may be serialized as two hypermedia forms, which include semantic identifiers from a controlled vocabulary (e.g., schema.org) and the instructions on how to formulate the requests for opening and locking, respectively. Overall, this allows to realize a Uniform Interface (see Section 4.4), which enables loose coupling between clients and servers.

Hypermedia controls span a kind of state machine where the nodes are resources (or action results) and the transitions are links or forms. Clients run this distributed state machine (i.e., the application) by retrieving representations, processing the data, and following the included links and/or submitting forms to modify remote state. 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. 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.

In the design, 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 (cf. load balancing). 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 aspect of RESTful API design is to model the system as a set of resources, which potentially can be created and/or deleted dynamically and whose state can be retrieved and/or modified.

Uniform Resource Identifiers (URIs) are used to indicate resources 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 "https" scheme. A hostname (e.g., a fully qualified domain name) or an IP address literal, potentially followed by a transport layer port number, are usually used for the authority component. The path and query contain data to identify a resource within the scope of the scheme-dependent naming authority (i.e., "http://www.example.org/" is a different authority than "https://www.example.org"). The fragment allows referring 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. [RFC8820] 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 protocols HTTP for "http" and "https" and CoAP 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 remainder 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 dereferenceable, e.g., the "urn" scheme can be used to construct unique names in registered namespaces. In particular the "urn:dev" URI [RFC9039] details multiple ways for generating and representing endpoint identifiers of IoT devices.

The query parameters can be used to parameterize 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 used consistently. Whether the order of the query parameters matters in URIs is unspecified; they can be re-ordered, for instance by proxies. Therefore, applications should not rely on their order; see Section 3.3.4 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 a server or manipulate resource state on the (origin) server by transferring resource representations. Resource representations must have metadata that identifies the representation format used, so the representations can be interpreted correctly. This is usually a simple string such as the IANA-registered Internet Media Types. 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 [RFC8949]
- * "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" (media type plus potential parameters), 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. A PUT request applied to the target resource can have side effects on other resources.

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 starting 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 subsections 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. Needing fewer 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 [RFC8288]. 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 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, which are embedded in representations or response message headers. 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 processable 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 doorbell 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 doorbell 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 doorbell 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 doorbell 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 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 doorbell -- 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 doorbell 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 Oca, 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]
Amsüss, C. and T. Fossati, "The Constrained RESTful Application Language (CoRAL)", Work in Progress, Internet-Draft, draft-ietf-core-coral-04, 25 October 2021, <<https://www.ietf.org/archive/id/draft-ietf-core-coral-04.txt>>.

[I-D.ietf-core-dynlink]

Koster, M. and B. Silverajan, "Dynamic Resource Linking for Constrained RESTful Environments", Work in Progress, Internet-Draft, draft-ietf-core-dynlink-14, 12 July 2021, <<https://www.ietf.org/archive/id/draft-ietf-core-dynlink-14.txt>>.

[I-D.ietf-core-href]

Bormann, C. and H. Birkholz, "Constrained Resource Identifiers", Work in Progress, Internet-Draft, draft-ietf-core-href-09, 15 January 2022, <<https://www.ietf.org/archive/id/draft-ietf-core-href-09.txt>>.

[I-D.ietf-core-resource-directory]

Amsüss, C., Shelby, Z., Koster, M., Bormann, C., and P. V. D. Stok, "CoRE Resource Directory", Work in Progress, Internet-Draft, draft-ietf-core-resource-directory-28, 7 March 2021, <<https://www.ietf.org/archive/id/draft-ietf-core-resource-directory-28.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>>.

[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>>.
- [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>>.
- [RFC8288] Nottingham, M., "Web Linking", RFC 8288, DOI 10.17487/RFC8288, October 2017, <<https://www.rfc-editor.org/info/rfc8288>>.
- [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>>.
- [RFC8949] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", STD 94, RFC 8949, DOI 10.17487/RFC8949, December 2020, <<https://www.rfc-editor.org/info/rfc8949>>.

[RFC9039] Arkko, J., Jennings, C., and Z. Shelby, "Uniform Resource Names for Device Identifiers", RFC 9039, DOI 10.17487/RFC9039, June 2021, <<https://www.rfc-editor.org/info/rfc9039>>.

[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, <<https://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, <<https://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/>>.

[HCI] Interaction Design Foundation, "The Encyclopedia of Human-Computer Interaction", 2nd Ed., 2013, <<https://www.interaction-design.org/literature/book/the-encyclopedia-of-human-computer-interaction-2nd-ed>>.

[I-D.bormann-core-media-content-type-format] Bormann, C. and H. Birkholz, "On Media-Types, Content-Types, and related terminology", Work in Progress, Internet-Draft, draft-bormann-core-media-content-type-format-04, 22 February 2021, <<https://www.ietf.org/archive/id/draft-bormann-core-media-content-type-format-04.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, <<https://www.ietf.org/archive/id/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, <<https://www.ietf.org/archive/id/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-18, 4 June 2021, <<https://www.ietf.org/archive/id/draft-ietf-ace-dtls-authorize-18.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-46, 8 November 2021, <<https://www.ietf.org/archive/id/draft-ietf-ace-oauth-authz-46.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-19, 6 May 2021, <<https://www.ietf.org/archive/id/draft-ietf-ace-oscore-profile-19.txt>>.

[I-D.ietf-asdf-sdf]

Koster, M. and C. Bormann, "Semantic Definition Format (SDF) for Data and Interactions of Things", Work in Progress, Internet-Draft, draft-ietf-asdf-sdf-10, 16 January 2022, <<https://www.ietf.org/archive/id/draft-ietf-asdf-sdf-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, <<https://www.ietf.org/archive/id/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>>.
- [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>>.
- [RFC8820] Nottingham, M., "URI Design and Ownership", BCP 190, RFC 8820, DOI 10.17487/RFC8820, June 2020, <<https://www.rfc-editor.org/info/rfc8820>>.
- [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/>>.

