

IIoT
Internet-Draft
Intended status: Standards Track
Expires: May 3, 2018

L. Geng
China Mobile
M. Zhang
M. McBride
B. Liu
Huawei
October 30, 2017

Problem Statement of Edge Computing beyond Access Network for Industrial
IoT
draft-geng-iiot-edge-computing-problem-statement-00

Abstract

This document introduces the concept of Beyond Edge Computing (BEC) which brings edge computing capabilities down to the level of customers' premises for industrial IoT use cases. The purpose of the document is to discuss the general problem statement of BEC including capabilities, and use cases. Potential technical gaps in IETF problem scope that are related to BEC are also evaluated.

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 May 3, 2018.

Copyright Notice

Copyright (c) 2017 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	2
1.1. Requirements Language	3
1.2. Terminology	3
2. The Concept and Capabilities of Beyond Edge Computing	3
2.1. Relationship between BEC and and Cloud Computing	5
3. Reference Architecture	5
4. Use Cases of BEC	7
5. Gap Analysis	9
6. IANA Considerations	10
7. Security Considerations	10
8. Acknowledgement	10
9. Normative References	10
Authors' Addresses	11

1. Introduction

Edge computing is an important network architecture particularly in the support of Industrial verticals such as Energy, Manufacturing, Healthcare, Mining and Smart City/Buildings. Edge computing will provide local compute, storage and connectivity services particularly for latency and bandwidth sensitive applications. There are several organizations which are working on edge computing definition and architecture with various emphases. For instance, ETSI MEC (previously mobile edge computing and now multi-access edge computing) looks at edge computing from the perspective of the edge of the provider network. It also has a successive convention of focusing on cellular network requirements. The Industrial Internet Consortium (IIC) and Edge Computing Consortium (ECC) works on edge computing in a more general view of industrial IoT, where edge computing nodes even closer to verticals (i.e. the very first hops where the service is connected to the network). Typically, the edge computing nodes are located at customers' premises. However, IIC and ECC are not standard organizations and they rely on communities such as IETF to provide protocols and API definitions for their architectural use especially as Operation Technology (OT), Information Technology (IT) and Communication Technology (CT) converge.

Edge computing concepts have been presented in various groups within the IETF/IRTF. The edge computing topic, similar to cloud computing,

is much too broad to tackle within the IETF. There are specific protocol/interface areas, however, that can be worked on in the IETF once we identify a specific area of focus. BEC is one of the specific area which looks at edge computing from the industrial verticals such as factory, hospital, power and city/ building perspective and down to the level of local edge support for sensors, engines, pumps and machinery.

A simple example, of BEC, is factory equipment having connected sensors which are generating data and sending to the equipment within an edge computing environment. One sensor, connected to this equipment, could generate an event, such as overheating, and a connected actuator could quickly increase fan span or reduce engine speed depending upon the data within the local edge computing node. This type of event is being controlled today within closed industrial command and control protocols. But what is not generally available is the ability for open edge computing equipment to generate predictive maintenance and command and control across factories, verticals and into the cloud. This is where we see a gap in standards definitions and an opportunity for new protocols and interfaces, in which IETF could play a particularly important role.

1.1. Requirements Language

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 [RFC2119].

1.2. Terminology

- o BEC - Beyond Edge Computing, a concept of edge computing where the edge computing devices are deployed directly at customers' premises so that beyond the access network of a service provider.

2. The Concept and Capabilities of Beyond Edge Computing

Beyond Edge Computing (BEC) looks at the on-site intelligent evolution of industrial verticals. It brings the computing capability down to the level of customer premises where devices are managed by customers therefore typically beyond the reach of access network of a service provider.

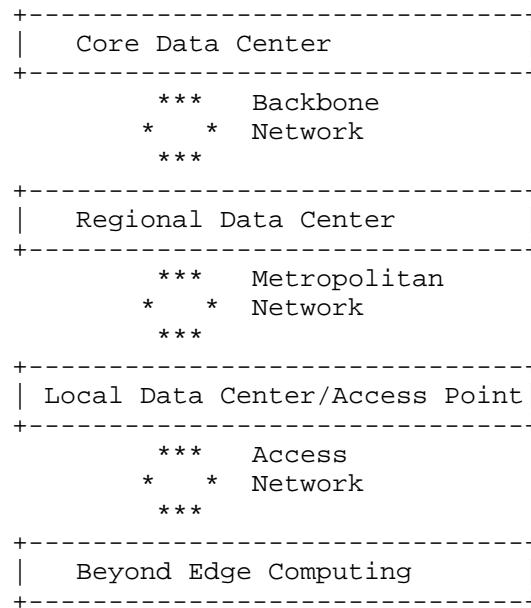


Figure 1: Beyond Edge Computing in the Network

Figure 1 illustrates the schematic diagram of BEC in terms of its position in an overall network. BEC takes care of the first hop where the service of a particular industrial vertical connects to the network. It can be regarded as an extended intelligent connectivity capability of a service provider's network to industrial verticals. Meanwhile, it also expands the cloud computing ability directly to customers' sites.

BEC has the following capabilities.

1. Heterogeneous IoT device compatibility
2. Extremely low and deterministic service latency
3. Local data pre-processing and offloading
4. Isolation of system resources
5. Offline process
6. End-to-end security
7. Distributed artificial intelligence

8. Real-time operation
9. Unified API for multi-ecosystem edge application
10. Service isolation for network slicing

2.1. Relationship between BEC and and Cloud Computing

BEC is different from Cloud Computing in the following perspectives.

- o Edge is closer to things than the Cloud, it's feasible to meet the high reliability, bounded latency or real-time requirements of verticals such AR/VR, automatic driving and smart manufacturing. For Cloud Computing, the latency is regarded as a performance index. As for Edge Computing, bounded latency is often a mandatory requirement.

- o Data can be stored at the Edge which are under the control of end users so that user's privacy can be preserved. Video surveillance, Healthcare are typical use case scenarios for this perspective.

- o Raw data can be preprocessed at the edge while only critical information is uploaded to the Cloud. In this way, Edge Computing promises lower communication cost than the traditional Cloud-only architecture.

- o The resources for Cloud Computing can be elastically allocated thanks to virtualization and resource pooling. Virtualization provides certain reliability to Cloud Computing. In comparison, the resources are normally constrained for Edge Computing. Reliability is sometimes realized through the redundant placement of physical edge devices.

- o The hardware and software of Cloud Computing are normally standardized. Devices and software being used in Edge Computing can be quite different considering various verticals are adopting Edge Computing.