Authors' Addresses

Ari Keranen
Ericsson
FI-02420 Jorvas
Finland
Email: ari.keranen@ericsson.com

Matthias Kovatsch
Huawei Technologies
Riesstr. 25
D-80992 Munich
Germany
Email: matthias.kovatsch@huawei.com

Klaus Hartke
Email: hartke@projectcool.de

Network Working Group
Internet-Draft
Intended status: Informational
Expires: August 22, 2021

M. Sethi
Ericsson
B. Sarikaya
Denpel Informatique
D. Garcia-Carrillo
University of Oviedo
February 18, 2021

Secure IoT Bootstrapping: A Survey
draft-sarikaya-t2trg-sbootstrapping-11

Abstract

This draft provides an overview of the various terms that are used when discussing bootstrapping of IoT devices. We document terms that have been used within the IETF as well as other standards bodies. We investigate if the terms refer to the same phenomena or have subtle differences. We provide recommendations on the applicability of terms in different contexts. Finally, this document presents a survey of secure bootstrapping mechanisms available for smart objects that are part of an Internet of Things (IoT) network. The survey does not prescribe any one mechanism and rather presents IoT developers with different options to choose from, depending on their use-case, security requirements, and the user interface available on their IoT devices.

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 August 22, 2021.

Copyright Notice

Copyright (c) 2021 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	3
2. Terminology	4
3. Usage of bootstrapping terminology in standards	4
3.1. Device Provisioning Protocol (DPP)	4
3.2. Open Mobile Alliance (OMA) Lightweight M2M (LwM2M)	5
3.3. Open Connectivity Foundation (OCF)	6
3.4. Bluetooth	7
3.5. Fast IDentity Online (FIDO) alliance	7
3.6. Internet Engineering Task Force (IETF)	8
3.6.1. Enrollment over Secure Transport (EST)	8
3.6.2. Bootstrapping Remote Secure Key Infrastructures (BRSKI)	8
3.6.3. Secure Zero Touch Provisioning	8
3.6.4. Nimble out-of-band authentication for EAP (EAP-NOOB)	9
4. Comparison	9
5. Recommendations	9
6. Classification of available mechanisms	10
7. IoT Device Bootstrapping Methods	11
7.1. Managed Methods	11
7.1.1. Bootstrapping in LPWAN	13
7.2. Peer-to-Peer or Ad-hoc Methods	14
7.3. Leap-of-faith/Opportunistic Methods	15
7.4. Hybrid Methods	16
8. Security Considerations	16
9. IANA Considerations	17
10. Acknowledgements	17
11. Informative References	17
Authors' Addresses	23

1. Introduction

We informally define bootstrapping as "any process that takes place before a device can become operational". While bootstrapping is necessary for all computing devices, until recently, most of our devices typically had sufficient computing power and user interface (UI) for ensuring somewhat smooth operation. For example, a typical laptop device required the user to setup a username/password as well as enter network settings for Internet connectivity. Following these steps ensured that the laptop was fully operational.

The problem of bootstrapping is however exacerbated for Internet of Things (IoT) networks. The size of an IoT network varies from a couple of devices to tens of thousands, depending on the application. Smart objects/things/devices in IoT networks are produced by a variety of vendors and are typically heterogeneous in terms of the constraints on their power supply, communication capability, computation capacity, and user interfaces available. This problem of bootstrapping in IoT was identified by Sethi et al. [Sethi14] while developing a bootstrapping solution for smart displays. Although this document focuses on bootstrapping terminology and methods for IoT devices, we do not exclude bootstrapping related terminology used in other contexts.

Bootstrapping devices typically also involves providing them with some sort of network connectivity. Indeed, the functionality of a disconnected device is rather limited. Bootstrapping devices often assumes that some network has been setup a-priori. Setting up and maintaining a network itself is challenging. For example, users may need to configure the network name (called as Service Set Identifier (SSID) in Wi-Fi networks) and passphrase before new devices can be bootstrapped. Specifications such as the Wi-Fi Alliance Simple Configuration [simpleconn] help users setup networks. However, this document is only focused on terminology and processes associated with bootstrapping devices and excludes any discussion on setting up networks before devices can be bootstrapped.

In addition to our informal definition, it is helpful to look at other definitions of bootstrapping. The IoT@Work project defines bootstrapping in the context of IoT as "the process by which the state of a device, a subsystem, a network, or an application changes from not operational to operational" [iotwork]. Vermillard [vermillard], on the other hand, describes bootstrapping as the procedure by which an IoT device gets the URLs and secret keys for reaching the necessary servers. Vermillard notes that the same process is useful for re-keying, upgrading the security schemes, and for redirecting the IoT devices to new servers.

There are several terms that have often been used in the context of bootstrapping:

- o Bootstrapping
- o Provisioning
- o Onboarding
- o Enrollment
- o Commissioning
- o Initialization
- o Configuration
- o Registration

We attempt to find out whether all these terms refer to the same phenomena. We begin by looking at how these terms have been used in various standards and standardization bodies in Section 3. We then summarize our understanding in Section 4, and provide our recommendations on their usage in Section 5. Section 6 provides a taxonomy of bootstrapping methods and Section 7 categorizes methods according to the taxonomy.

2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119][RFC8174].

3. Usage of bootstrapping terminology in standards

To better understand bootstrapping related terminology, let us first look at the terms used by some existing specifications:

3.1. Device Provisioning Protocol (DPP)

The Wi-Fi Alliance Device provisioning protocol (DPP) [dpp] describes itself as a standardized protocol for providing user friendly Wi-Fi setup while maintaining or increasing the security. DPP relies on a configurator, e.g. a smartphone application, for setting up all other devices, called enrollees, in the network. DPP has the following three phases/sub-protocols:

- o **Bootstrapping:** The configurator obtains bootstrapping information from the enrollee using an out-of-band channel such as scanning a QR code or tapping NFC. The bootstrapping information includes the public-key of the device and metadata such as the radio channel on which the device is listening.
- o **Authentication:** In DPP, either the configurator or the enrollee can initiate the authentication protocol. The side initiating the authentication protocol is called as the initiator while the other side is called the responder. The authentication sub-protocol provides authentication of the responder to an initiator. It can optionally authenticate the initiator to the responder (only if the bootstrapping information was exchange out-of-band in both directions).
- o **Configuration:** Using the key established from the authentication protocol, the enrollee asks the configurator for network information such as the SSID and passphrase of the access point.

3.2. Open Mobile Alliance (OMA) Lightweight M2M (LwM2M)

The OMA LwM2M specification [oma] defines an architecture where a new device (LwM2M client) contacts a Bootstrap-server which is responsible for "provisioning" essential information such as credentials. After receiving this essential information, the LwM2M client device "registers" itself with one or more LwM2M Servers which will manage the device during its lifecycle. The current standard defines the following four bootstrapping modes:

- o **Factory Bootstrap:** An IoT device in this case is configured with all the necessary bootstrap information during manufacturing and prior to its deployment.
- o **Bootstrap from Smartcard:** An IoT device retrieves and processes all the necessary bootstrap data from a Smartcard.
- o **Client Initiated Bootstrap:** This mode provides a mechanism for an IoT client device to retrieve the bootstrap information from a Bootstrap Server. This requires the client device to have an account at the Bootstrap Server and credentials to obtain the necessary information securely.
- o **Server Initiated Bootstrap:** In this bootstrapping mode, the bootstrapping server configures all the bootstrap information on the IoT device without receiving a request from the client. This means that the bootstrap server needs to know if a client IoT Device is ready for bootstrapping before it can be configured.

For example, a network may inform the bootstrap server of a new connecting IoT client device.

3.3. Open Connectivity Foundation (OCF)

The Open Connectivity Foundation (OCF) [ocf] defines the process before a device is operational as onboarding. The first step of this onboarding process is "configuring" the ownership, i.e., establishing a legitimate user that owns the device. For this, the user is supposed to use an Onboarding tool (OBT) and an Owner Transfer Methods (OTM).

The OBT is described as a logical entity that may be implemented on a single or multiple entities such as a home gateway, a device management tool, etc. OCF lists several optional OTMs. At the end of the execution of an OTM, the onboarding tool must have "provisioned" an Owner Credential onto the device. The following owner transfer methods are specified:

- o Just works: Performs an un-authenticated Diffie-Hellman key exchange over Datagram Transport Layer Security (DTLS). The key exchange results in a symmetric session key which is later used for provisioning. Naturally, this mode is vulnerable to Man-in-The-Middle (MiTM) attackers.
- o Random PIN: The device generates a PIN code that is entered into the onboarding tool by the user. This pin code is used together with TLS-PSK ciphersuites for establishing a symmetric session key. OCF recommends PIN codes to have an entropy of 40 bits.
- o Manufacturer certificate: An onboarding tool authenticates the device by verifying a manufacturer installed certificate. Similarly, the device may authenticate the onboarding tool by verifying its signature.
- o Vendor specific: Vendors implement their own transfer method that accommodates any specific device constraints.

Once the onboarding tool and the new device have authenticated and established secure communication, the onboarding tool "provisions"/"configures" the device with Owner credentials. Owner credentials may consist of certificates, shared keys, or Kerberos tickets for example.

The OBT additionally configures/provisions information about the Access Management Service (AMS), the Credential Management Service (CMS), and the credentials for interacting with them. The AMS is

responsible for provisioning access control entries, while the CMS provisions security credentials necessary for device operation.

3.4. Bluetooth

Bluetooth mesh provisioning. Beacons for discovery. Public-key exchange followed by authentication. Finally provisioning of the network key and unicast address. To be expanded.

3.5. Fast IDentity Online (FIDO) alliance

The Fast IDentity Online Alliance (FIDO) is currently specifying an automatic onboarding protocol for IoT devices [fidospec]. The goal of this protocol is to provide a new IoT device with information for interacting securely with an online IoT platform. This protocol allows owners to choose the IoT platform for their devices at a late stage in the device lifecycle. The draft specification refers to this feature as "late binding".

The FIDO IoT protocol itself is composed of one Device Initialization (DI) protocol and 3 Transfer of Ownership (TO) protocols T00, T01, T02. Protocol messages are encoded in Concise Binary Object Representation (CBOR) [RFC8949] and can be transported over application layer protocols such as Constrained Application Protocol (CoAP) [RFC7252] or directly over TCP, Bluetooth etc. FIDO IoT however assumes that the device already has IP connectivity to a rendezvous server. Establishing this initial IP connectivity is explicitly stated as "out-of-scope" but the draft specification hints at the usage of Hypertext Transfer Protocol (HTTP) [RFC7230] proxies for IP networks and other forms of tunneling for non-IP networks.

The specification only provides a non-normative example of the DI protocol which must be executed in the factory during device manufacture. This protocol embeds initial ownership and manufacturing credentials into Restricted Operation Environment (ROE) on the device. The initial information embedded also includes a unique device identifier (called as GUID in the specification). After DI is executed, the manufacturer has an ownership voucher which is passed along the supply chain to the device owner.

When a device is unboxed and powered on by the new owner, the device discovers a network-local or an Internet-based rendezvous server. Protocols (T00, T01, and T02) between the device, the rendezvous server, and the new owner (as the owner onboarding service) ensure that the device and the new owner are able to authenticate each other. Thereafter, the new owner establishes cryptographic control of the device and provides it with credentials of the IoT platform which the device should use.

3.6. Internet Engineering Task Force (IETF)

In this section, we will look at some IETF standards and draft specifications related to IoT bootstrapping.