3. Reference Architecture

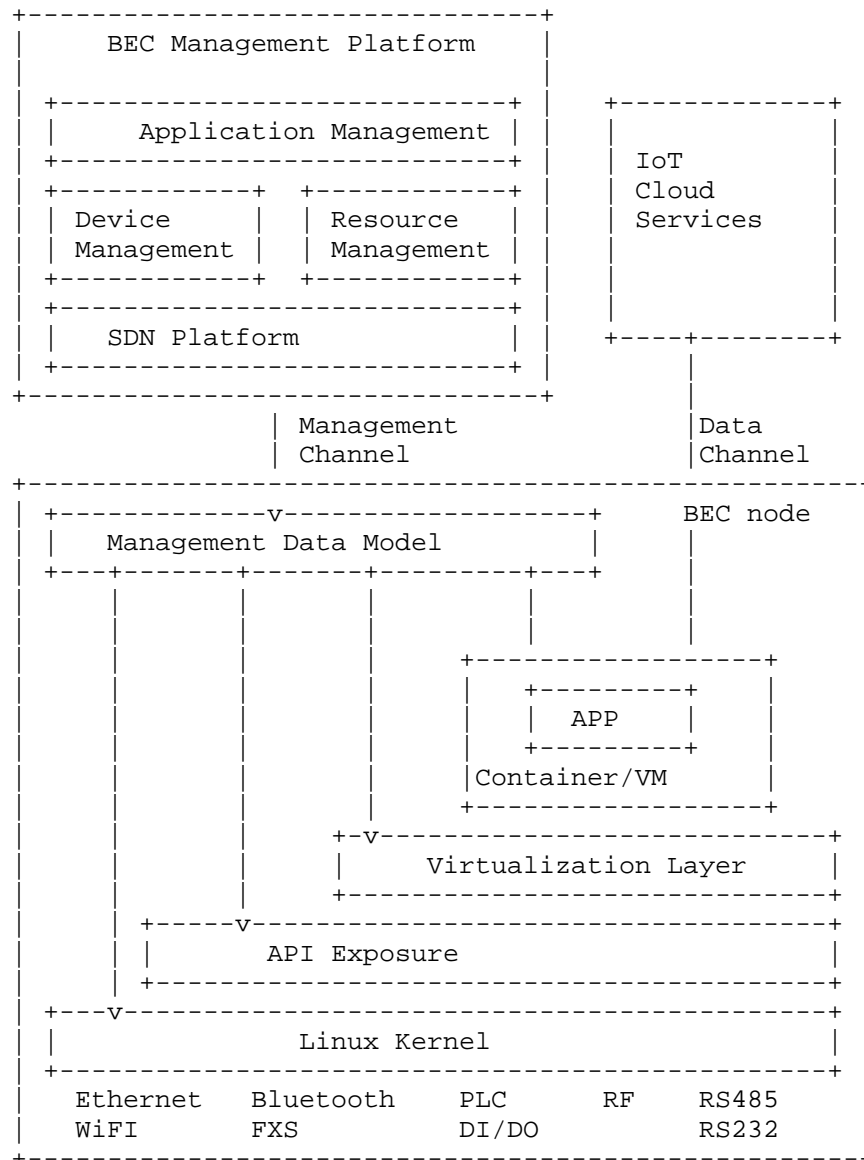


Figure 2: The Reference Architecture of Beyond Edge Computing

Figure 2 demonstrates the reference architecture of BEC system with a managed BEC node and a cloud-based management platform. An IoT

gateway is the typical form of a BEC node device. Gateways always play important role in the Cloud-Edge architecture since they are the most popular devices where verticals are provided with various capabilities such as computing, storage and networking. In addition, applications for various vertical customers are developed by themselves or third-party while deployed on demand. Giving the edge computing ability of BEC, much of the data can be processed by applications running on the gateway locally as required by vertical customers. The gateways are commonly versatile protocol speakers so that devices speaking different protocols can communicate with them. The East-West connectivity between BEC nodes might be enabled by various protocols such as OPC-UA, MQTT, TSN and other deterministic Ethernet protocols, for example EtherCat, Ethernet/IP, Profinet. To facilitate the operation of the BEC system, a central controller in the cloud is provisioned to the customer. It mainly supervises the device, virtualization resource and application life cycle of the BEC node

The key requirements of BEC are in providing distribution service entities on the customers' site (end AP, devices) to meet the growing demand for low latency, reliable, and secure vertical industries. The Computing, Storage, I/O isolation are remotely managed at the edge to provide certain dedication and quality guarantees. Agile, flexible and scalable deployment of services from operator/third party down to the edge through software entities (VM/ Containers). A light weight MANO like approach is needed to provide resource provisioning and VNF deployment. A unified API definition is needed to support the co- existence of multi-ecosystem at the BEC node. And there needs to be the ability for the edge device to map specific service requirements with an end to end network slice with certain guarantees and pass the policy identification along the path to the centralized DC.

4. Use Cases of BEC

1. Elevator Networks

Description: There are more than 15 million elevators around the world and the daily maintenance of these elevators costs elevator operators a large amount of revenue. An elevator usually carries hundreds of sensors which are generating a large amount of data to be uploaded to the cloud. The BEC nodes can preprocess the data gathered from elevator sensors so that the volume to be uploaded to the Cloud is greatly reduced. Based on the input from elevator sensors, AI programs installed on BEC nodes may locally make decisions without the intervention of the Cloud. For example, when the noise or vibration of an elevator exceeds a certain level, the

BEC node may notify elevator maintainers to reach this elevator and perform maintenance or repair.

Goal: BEC nodes are deployed into elevators to gather/preprocess/compress the data to save the communication cost. Based on the data gathered from elevator sensors, BEC nodes can assist operators to do predictive maintenance.

Requirements: Customized gateways operated by elevator providers. An open platform with VMs/containers which can hold customized Apps. These Apps are managed by elevator operators while developed by gateway vendors or any other third parties. Various connectivities are supported (2G/3G/LTE/eMTC/Ethernet) by BEC nodes. A central controller to perform configuration and management of the network. AI models are trained on the Cloud while the reasoning of these AI models are performed at the Edge.

2. Street Lights

Description: BEC nodes are placed on street lights to act as board routers of LLNs. BEC nodes may act as RSUs of vehicle networks. With AI programs installed on the BEC nodes, reasoning and decisions might be made locally at the edge. For example, BEC nodes can adjust the lights' brightness and operating hours according to environment parameters, providing illumination when needed while reducing power consumption. With sensors on trash cans, BEC nodes are aware whether a trash can is full. Trash collecting cars can communicate with the BEC nodes to determine whether to reach a trash can to collect the trash.

Goal: BEC nodes gather data from sensors which are used to monitor various information (e.g., brightness, temperature, humidity) of a district.

Requirements: BEC nodes SHOULD support ROLL [RFC] in order to join the LLN as a board router. Various wired/wireless communication protocols such as Radio Frequency (RF) protocols (e.g., Zigbee, WI-SUN) and Power Line Communication (PLC) should be supported. The BEC nodes can use 2G/3G/LTE/Ethernet to communicate with the Cloud.

3. Smart Manufacturing

Description: BEC nodes join the industrial manufacturing network and provide the networking function. Control messages that requires deterministic latency will be carried on this network. BEC nodes need to support deterministic networking protocols such IEEE Time Sensitive Networking (TSN), Profinet, Ethernet/IP, EtherCat, etc.