3.6.1. Enrollment over Secure Transport (EST)

Enrollment over Secure Transport (EST) [RFC7030] defines a profile of Certificate Management over CMS (CMC) [RFC5272]. EST relies on Hypertext Transfer Protocol (HTTP) and Transport Layer Security (TLS) for exchanging CMC messages and allows client devices to obtain client certificates and associated Certification Authority (CA) certificates. A companion specification for using EST over secure CoAP has also been standardized [I-D.ietf-ace-coap-est]. EST assumes that some initial information is already distributed so that EST client and servers can perform mutual authentication before continuing with protocol. [RFC7030] further defines "Bootstrap Distribution of CA Certificates" which allows minimally configured EST clients to obtain initial trust anchors. It relies on human users to verify information such as the CA certificate "fingerprint" received over the unauthenticated TLS connection setup. After successful completion of this bootstrapping step, clients can proceed to the enrollment step during which they obtain client certificates and associated CA certificates.

3.6.2. Bootstrapping Remote Secure Key Infrastructures (BRSKI)

The ANIMA working group is working on a bootstrapping solution for devices that relies on 802.1AR vendor certificates called Bootstrapping Remote Secure Key Infrastructures (BRSKI) [I-D.ietf-anima-bootstrapping-keyinfra]. In addition to vendor installed IEEE 802.1AR certificates, a vendor based service on the Internet is required. Before being authenticated, a new device only needs link-local connectivity, and does not require a routable address. When a vendor provides an Internet based service, devices can be forced to join only specific domains. The document highlights that the described solution is aimed in general at non-constrained (i.e. class 2+ defined in [RFC7228]) devices operating in a non-challenged network. It claims to scale to thousands of devices located in hostile environments, such as ISP provided CPE devices which are drop-shipped to the end user.

3.6.3. Secure Zero Touch Provisioning

[RFC8572] defines a bootstrapping strategy for enabling devices to securely obtain all the configuration information with no installer input, beyond the actual physical placement and connection of cables. Their goal is to enable a secure NETCONF [RFC6241] or RESTCONF

[RFC8040] connection to the deployment specific network management system (NMS). This bootstrapping method requires the devices to be configured with trust anchors in the form of X.509 certificates. [RFC8572] is similar to BRSKI based on [RFC8366].

3.6.4. Nimble out-of-band authentication for EAP (EAP-NOOB)

EAP-NOOB [I-D.ietf-emu-eap-noob] defines an EAP method where the authentication is based on a user-assisted out-of-band (OOB) channel between the server and peer. It is intended as a generic bootstrapping solution for IoT devices which have no pre-configured authentication credentials and which are not yet registered on the authentication server. This method claims to be more generic than most ad-hoc bootstrapping solutions in that it supports many types of OOB channels. The exact in-band messages and OOB message contents are specified and not the OOB channel details. EAP-NOOB also supports IoT devices with only output (e.g. display) or only input (e.g. camera). It makes combined use of both secrecy and integrity of the OOB channel for more robust security than the ad-hoc solutions.

4. Comparison

There are several stages before a device becomes fully operational. This typically involves establishing some initial trust after which credentials and other parameters are configured. For DPP, bootstrapping is the first step before authentication and provisioning of credentials can occur. For EST, bootstrapping happens as the first step when the client devices have no certificates available for starting enrollment. Provisioning/configuring are terms used for providing additional information to devices before they are fully operational. For example, credentials are provisioned onto the device. But before credential provisioning, a device is bootstrapped and authenticated. Some protocols may only deal with parts of the process. For example, TLS maybe used for authentication after bootstrapping. A separate device management protocol then may run over this TLS tunnel for provisioning operational information and credentials.

5. Recommendations

- o It is recommended that the IETF use the term "bootstrapping" for the initial (authentication) step that a device must perform. Bootstrapping will likely happen before the device has obtained full network connectivity.
- o It is recommended to use the term "provisioning"/"configuring" for the process of providing necessary information to a device to

become operational after initial authentication is complete. As is evident from above, provisioning and configuring may include bootstrapping and authentication as a sub protocol.

- o IETF specifications should aim to avoid mixing terminology or adding new terminology for better consistency.

6. Classification of available mechanisms

Given the large number of bootstrapping protocols and related specifications, it can be helpful to classify them. We categorize the available bootstrapping solutions into the following major classes:

- o **Managed methods:** These methods rely on pre-established trust relations and authentication credentials. They typically utilize centralized servers for authentication, although several such servers may join to form a distributed federation. Example methods include Extensible Authentication Protocol (EAP) [RFC3748], Generic Bootstrapping Architecture (GBA) [TS33220], Kerberos [RFC4120], Bootstrapping Remote Secure Key Infrastructures (BRSKI) and vendor certificates [vendorcert]. EAP Transport Layer Security EAP-TLS [I-D.ietf-emu-eap-tls13] for instance assumes that both the client and the server have certificates to authenticate each other. Based on this authentication, the server authorizes the client for network access. The Eduroam federation [RFC7593] uses a network of such servers to support roaming clients.
- o **Opportunistic and leap-of-faith methods:** In these methods, rather than verifying the initial authentication, the continuity of the initial identity or connection is verified. Some of these methods assume that the attacker is not present during the initial setup. Example methods include Secure Neighbor Discovery (SEND) [RFC3971] and Cryptographically Generated Addresses (CGA) [RFC3972], Wifi Protected Setup (WPS) push button [wps], and Secure Shell (SSH) [RFC4253].
- o **Peer-to-Peer (P2P) and Ad-hoc methods:** These bootstrapping methods do not rely on any pre-established credentials. Instead, the bootstrapping protocol results in credentials being established for subsequent secure communication. Such bootstrapping methods typically perform an unauthenticated Diffie-Hellman exchange [dh] and then use an out-of-band (OOB) communication channel to prevent a man-in-the-middle attack (MitM). Various secure device pairing protocols fall in this category. Based on how the OOB channel is used, the P2P methods can be further classified into two sub categories:

- * Key derivation: Contextual information received over the OOB channel is used for shared key derivation. For example, [proximate] relies on the common radio environment of the devices being paired to derive the shared secret which would then be used for secure communication.
- * Key confirmation: A Diffie-Hellman key exchange occurs over the insecure network and the established key is used to authenticate with the help of the OOB channel. For example, Bluetooth simple pairing [SimplePairing] use the OOB channel to ask the user to compare pins and approve the completed exchange.
- o Hybrid methods: Most deployed methods are hybrid and use components from both managed and ad-hoc methods. For instance, central management may be used for devices after they have been registered with the server using ad-hoc registration methods.