The gateway can also help monitor the equipment's status, and send out alarms or warnings when malfunction is detected or predicted.

Goal: Edge computing enables interconnection of deterministic networks.

Requirements: BEC nodes should support industrial machine-to-machine message bus connectivity protocols such as OPC-UA, DDS, MQTT. The network may be configured by a central controller using Netconf/YANG. BEC nodes should support the interconnection of heterogeneous deterministic Ethernet protocols.

4. Smart grid

Description: BEC nodes can be deployed in three scenarios of the smart grid. In advanced metering infrastructure (AMI), besides the routing function, a BEC node can also act as a concentrator to collect and aggregate the meters' data. It can also provide primary analysis to detect theft and outage. Firewall function can be deployed at the gateway to deal with attacks from the edge. In distribution automation (DA), BEC nodes provide bounded latency connection between controller and actuators such as switches and transformers. Edge computing applications can be implemented on these devices to monitor the status and react rapidly to faults. In terms of micro grid, the BEC node monitors the local power generation and storage, and helps smoothly integrate the energy generated by photovoltaic panels and wind turbines, whose power is very unstable, into the macro grid.

Goal: In AMI, the BEC node works as a router, firewall and concentrator, providing data preprocess services. In DA, BEC node enables the deterministic connection between controllers and actuators. In micro grid, BEC node is the coordinator between distributed and centralized generation.

Requirements: The gateway should support various wired/wireless communication protocols, such as Power Line Communication (PLC), Radio Frequency (RF), NB-IOT and 2G/3G/LTE. Bounded latency is required in automation use cases. Open platforms are needed to accommodate various applications providing data analysis, fault detection and automation control capabilities.

5. Gap Analysis

1. Multiple Virtualization Technologies Coexistence/Coordination:

Different virtualization technologies needed to meet the various vertical requirements. Coexistence and resource coordination is

needed. The focus is on the edge where different types of ASICs are found which require more input on selection.

2. Light weight Device-level management and virtual resource management:

Netconf/YANG to be considered as a baseline interface solution. Information and data modelling will need to be defined.

3. Framework and API for multi-ecosystem:

BEC framework for life cycle management. Unified API definition between framework and app including Local networking, Computing, Storage, OAM, UNI IO management and event-message management.

4. Runtime Updates

BEC nodes are commonly deployed to run for a long periods of time without downtime. Even during the update of configuration, software or firmware, the BEC nodes are required to serve the network with no interruption. For mesh-based networks, there might be some devices which are in sleeping mode. IP multicast protocols are not applicable here [draft-iab-iotsu-workshop-00]. Multicast Protocol for Low-Power and Lossy Networks (MPL) [RFC7731] should be used.

6. IANA Considerations

N/A

7. Security Considerations

Security considerations will be a critical component of IIoT edge computing particularly as intelligence is moved to the extreme edge.

8. Acknowledgement

The authors would like to thank Sami Kekki for his feedback on this draft.

9. Normative References

[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>>.

Authors' Addresses

Liang Geng
China Mobile

Email: gengliang@chinamobile.com

Mingui (Martin) Zhang
Huawei

Email: zhangmingui@huawei.com

Mike McBride
Huawei

Email: michael.mcbride@huawei.com

Bing Liu
Huawei

Email: remy.liubing@huawei.com

T2TRG
Internet-Draft
Intended status: Standards Track
Expires: September 6, 2018

L. Geng
China Mobile
M. Zhang
M. McBride
B. Liu
Huawei
March 5, 2018

Problem Statement of Edge Computing on Premises for Industrial IoT
draft-geng-iiot-edge-computing-problem-statement-01

Abstract

This document introduces the concept of Beyond Edge Computing (BEC) which brings edge computing capabilities down to the level of customers' premises for industrial IoT use cases. The purpose of the document is to discuss the general problem statement of BEC including capabilities, and use cases. Potential technical gaps in IETF problem scope that are related to BEC are also evaluated.

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 September 6, 2018.

Copyright Notice

Copyright (c) 2018 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	2
1.1. Requirements Language	3
1.2. Terminology	3
2. The Concept and Capabilities of Beyond Edge Computing	3
2.1. Heterogeneous IoT Device Compatibility	4
2.2. Deterministic Networking	5
2.3. Management of Massive Amount of Devices	5
2.4. Support of Network Slicing	5
2.5. Multi-ecosystem Environment	5
3. Reference Architecture	5
4. Use Cases of BEC	7
5. IANA Considerations	9
6. Security Considerations	10
7. Acknowledgement	10
8. Normative References	10
Authors' Addresses	10

1. Introduction

Edge computing is an important network architecture particularly in the support of Industrial verticals such as Energy, Manufacturing, Healthcare, Mining and Smart City/Buildings. Edge computing will provide local compute, storage and connectivity services particularly for latency and bandwidth sensitive applications. There are several organizations which are working on edge computing definition and architecture with various emphases. For instance, ETSI MEC (previously mobile edge computing and now multi-access edge computing) looks at edge computing from the perspective of the edge of the provider network. It also has an successive convention of focusing on cellular network requirements. The Industrial Internet Consortium (IIC) and Edge Computing Consortium (ECC) works on edge computing in a more general view of industrial IoT, where edge computing nodes are even closer to verticals (i.e. the very first hops where the service is connected to the network). Typically, the edge computing nodes are located at customers' premises. However, IIC and ECC are not standard organizations and they rely on communities such as IETF to provide protocols and API definitions for their architectural use especially as Operation Technology (OT), Information Technology (IT) and Communication Technology (CT) converge.

Edge computing concepts have been presented in various groups within the IETF/IRTF. The edge computing topic, similar to cloud computing, is much too broad to tackle within the IETF. There are specific protocol/interface areas, however, that can be worked on in the IETF once we identify a specific area of focus. BEC is one of the specific area which looks at edge computing from the industrial verticals such as factory, hospital, power and city/ building perspective and down to the level of local edge support for sensors, engines, pumps and machinery.

A simple example, of BEC, is factory equipment having connected sensors which are generating data and sending to the equipment within an edge computing environment. One sensor, connected to this equipment, could generate an event, such as overheating, and an connected actuator could quickly increase fan span or reduce engine speed depending upon the data within the local edge computing node. This type of event is being controlled today within closed industrial command and control protocols. But what is not generally available is the ability for open edge computing equipment to generate predictive maintenance and command and control across factories, verticals and into the cloud. This is where we see a gap in standards definitions and an opportunity for new protocols and interfaces, in which IETF could play a particularly important role.

1.1. Requirements Language

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

1.2. Terminology

- o BEC - Beyond Edge Computing, a concept of on-premises edge computing where the devices deployed directly at customers' premise are worked on.

2. The Concept and Capabilities of Beyond Edge Computing

Beyond Edge Computing (BEC) looks at the on-site intelligent evolution of industrial verticals. It brings the edge computing capability down to the level of customer premises, which are typical beyond the access network of a service provider.

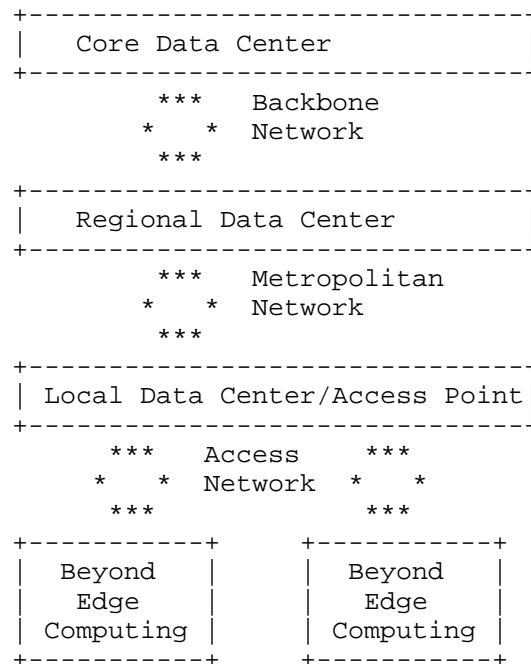


Figure 1: Beyond Edge Computing in the Network

Figure 1 illustrates the schematic diagram of BEC in terms of its position in a overall network. BEC takes care of the first hop where the service of a particular industrial vertical connects to the network. It can be regarded as an extended intelligent connectivity capability of a service provider's network to industrial verticals. Meanwhile, it also expands the cloud computing ability directly to customers' sites.

2.1. Heterogeneous IoT Device Compatibility

Vertical industries have various interfaces for on-premises LAN and WAN. This include both wireless and fixed line systems. Taking the field bus as an example, 10 different specification are defined in IEC61158, which is only a small portion of the existing field bus technologies people can actually find in real world. BEC should provide the capability of protocol translation and mapping, which enables the inter-operation between different systems.

2.2. Deterministic Networking

One of the most important motivation of BEC is to reduce the propagation latency by the deployment of services most closer to users. However, the processing and scheduling latency cannot directly benefit from closer deployment. Especially as the end-to-end propagation latency for a edge service has been reduced to less than 1 ms, processing and scheduling latency became even more essential. At the same time, emerging services including AR/VR, Remote robot system and motion control rely on not only low but also a strictly deterministic latency. Real-time operation system will be a preferred choice for BEC. Additionally, high-precision time synchronization and packet preemption are also significant characteristics for deterministic network in BEC system.

2.3. Management of Massive Amount of Devices

It is expected the hundreds of millions of BEC nodes will be deployed in industrial internet scenario, typically in a form of gateway. Management is the key to provide reliable and flexible edge services using BEC nodes. It is preferred to have unified interface to realize both device-level and resource-level management of BEC nodes.

2.4. Support of Network Slicing

An important benefit of BEC is the capability of edge service deployment. Based on light-weight distributed NFV technologies, the resource on BEC nodes can be dedicated for particular application. At the same time, the packet of a certain edge service can be labeled and steered to the preferred network slice at the WAN side, creating a true end-to-end network slice for industrial verticals.

2.5. Multi-ecosystem Environment

It is preferred to have a unified set of APIs, which achieve a deep-level capability exposure of the BEC nodes. The functions can be exposed to the upper layer applications in terms of forwarding, address, management and physical interface functionalities.

3. Reference Architecture

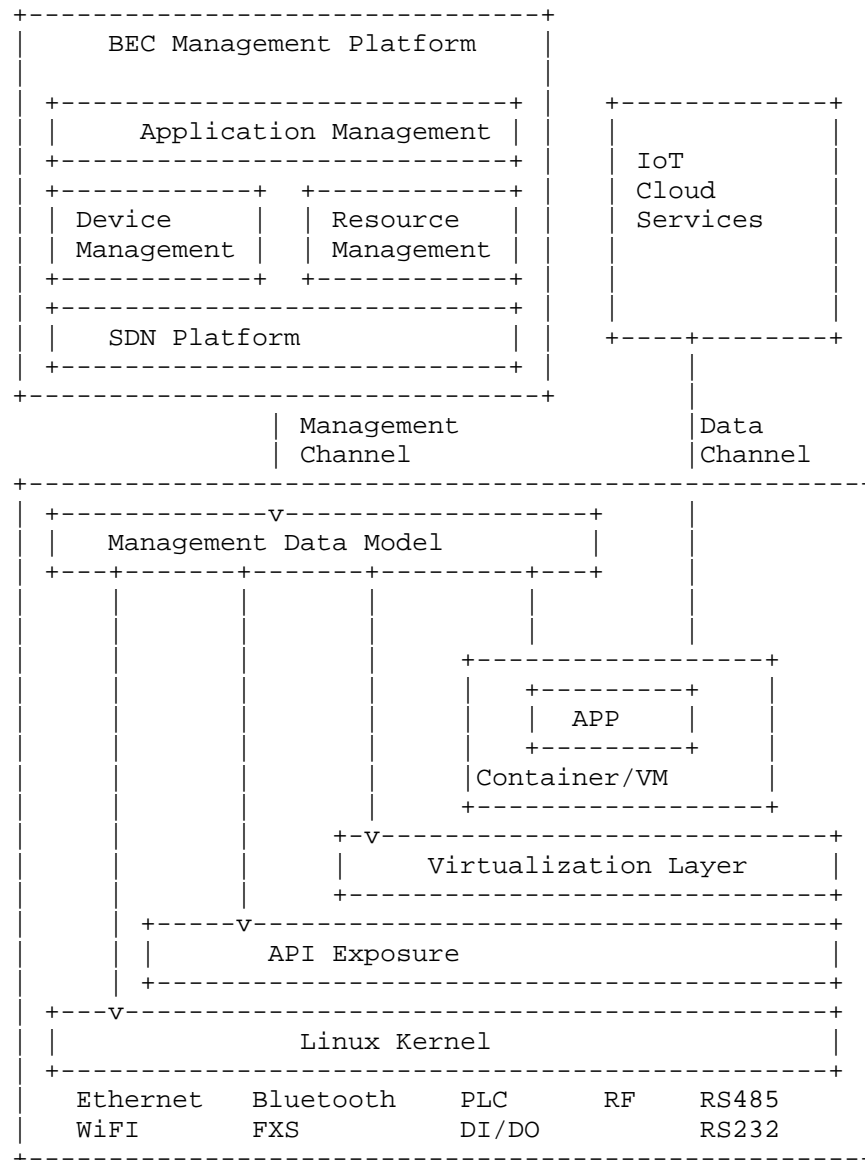


Figure 2: The Reference Architecture of Beyond Edge Computing

Figure 2 demonstrates the reference architecture of BEC system with a managed BEC node and a cloud-based management platform. An IoT

gateway is the typical form of a BEC node device. Gateways always play important role in the Cloud-Edge architecture since they are the most popular devices where verticals are provided with various capabilities such as computing, storage and networking. In addition, applications for various vertical customers are developed by themselves or third-party while deployed on demand. Giving the edge computing ability of BEC, much of the data can be processed by applications running on the gateway locally as required by vertical customers. The gateways are commonly versatile protocol speakers so that devices speaking different protocols can communicate with the them. The East-West connectivity between BEC nodes might be enabled by various protocols such as OPC-UA, MQTT, TSN and other deterministic Ethernet protocols, for example EtherCat, Ethernet/IP, Profinet. To facilitate the operation of the BEC system, a central controller in the cloud is provisioned to the customer. It mainly supervise the device, virtualization resource and application life cycle of the BEC node.