It is important to note here that categorization of different methods is not always easy or clear. For example, all the opportunistic and leap-of-faith methods become managed methods after the initial vulnerability window. The choice of bootstrapping method used for devices depends heavily on the business case. Questions that may govern the choice include: What third parties are available? Who wants to retain control or avoid work? In each category, there are many different methods of secure bootstrapping available. The choice of the method may also be governed by the type of device being bootstrapped.

7. IoT Device Bootstrapping Methods

In this section we look at additional bootstrapping protocols for IoT devices which are not covered in Section 3. Protocols already covered in Section 3 however are mentioned in their respective classes. This list is non-exhaustive.

7.1. Managed Methods

EAP-TLS is a widely used EAP method for network access authentication [I-D.ietf-emu-eap-tls13]. It requires certificate-based mutual authentication and a public key infrastructure. The ZigBee Alliance has specified an IPv6 stack for IEEE 802.15.4 [IEEE802.15.4] devices used in smart meters developed primarily for SEP 2.0 (Smart Energy Profile) application layer traffic [SEP2.0]. The ZigBee IP stack uses EAP-TLS for secure bootstrapping of devices.

EAP-PSK [RFC4764] is another EAP method that realizes mutual authentication and session key derivation using a Pre-Shared Key

(PSK). Given the light-weight nature of EAP-PSK, it can be suitable for resource-constrained devices. However, secure distribution of a large number of PSKs can be challenging.

CoAP-EAP [I-D.marin-ace-wg-coap-eap] defines a bootstrapping service for IoT. The authors propose transporting EAP over CoAP [RFC7252] for the constrained link, and communication with AAA infrastructures in the non-constrained link. While the draft discusses the use of EAP-PSK, the authors claim that they are specifying a new EAP lower layer and any EAP method which results in generation is suitable.

Protocol for Carrying Authentication for Network Access (PANA) [RFC5191] is a network layer protocol with which a node can authenticate itself to gain access to the network. PANA does not define a new authentication protocol and rather uses EAP over User Datagram Protocol (UDP) for authentication.

Colin O'Flynn [I-D.oflynn-core-bootstrapping] proposes the use of PANA for secure bootstrapping of resource constrained devices. He demonstrates how a 6LoWPAN Border Router (PANA Authentication Agent (PAA)) can authenticate the identity of a joining constrained device (PANA Client). Once the constrained device has been successfully authenticated, the border router can also provide network and security parameters to the joining device.

Hernandez-Ramos et al. [panaiot] also use EAP-TLS over PANA for secure bootstrapping of smart objects. They extend their bootstrapping scheme for configuring additional keys that are used for secure group communication.

Generic Bootstrapping Architecture (GBA) is another bootstrapping method that falls in centralized category. GBA is part of the 3GPP standard [TS33220] and is based on 3GPP Authentication and Key Agreement (3GPP AKA). GBA is an application independent mechanism to provide a client application (running on the User equipment (UE)) and any application server with a shared session secret. This shared session secret can subsequently be used to authenticate and protect the communication between the client application and the application server. GBA authentication is based on the permanent secret shared between the UE's Universal Integrated Circuit Card (UICC), for example SIM card, and the corresponding profile information stored within the cellular network operator's Home Subscriber System (HSS) database. [I-D.sethi-gba-constrained] describes a resource-constrained adaptation of GBA for IoT.

The four bootstrapping modes specified by the Open Mobile Alliance (OMA) Light-weight M2M (LwM2M) standard require some sort of pre-

provisioned credentials on the device. All the four modes are examples of managed bootstrapping methods.

The Kerberos protocol [RFC4120] is a network authentication protocol that allows several endpoints to communicate over an insecure network. Kerberos relies on a symmetric cryptography scheme and requires a trusted third party, that guarantees the identities of the various actors. It relies on the use of "tickets" for nodes to prove identity to one another in a secure manner. There has been research work on using Kerberos for IoT devices [kerberosiot].

It is also important to mention some of the work done on implicit certificates and identity-based cryptographic schemes [himmo], [implicit]. While these are interesting and novel schemes that can be a part of securely bootstrapping devices, at this point, it is hard to speculate on whether such schemes would see large-scale deployment in the future.

7.1.1. Bootstrapping in LPWAN

Low Power Wide Area Network (LPWAN) encompasses a wide variety of technologies whose link-layer characteristics are severely constrained in comparison to other typical IoT link-layer technologies such as Bluetooth or IEEE 802.15.4. While some LPWAN technologies rely on proprietary bootstrapping solutions which are not publicly accessible, others simply ignore the challenge of bootstrapping and key distribution. In this section, we discuss the bootstrapping methods used by LPWAN technologies covered in [RFC8376].

- o LoRaWAN [LoRaWAN] describes its own protocol to authenticate nodes before allowing them join a LoRaWAN network. This process is called as joining and it is based on pre-shared keys (called AppKeys in the standard). The joining procedure comprises only one exchange (join-request and join-accept) between the joining node and the network server. There are several adaptations to this joining procedure that allow network servers to delegate authentication and authorization to a backend AAA infrastructure [RFC2904].
- o Wi-SUN Alliance Field Area Network (FAN) uses IEEE 802.1X and EAP-TLS for network access authentication. It performs a 4-way handshake to establish a session keys after EAP-TLS authentication.
- o NB-IoT relies on the traditional 3GPP mutual authentication scheme based on a shared-secret in the Subscriber Identity Module (SIM) of the device and the mobile operator.

- o Sigfox security is based on unique device identifiers and cryptographic keys. As stated in [RFC8376], although the algorithms and keying details are not publicly available, there is sufficient information to indicate that bootstrapping in Sigfox is based on pre-established credentials between the device and the Sigfox network.