The key requirements of BEC are in providing distribution service entities on the customers site (end AP, devices) to meet the growing demand for low latency, reliable, and secure vertical industries. The Computing, Storage, I/O isolation are remotely managed at the edge to provide certain dedication and quality guarantees. Agile, flexible and scalable deployment of services from operator/third party down to the edge through software entities (VM/ Containers). A light weight MANO like approach is needed to provide resource provisioning and VNF deployment. A unified API definition is needed to support the co- existence of multi-ecosystem at the BEC node. And there needs to be the ability for the edge device to map specific service requirements with an end to end network slice with certain guarantees and pass the policy identification along the path to the centralized DC.

4. Use Cases of BEC

1. Elevator Networks

Description: There are more than 15 million elevators around the world and the daily maintenance of these elevators costs elevator operators a large amount of revenue. An elevator usually carries hundreds of sensors which are generating a large amount of data to be uploaded to the cloud. The BEC nodes can preprocess the data gathered from elevator sensors so that the volume to be uploaded to the Cloud is greatly reduced. Based on the input from elevator sensors, AI programs installed on BEC nodes may locally make decisions without the intervention of the Cloud. For example, when the noise or vibration of an elevator exceeds a certain level, the

BEC node may notify elevator maintainers to reach this elevator and perform maintenance or repair.

Goal: BEC nodes are deployed into elevators to gather/preprocess/compress the data to save the communication cost. Based on the data gathered from elevator sensors, BEC nodes can assist operators to do predictive maintenance.

Requirements: Customized gateways operated by elevator providers. An open platform with VMs/containers which can hold customized Apps. These Apps are managed by elevator operators while developed by gateway vendors or any other third parties. Various connectivities are supported (2G/3G/LTE/eMTC/Ethernet) by BEC nodes. A central controller to perform configuration and management of the network. AI models are trained on the Cloud while the reasoning of these AI models are performed at the Edge.

2. Intelligent Street Lights

Description: BEC nodes are placed on street lights to act as board routers of LLNs. BEC nodes may act as RSUs of vehicle networks. With AI programs installed on the BEC nodes, reasoning and decisions might be made locally at the edge. For example, BEC nodes can adjust the lights' brightness and operating hours according to environment parameters, providing illumination when needed while reducing power consumption. With sensors on trash cans, BEC nodes are aware whether a trash can is full. Trash collecting cars can communicate with the BEC nodes to determine whether to reach a trash can to collect the trash.

Goal: BEC nodes gather data from sensors which are used to monitor various information (e.g., brightness, temperature, humidity) of a district.

Requirements: BEC nodes SHOULD support ROLL [RFC] in order to join the LLN as a board router. Various wired/wireless communication protocols such as Radio Frequency (RF) protocols (e.g., Zigbee, WI-SUN) and Power Line Communication (PLC) should be supported. The BEC nodes can use 2G/3G/LTE/Ethernet to communicate with the Cloud.

3. Smart Manufacturing

Description: BEC nodes join the industrial manufacturing network and provide the networking function. Control messages that requires deterministic latency will be carried on this network. BEC nodes need to support deterministic networking protocols such IEEE Time Sensitive Networking (TSN), Profinet, Ethernet/IP, EtherCat, etc.

The gateway can also help monitor the equipments' status, and send out alarms or warnings when malfunction is detected or predicted.

Goal: Edge computing enables interconnection of deterministic networks.

Requirements: BEC nodes should support industrial machine-to-machine message bus connectivity protocols such as OPC-UA, DDS, MQTT. The network may be configured by a central controller using Netconf/YANG. BEC nodes should support the interconnection of heterogeneous deterministic Ethernet protocols.

4. Smart grid

Description: BEC nodes can be deployed in three scenarios of the smart grid. In advanced metering infrastructure (AMI), besides the routing function, it can also act as a concentrator to collect and aggregate the meters' data. It can also provide primary analysis to detect theft and outage. Firewall function can be deployed at the gateway to deal with attacks from the edge. In distribution automation (DA), BEC nodes provide bounded latency connection between controller and actuators such as switches and transformers. Edge computing applications can be implemented on these devices to monitor the status and react rapidly to faults. In terms of microgrid, the BEC node monitors the local power generation and storage, and helps smoothly integrate the energy generated by photovoltaic panels and wind turbines, whose power is very unstable, into the macro grid.

Goal: In AMI, the BEC node works as a router, firewall and concentrator, providing data preprocess services. In DA, BEC node enables the deterministic connection between controllers and actuators. In microgrid, BEC node is the coordinator between distributed and centralized generation.

Requirements: The gateway should support various wired/wireless communication protocols, such as Power Line Communication (PLC), Radio Frequency (RF), NB-IOT and 2G/3G/LTE. Bounded latency is required in automation use cases. Open platforms are needed to accommodate various applicaitons providing data analysis, fault detection and automation control capabilities.

5. IANA Considerations

N/A

6. Security Considerations

Security considerations will be a critical component of IIoT edge computing particularly as intelligence is moved to the extreme edge.

7. Acknowledgement

The authors would like to thank Sami Kekki for his feedback on this draft.

8. Normative References

[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>>.

Authors' Addresses

Liang Geng
China Mobile

Email: gengliang@chinamobile.com

Mingui (Martin) Zhang
Huawei

Email: zhangmingui@huawei.com

Mike McBride
Huawei

Email: michael.mcbride@huawei.com

Bing Liu
Huawei

Email: remy.liubing@huawei.com

Network Working Group
Internet-Draft
Intended status: Informational
Expires: May 2, 2018

O. Garcia-Morchon
Philips IP&S
S. Kumar
Philips Research
M. Sethi
Ericsson
October 29, 2017

State-of-the-Art and Challenges for the Internet of Things Security
draft-irtf-t2trg-iot-secons-08

Abstract

The Internet of Things (IoT) concept refers to the usage of standard Internet protocols to allow for human-to-thing and thing-to-thing communication. The security needs for the IoT are well-recognized and many standardization steps for providing security have been taken, for example, the specification of Constrained Application Protocol (CoAP) over Datagram Transport Layer Security (DTLS). However, security challenges still exist and there are some use cases that lack a suitable solution. In this document, we first discuss the various stages in the lifecycle of a thing. Next, we document the various security threats to a thing and the challenges that one might face to protect against these threats. Lastly, we discuss the next steps needed to facilitate the deployment of secure IoT systems. This document can be used by IoT standards specifications as a reference for details about security considerations applying to the specified protocol.

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 <http://datatracker.ietf.org/drafts/current/>.

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

This Internet-Draft will expire on May 2, 2018.

Copyright Notice

Copyright (c) 2017 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction	3
2. Motivation and background	4
2.1. The Thing Lifecycle	4
2.2. Security building blocks	6
3. Security Threats and Managing Risk	9
4. State-of-the-Art	13
4.1. IP-based IoT Protocols and Standards	14
4.2. Existing IP-based Security Protocols and Solutions	16
4.3. IoT Security Guidelines	19
5. Challenges for a Secure IoT	22
5.1. Constraints and Heterogeneous Communication	22
5.1.1. Resource Constraints	23
5.1.2. Denial-of-Service Resistance	24
5.1.3. End-to-end security, protocol translation, and the role of middleboxes	24
5.1.4. New network architectures and paradigm	26
5.2. Bootstrapping of a Security Domain	27
5.3. Operational Challenges	27
5.3.1. Group Membership and Security	27
5.3.2. Mobility and IP Network Dynamics	28
5.4. Secure software update	29
5.5. End-of-Life	31
5.6. Verifying device behavior	31
5.7. Testing: bug hunting and vulnerabilities	32
5.8. Quantum-resistance	32
5.9. Privacy protection	33
5.10. Data leakage	34
5.11. Trustworthy IoT Operation	35
6. Conclusions and Next Steps	35

7. Security Considerations	36
8. IANA Considerations	36
9. Acknowledgments	36
10. Informative References	36
Authors' Addresses	47

1. Introduction

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

The things that are part of the Internet of Things are no longer unresponsive and have transformed into computing devices that understand and react to the environment they reside in. These things are also often referred to as smart objects or smart devices. The introduction of IPv6 [RFC6568] and CoAP [RFC7252] as fundamental building blocks for IoT applications allows connecting IoT hosts to the Internet. This brings several advantages including: (i) a homogeneous protocol ecosystem that allows simple integration with other Internet hosts; (ii) simplified development for devices that significantly vary in their capabilities; (iii) a unified interface for applications, removing the need for application-level proxies. These building blocks greatly simplify the deployment of the envisioned scenarios which range from building automation to production environments and personal area networks.

This document presents an overview of important security aspects for the Internet of Things. We begin by discussing the lifecycle of a thing and giving general definitions of the security building blocks in Section 2. In Section 3, we discuss security threats for the IoT and methodologies for managing these threats when designing a secure system. Section 4 reviews existing IP-based (security) protocols for the IoT and briefly summarizes existing guidelines and regulations. Section 5 identifies remaining challenges for a secure IoT and discusses potential solutions. Section 6 includes final remarks and

conclusions. This document can be used by IoT standards specifications as a reference for details about security considerations applying to the specified system or protocol.

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

2. Motivation and background

This section begins by describing the lifecycle of a thing. It then details the five different security building blocks that can be used for analyzing and classifying the security aspects of the IoT.

2.1. The Thing Lifecycle

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

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

specific task. It is therefore unlikely that one single manufacturer will create all nodes in a building. Hence, interoperability as well as trust bootstrapping between nodes of different vendors is important.

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

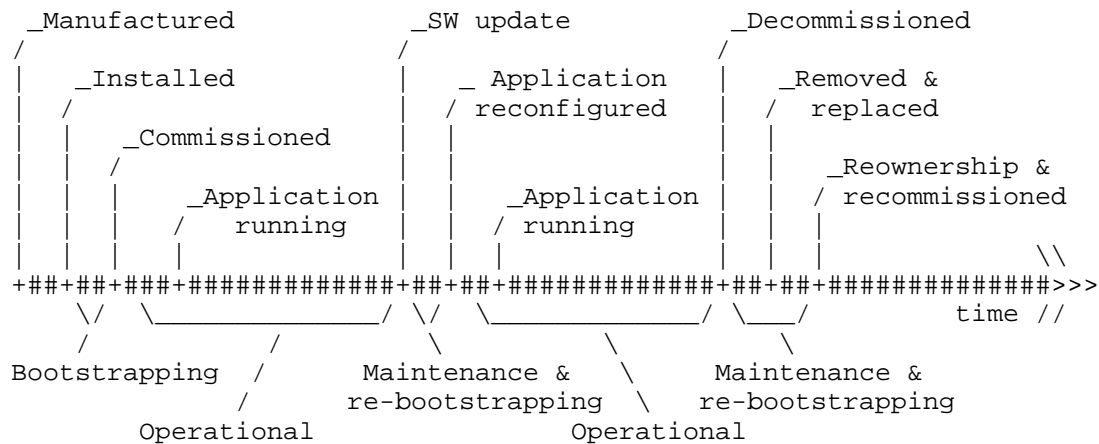


Figure 1: The lifecycle of a thing in the Internet of Things

2.2. Security building blocks

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

The term security subsumes a wide range of primitives, protocols, and procedures. Firstly, it includes the basic provision of security services that include confidentiality, authentication, integrity, authorization, non-repudiation, and availability along with some augmented services, such as duplicate detection and detection of stale packets (timeliness). These security services can be

implemented by means of a combination of cryptographic mechanisms, such as block ciphers, hash functions, or signature algorithms, and non-cryptographic mechanisms, which implement authorization and other security policy enforcement aspects. For ensuring security in IoT networks, we should not only focus on the required security services, but also pay special attention to how these services are realized in the overall system and how the security functionalities are executed in practice. To this end, we consider five major "building blocks" to analyze and classify security aspects for IoT:

1. IoT security architecture: refers to the system-level elements involved in the management of security relationships between things (for example, centralized or distributed). For instance, a smart home could rely on a centralized key distribution center in charge of managing cryptographic keys, devices, users, access control and privacy policies.
2. The security model within a thing: describes the way security parameters, keys, processes, and applications are managed within a smart object. This includes aspects such as application process separation, secure storage of key materials, etc. For instance, some smart objects might have extremely limited resources and limited capabilities to protect secret keys. In contrast, other devices used in critical applications, such as a pacemaker, may rely on methods to protect cryptographic keys and functionality.
3. Secure bootstrapping: denotes the process by which a thing securely joins an IoT system at a given location and point of time. For instance, bootstrapping of a connected camera can include the authentication and authorization of the device as well as the transfer of security parameters necessary for operation in a given network.
4. Network security: describes the mechanisms applied within a network to ensure secure operation. Specifically, it prevents attackers from endangering or modifying the expected operation of a smart object. It also protects the network itself from malicious things. Network security can include several mechanisms ranging from data link layer security, secure routing, and network layer security.
5. Application security: describes mechanisms to allow secure transfer of application data. The security may be implemented at different layers of the Internet protocol suite. For instance, assume a smart object such as an environmental sensor that is connected to a backend system. Application security here can refer to the exchange of secure blocks of data such as

measurements between the sensor and the backed, or it can also refer to a software update for the smart object. This data is exchanged end-to-end independently of the underlying network infrastructure, for example through proxies or other store-and-forward mechanisms.