From the above, it is clear that all LPWAN technologies rely on pre-provisioned credentials for authentication between a new device and the network. Thus, all of them can be categorized as managed bootstrapping methods.

7.2. Peer-to-Peer or Ad-hoc Methods

While managed methods are viable for many IoT devices, they may not be suitable or desirable in all scenarios. All the managed methods assume that some credentials are provisioned into the device. These credentials may be in the device micro-controller or in a replaceable smart card such as a SIM card. The methods also sometimes assume that the manufacturer embeds these credentials during the device manufacture on the factory floor. However, in many cases the manufacturer may not have sufficient incentive to do this. In other scenarios, it may be hard to completely trust and rely on the device manufacturer to securely perform this task. Therefore, many times, P2P or Ad-hoc methods of bootstrapping are used. We discuss a few example next.

P2P or ad-hoc bootstrapping methods are used for establishing keys and credential information for secure communication without any pre-provisioned information. These bootstrapping mechanisms typically rely on an out-of-band (OOB) channel in order to prevent man-in-the-middle (MitM) attacks. P2P and ad-hoc methods have typically been used for securely pairing personal computing devices such as smart phones. [devicepairing] provides a survey of such secure device pairing methods. Many original pairing schemes required the user to enter the same key string or authentication code to both devices or to compare and approve codes displayed by the devices. While these methods can provide reasonable security, they require user interaction that is relatively unnatural and often considered a nuisance. Thus, there is ongoing research for more natural ways of pairing devices. To reduce the amount of user-interaction required in the pairing process, several proposals use contextual or location-dependent information, or natural user input such as sound or movement, for device pairing [proximate].

The local association created between two devices may later be used for connecting/introducing one of the devices to a centralized server. Such methods would however be classified as hybrids.

EAP-NOOB [I-D.ietf-emu-eap-noob] is an example of P2P and ad-hoc bootstrapping method that establishes a security association between an IoT device (node) and an online server (unlike pairing two devices for local connections over WiFi or Bluetooth).

Thread Group commissioning [threadcommissioning] introduces a two phased process i.e. Petitioning and Joining. Entities involved are leader, joiner, commissioner, joiner router and border router. Leader is the first device in Thread network that must be commissioned using out-of-band process and is used to inject correct user generated Commissioning Credentials (can be changed later) into Thread Network. Joiner is the node that intends to get authenticated and authorized on Thread Network. Commissioner is either within the Thread Network (Native) or connected with Thread Network via a WLAN (External).

Under some topologies, Joiner Router and Border Router facilitate the Joiner node to reach Native and External Commissioner, respectively. Petitioning begins before Joining process and is used to grant sole commissioning authority to a Commissioner. After an authorized Commissioner is designated, eligible thread devices can join network. Pair-wise key is shared between Commissioner and Joiner, network parameters (such as network name, security policy, etc.,) are sent out securely (using pair-wise key) by Joiner Router to Joiner for letting Joiner to join the Thread Network. Entities involved in Joining process depends on system topology i.e. location of Commissioner and Joiner. Thread networks only operate using IPv6. Thread devices can devise GUAs (Global Unicast Addresses) [RFC4291]. Provision also exist via Border Router, for Thread device to acquire individual global address by means of DHCPv6 or using SLAAC (Stateless Address Autoconfiguration) address derived with advertised network prefix.

7.3. Leap-of-faith/Opportunistic Methods

Bergmann et al. [simplekey] develop a secure bootstrapping mechanism that does not rely on pre-provisioned credentials using resurrecting-duckling imprinting scheme. Their bootstrapping protocol involves three distinct phases: discover (the duckling node searches for network nodes that can act as mother node), imprint (the mother node imprints a shared secret establishing a secure channel once a positive response is received for the imprinting request) and configure (additional configuration information such as network prefix and default gateway are configured). In this model for bootstrapping, a small initial vulnerability window is acceptable and can be mitigated using techniques such as a Faraday Cage (securing the communication physically) to protect the environment of the mother and duck nodes, though this may be inconvenient for the user.

7.4. Hybrid Methods

[RFC7250] defines how raw public keys can be used for mutual authentication of devices and servers. The extension specified in [RFC7250] simplifies `client_certificate_type` and `server_certificate_type` to carry only `SubjectPublicKeyInfo` structure with the raw public key instead of many other parameters found in typical X.509 version 3 certificates. Each side validates the keys received with pre-configured values stored. Using raw public keys for bootstrapping can be seen as a hybrid method. This is because it generally requires an out-of-band (OOB) step (P2P/Ad-hoc) where the raw public keys [RFC7250] are provided to the authenticating entities, after which the actual authentication occurs online (managed). CoAP already provides support for using raw public keys (see Section 9.1.3.2. of [RFC7252])

8. Security Considerations

This draft does not take any posture on the security properties of the different bootstrapping protocols discussed. Specific security considerations of bootstrapping protocols are present in the respective specifications.

Nonetheless, we briefly discuss some important security aspects which are not fully explored in various specifications.

Firstly, an IoT system may deal with authorization for resources and services separately from bootstrapping and authentication in terms of timing as well as protocols. As an example, two resource-constrained devices A and B may perform mutual authentication using credentials provided by an offline third-party X before device A obtains authorization for running a particular application on device B from an online third-party Y. In some cases, authentication and authorization maybe tightly coupled, e.g., successful authentication also means successful authorization.

Secondly, re-bootstrapping of IoT devices may be required since keys have limited lifetimes and devices may be lost or resold. Protocols and systems must have adequate provisions for revocation and re-bootstrapping. Re-bootstrapping must be as secure as the initial bootstrapping regardless of whether this re-bootstrapping is done manually or automatically over the network.

Lastly, some IoT networks use a common group key for multicast and broadcast traffic. As the number of devices in a network increase over time, a common group key may not be scalable and the same network may need to be split into separate groups with different keys. Bootstrapping and provisioning protocols may need appropriate

mechanisms for identifying and distributing keys to the current member devices of each group.

9. IANA Considerations

There are no IANA considerations for this document.

10. Acknowledgements

We would like to thank Tuomas Aura, Hannes Tschofenig, and Michael Richardson for providing extensive feedback as well as Rafa Marin-Lopez for his support.

11. Informative References

- [devicepairing] Mirzadeh, S., Cruickshank, H., and R. Tafazolli, "Secure Device Pairing: A Survey", IEEE Communications Surveys and Tutorials , pp. 17-40, 2014.
- [dh] Diffie, W. and M. Hellman, "New directions in cryptography", IEEE Transactions on Information Theory , vol. 22, no. 6, pp. 644-654, 1976.
- [dpp] Wi-Fi Alliance, "Wi-Fi Device Provisioning Protocol (DPP)", Wi-Fi Alliance , 2018, <https://www.wi-fi.org/download.php?file=/sites/default/files/private/Device_Provisioning_Protocol_Specification_v1.1_1.pdf>.
- [fidospec] Fast Identity Online Alliance, "FIDO IoT Spec", Fido Alliance , August 2020, <<https://fidoalliance.org/specs/internet-of-things/FIDO-IoT-spec.html>>.
- [himmo] Garcia-Morchon, O., Rietman, R., Sharma, S., Tolhuizen, L., and J. Torre-Arce, "DTLS-HIMMO: Efficiently Securing a Post-Quantum World with a Fully-Collusion Resistant KPS", Submitted to NIST Workshop on Cybersecurity in a Post-Quantum World , version 20141225:065757, December 2014, <<https://eprint.iacr.org/2014/1008>>.
- [I-D.ietf-ace-coap-est] Stok, P., Kampanakis, P., Richardson, M., and S. Raza, "EST over secure CoAP (EST-coaps)", draft-ietf-ace-coap-est-18 (work in progress), January 2020.

- [I-D.ietf-anima-bootstrapping-keyinfra]
Pritikin, M., Richardson, M., Eckert, T., Behringer, M.,
and K. Watsen, "Bootstrapping Remote Secure Key
Infrastructures (BRSKI)", draft-ietf-anima-bootstrapping-
keyinfra-45 (work in progress), November 2020.
- [I-D.ietf-emu-eap-noob]
Aura, T., Sethi, M., and A. Peltonen, "Nimble out-of-band
authentication for EAP (EAP-NOOB)", draft-ietf-emu-eap-
noob-03 (work in progress), December 2020.
- [I-D.ietf-emu-eap-tls13]
Mattsson, J. and M. Sethi, "Using EAP-TLS with TLS 1.3",
draft-ietf-emu-eap-tls13-13 (work in progress), November
2020.
- [I-D.marin-ace-wg-coap-eap]
Marin-Lopez, R. and D. Garcia-Carrillo, "EAP-based
Authentication Service for CoAP", draft-marin-ace-wg-coap-
eap-07 (work in progress), January 2021.
- [I-D.oflynn-core-bootstrapping]
Sarikaya, B., Ohba, Y., Cao, Z., and R. Cragie, "Security
Bootstrapping of Resource-Constrained Devices", draft-
oflynn-core-bootstrapping-03 (work in progress), November
2010.
- [I-D.sethi-gba-constrained]
Sethi, M., Lehtovirta, V., and P. Salmela, "Using Generic
Bootstrapping Architecture with Constrained Devices",
draft-sethi-gba-constrained-01 (work in progress),
February 2014.
- [IEEE802.15.4]
"IEEE Std. 802.15.4-2015", April 2016,
<[http://standards.ieee.org/findstds/
standard/802.15.4-2015.html](http://standards.ieee.org/findstds/standard/802.15.4-2015.html)>.
- [implicit]
Porambage, P., Schmitt, C., Kumar, P., Gurtov, A., and M.
Ylianttila, "Pauthkey: A pervasive authentication protocol
and key establishment scheme for wireless sensor networks
in distributed iot applications", International Journal of
Distributed Sensor Networks , Hindawi Publishing
Corporation , 2014.
- [iotwork] European Commission FP7, "IoT@Work bootstrapping
architecture Deliverable D2.2", June 2011.

- [kerberosiot] Hardjono, T., "Kerberos for Internet-of-Things", February 2014, <https://kit.mit.edu/sites/default/files/documents/Kerberos_Internet_of%20Things.pdf>.
- [LoRaWAN] Sornin, N., Luis, M., Eirich, T., and T. Kramp, "LoRa Specification V1.0", January 2015, <<https://www.lora-alliance.org/portals/0/specs/LoRaWAN%20Specification%201R0.pdf>>.
- [ocf] Open Connectivity Foundation, "OCF Security Specification", Open Connectivity Foundation , June 2017, <https://openconnectivity.org/specs/OCF_Security_Specification_v1.0.0.pdf>.
- [oma] Open Mobile Alliance, "Lightweight Machine to Machine Technical Specification: Core", Open Mobile Alliance , November 2020, <www.openmobilealliance.org/release/LightweightM2M/V1_2-20201110-A/OMA-TS-LightweightM2M_Core-V1_2-20201110-A.pdf>.
- [panaiot] Hernandez-Ramos, J., Carrillo, D., Marin-Lopez, R., and A. Skarmeta, "Dynamic Security Credentials PANA-based Provisioning for IoT Smart Objects", 2nd World Forum on Internet of Things (WF-IoT) , IEEE , 2015.
- [proximate] Mathur, S., Miller, R., Varshavsky, A., Trappe, W., and N. Mandayam, "Proximate: proximity-based secure pairing using ambient wireless signals.", Proceedings of MobiSys International Conference , pp. 211-224, June 2011.
- [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>>.
- [RFC2904] Vollbrecht, J., Calhoun, P., Farrell, S., Gommans, L., Gross, G., de Bruijn, B., de Laat, C., Holdrege, M., and D. Spence, "AAA Authorization Framework", RFC 2904, DOI 10.17487/RFC2904, August 2000, <<https://www.rfc-editor.org/info/rfc2904>>.
- [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>>.