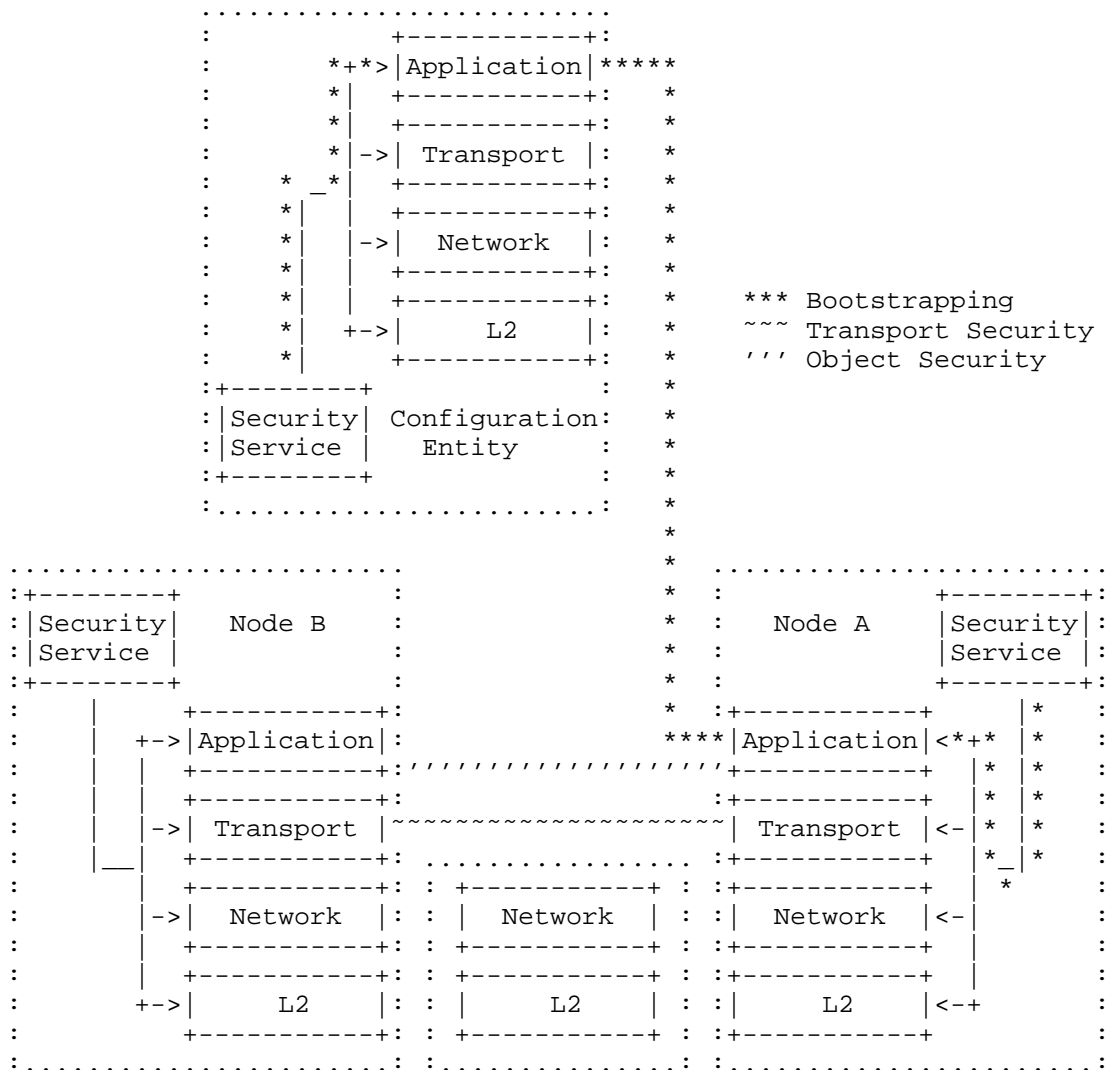


Figure 2: Overview of Security Mechanisms

Inspired by the security framework for routing over low power and lossy network [RFC7416], we show an example security model of a smart

object and illustrate how different security concepts and lifecycle phases map to the Internet communication stack. Assume a centralized architecture in which a configuration entity stores and manages the identities of the things associated with the system along with their cryptographic keys. During the bootstrapping phase, each thing executes the bootstrapping protocol with the configuration entity, thus obtaining the required device identities and the keying material. The security service on a thing in turn stores the received keying material for the network layer and application security mechanisms for secure communication. Things can then securely communicate with each other during their operational phase by means of the employed network and application security mechanisms.

3. Security Threats and Managing Risk

Security threats in related IP protocols have been analyzed in multiple documents including HTTPS [RFC2818], COAP [RFC7252], 6LoWPAN [RFC4919], ANCP [RFC5713], DNS security threats [RFC3833], IPv6 ND [RFC3756], and PANA [RFC4016]. In this section, we specifically discuss the threats that could compromise an individual thing, or the network as a whole. Note that these set of threats might go beyond the scope of Internet protocols but we gather them here for the sake of completeness. We also note that these threats can be classified according to either (i) the thing's lifecycle phases (when does the threat occur?) or (ii) the security building blocks (which functionality is affected by the threat?). All these threats are summarized in Figure 3.

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

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

3. Eavesdropping attack: During the commissioning of a thing into a network, it may be susceptible to eavesdropping, especially if operational keying materials, security parameters, or configuration settings, are exchanged in clear using a wireless medium or if used cryptographic algorithms are not suitable for the envisioned lifetime of the device and the system. After obtaining the keying material, the attacker might be able to recover the secret keys established between the communicating entities, thereby compromising the authenticity and confidentiality of the communication channel, as well as the authenticity of commands and other traffic exchanged over this communication channel. When the network is in operation, T2T communication may be eavesdropped upon if the communication channel is not sufficiently protected or in the event of session key compromise due to protocol weaknesses or a long period of usage without key renewal or updates.
4. Man-in-the-middle attack: Both the commissioning phase and operational phases may also be vulnerable to man-in-the-middle attacks, for example, when keying material between communicating entities is exchanged in the clear and the security of the key establishment protocol depends on the tacit assumption that no third party can eavesdrop during the execution of this protocol. Additionally, device authentication or device authorization may be non-trivial, or may need support of a human decision process, since things usually do not have a-priori knowledge about each other and cannot always differentiate friends and foes via completely automated mechanisms. Thus, even if the key establishment protocol provides cryptographic device authentication, this knowledge on device identities may still need complementing with a human-assisted authorization step (thereby, presenting a weak link and offering the potential of man-in-the-middle attacks this way).
5. Firmware attacks: When a thing is in operation or maintenance phase, its firmware or software may be updated to allow for new functionality or new features. An attacker may be able to exploit such a firmware upgrade by replacing the thing's software with malicious software, thereby influencing the operational behavior of the thing. For example, an attacker

could add a piece of malicious code to the firmware that will cause it to periodically report the energy usage of the lamp to a data repository for analysis. Similarly, devices whose software has not been properly maintained and updated might contain vulnerabilities that might be exploited by attackers to replace the firmware on the device.