- [RFC3971] Arkko, J., Ed., Kempf, J., Zill, B., and P. Nikander, "SEcure Neighbor Discovery (SEND)", RFC 3971, DOI 10.17487/RFC3971, March 2005, <<https://www.rfc-editor.org/info/rfc3971>>.
- [RFC3972] Aura, T., "Cryptographically Generated Addresses (CGA)", RFC 3972, DOI 10.17487/RFC3972, March 2005, <<https://www.rfc-editor.org/info/rfc3972>>.
- [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>>.
- [RFC4253] Ylonen, T. and C. Lonvick, Ed., "The Secure Shell (SSH) Transport Layer Protocol", RFC 4253, DOI 10.17487/RFC4253, January 2006, <<https://www.rfc-editor.org/info/rfc4253>>.
- [RFC4291] Hinden, R. and S. Deering, "IP Version 6 Addressing Architecture", RFC 4291, DOI 10.17487/RFC4291, February 2006, <<https://www.rfc-editor.org/info/rfc4291>>.
- [RFC4764] Bersani, F. and H. Tschofenig, "The EAP-PSK Protocol: A Pre-Shared Key Extensible Authentication Protocol (EAP) Method", RFC 4764, DOI 10.17487/RFC4764, January 2007, <<https://www.rfc-editor.org/info/rfc4764>>.
- [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>>.
- [RFC5272] Schaad, J. and M. Myers, "Certificate Management over CMS (CMC)", RFC 5272, DOI 10.17487/RFC5272, June 2008, <<https://www.rfc-editor.org/info/rfc5272>>.
- [RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol (NETCONF)", RFC 6241, DOI 10.17487/RFC6241, June 2011, <<https://www.rfc-editor.org/info/rfc6241>>.
- [RFC7030] Pritikin, M., Ed., Yee, P., Ed., and D. Harkins, Ed., "Enrollment over Secure Transport", RFC 7030, DOI 10.17487/RFC7030, October 2013, <<https://www.rfc-editor.org/info/rfc7030>>.

- [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>>.
- [RFC7250] Wouters, P., Ed., Tschofenig, H., Ed., Gilmore, J., Weiler, S., and T. Kivinen, "Using Raw Public Keys in Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)", RFC 7250, DOI 10.17487/RFC7250, June 2014, <<https://www.rfc-editor.org/info/rfc7250>>.
- [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>>.
- [RFC7593] Wierenga, K., Winter, S., and T. Wolniewicz, "The eduroam Architecture for Network Roaming", RFC 7593, DOI 10.17487/RFC7593, September 2015, <<https://www.rfc-editor.org/info/rfc7593>>.
- [RFC8040] Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", RFC 8040, DOI 10.17487/RFC8040, January 2017, <<https://www.rfc-editor.org/info/rfc8040>>.
- [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>>.
- [RFC8366] Watsen, K., Richardson, M., Pritikin, M., and T. Eckert, "A Voucher Artifact for Bootstrapping Protocols", RFC 8366, DOI 10.17487/RFC8366, May 2018, <<https://www.rfc-editor.org/info/rfc8366>>.
- [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>>.
- [RFC8572] Watsen, K., Farrer, I., and M. Abrahamsson, "Secure Zero Touch Provisioning (SZTP)", RFC 8572, DOI 10.17487/RFC8572, April 2019, <<https://www.rfc-editor.org/info/rfc8572>>.

- [RFC8949] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", STD 94, RFC 8949, DOI 10.17487/RFC8949, December 2020, <<https://www.rfc-editor.org/info/rfc8949>>.
- [SEP2.0] ZigBee Alliance, "ZigBee IP Specification", March 2014, <<http://www.zigbee.org/non-menu-pages/zigbee-ip-download/>>.
- [Sethi14] Sethi, M., Oat, E., Di Francesco, M., and T. Aura, "Secure Bootstrapping of Cloud-Managed Ubiquitous Displays", Proceedings of ACM International Joint Conference on Pervasive and Ubiquitous Computing (UbiComp 2014), pp. 739-750, Seattle, USA, September 2014, <<http://dx.doi.org/10.1145/2632048.2632049>>.
- [simpleconn]
Wi-Fi Alliance, "Wi-Fi Simple Configuration", Wi-Fi Alliance, 2019, <https://www.wi-fi.org/download.php?file=/sites/default/files/private/Wi-Fi_Simple_Configuration_Technical_Specification_v2.0.7.pdf>.
- [simplekey]
Bergmann, O., Gerdes, S., and C. Bormann, "Simple Keys for Simple Smart Objects", Smart Object Security Workshop, IETF 83, March 2012.
- [SimplePairing]
Bluetooth, SIG, "Simple pairing whitepaper", Technical report, 2007.
- [threadcommissioning]
Thread Group, "Thread Commissioning", Thread Group, Inc., 2015.
- [TS33220] 3GPP, "3rd Generation Partnership Project; Technical Specification Group Services and System Aspects; Generic Authentication Architecture (GAA); Generic Bootstrapping Architecture (GBA) (Release 14)", December 2016, <<http://www.3gpp.org/DynaReport/33220.htm>>.
- [vendorcert]
IEEE std. 802.1ar-2009, "Standard for local and metropolitan area networks - secure device identity", December 2009.

[vermillard]

Vermillard, J., "Bootstrapping device security with lightweight M2M", Appeared on blog at medium.com , February 2015.

[wps]

Wi-Fi Alliance, "Wi-fi protected setup", Wi-Fi Alliance , 2007.

Authors' Addresses

Mohit Sethi
Ericsson
Hirsalantie 11
Jorvas 02420
Finland

Email: mohit@piuha.net

Behcet Sarikaya
Denpel Informatique

Email: sarikaya@ieee.org

Dan Garcia-Carrillo
University of Oviedo
Oviedo 33207
Spain

Email: garciadan@uniovi.es