6. Extraction of private information: IoT devices (such as sensors, actuators, etc.) are often physically unprotected in their ambient environment and they could easily be captured by an attacker. An attacker with physical access may then attempt to extract private information such as keys (for example, device's key, private-key, group key), sensed data (for example, healthcare status of a user), configuration parameters (for example, the Wi-Fi key), or proprietary algorithms (for example, algorithm performing some data analytics task). Even when the data originating from a thing is encrypted, attackers can perform traffic analysis to deduce meaningful information which might compromise the privacy of the thing's owner and/or user.
7. Routing attack: As highlighted in [ID-Daniel], routing information in IoT can be spoofed, altered, or replayed, in order to create routing loops, attract/repel network traffic, extend/shorten source routes, etc. Other relevant routing attacks include 1) Sinkhole attack (or blackhole attack), where an attacker declares himself to have a high-quality route/path to the base station, thus allowing him to do manipulate all packets passing through it. 2) Selective forwarding, where an attacker may selectively forward packets or simply drop a packet. 3) Wormhole attack, where an attacker may record packets at one location in the network and tunnel them to another location, thereby influencing perceived network behavior and potentially distorting statistics, thus greatly impacting the functionality of routing. 4) Sybil attack, whereby an attacker presents multiple identities to other things in the network.
8. Elevation of privilege: An attacker with low privileges can misuse additional flaws in the implemented authentication and authorization mechanisms of a thing to gain more privileged access to the thing and its data.
9. Privacy threat: The tracking of a thing's location and usage may pose a privacy risk to its users. For instance, an attacker can infer information based on the information gathered about individual things, thus deducing behavioral patterns of the user of interest to him. Such information may subsequently be sold to interested parties for marketing purposes and targeted advertising. In extreme cases, such information might be used

to track dissidents in oppressive regimes. Unlawful surveillance and interception of traffic to/from a thing by intelligence agencies is also a privacy threat.

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

The following table summarizes the above generic security threats and the potential point of vulnerabilities at different layers of the communication stack. We also include related documents that include a threat model that might apply to the IoT.

	Manufacturing	Installation/ Commissioning	Operation
System-level	Device Cloning	Substitution	Privacy threat Extraction of private inform.
Application Layer		RFC2818, RFC4016	RFC2818, Firmware replacement
Transport Layer		Eavesdropping & Man-in-the-middle attack,	Eavesdropping Man-in-the-middle
Network Layer		RFC4919, RFC5713 RFC3833, RFC3756	RFC4919, DoS attack Routing attack RFC3833
Physical Layer			DoS attack

Figure 3: Classification of threats according to the lifecycle phases

Dealing with above threats and finding suitable security mitigations is challenging. New threats and exploits also appear on a daily basis. Therefore, the existence of proper secure product creation processes that allow managing and minimizing risks during the lifecycle of IoT devices is at least as important as being aware of the threats. A non-exhaustive list of relevant processes include:

1. A Business Impact Analysis (BIA) assesses the consequences of the loss of basic security attributes: confidentiality, integrity and availability in an IoT system. These consequences might include the impact from lost data, reduced sales, increased expenses, regulatory fines, customer dissatisfaction, etc. Performing a business impact analysis allows a business to determine the relevance of having a proper security design.
2. A Risk Assessment (RA) analyzes security threats to an IoT system while considering their likelihood and impact. It also includes categorizing each of them with a risk level. Risks classified as moderate or high must be mitigated, i.e., the security architecture should be able to deal with those threat.
3. A privacy impact assessment (PIA) aims at assessing the Personally Identifiable Information (PII) that is collected, processed, or used in an IoT system. By doing so, the goal is to fulfill applicable legal requirements, determine risks and effects of manipulation and loss of PII.
4. Procedures for incident reporting and mitigation refer to the methodologies that allow becoming aware of any security issues that affect an IoT system. Furthermore, this includes steps towards the actual deployment of patches that mitigate the identified vulnerabilities.

BIA, RA, and PIA should generally be realized during the creation of a new IoT system or when deploying significant system/feature upgrades. In general, it is recommended to re-assess them on a regular basis taking into account new use cases and/or threats.

4. State-of-the-Art

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

4.1. IP-based IoT Protocols and Standards

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

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

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

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

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

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

direction. It also supports point-to-point traffic between the resource-constrained devices. A set of routing metrics and constraints for path calculation in RPL are also specified [RFC6551].

The IPv6 over Networks of Resource-constrained Nodes (6lo) [WG-6lo] working group of the IETF has specified how IPv6 packets can be transmitted over various link layer protocols that are commonly employed for resource-constrained smart object networks. There is also ongoing work to specify IPv6 connectivity for a Non-Broadcast Multi-Access (NBMA) mesh network that is formed by IEEE 802.15.4 TimeSlotted Channel Hopping (TSCH) links [ID-6tisch]. Other link layer protocols for which IETF has specified or is currently specifying IPv6 support include Bluetooth [RFC7668], Digital Enhanced Cordless Telecommunications (DECT) Ultra Low Energy (ULE) air interface [RFC8105], and Near Field Communication (NFC) [ID-6lonfc].

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

JavaScript Object Notation (JSON) is a lightweight text representation format for structured data [RFC7159]. It is often used for transmitting serialized structured data over the network. IETF has defined specifications for encoding public keys, signed content, and claims to be transferred between two parties as JSON objects. They are referred to as JSON Web Keys (JWK) [RFC7517], JSON Web Signatures (JWS) [RFC7515] and JSON Web Token (JWT) [RFC7519].

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

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

The Thing-2-Thing Research Group (T2TRG) [RG-T2TRG] is investigating the remaining research issues that need to be addressed to quickly turn the vision of IoT into a reality where resource-constrained

nodes can communicate with each other and with other more capable nodes on the Internet.

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

1. Thread [Thread]: Specifies the Thread protocol that is intended for a variety of IoT devices. It is an IPv6-based network protocol that runs over IEEE 802.15.4.
2. Industrial Internet Consortium [IIoT]: The consortium defines reference architectures and security frameworks for development, adoption and widespread use of Industrial Internet technologies based on existing IETF standards.
3. Internet Protocol for Smart Objects IPSO [IPSO]: The alliance specifies a common object model that enables application software on any device to interoperate with other conforming devices.
4. OneM2M [OneM2M]: The standards body defines technical and API specifications for IoT devices. It aims to create a service layer that can run on any IoT device hardware and software.
5. Open Connectivity Foundation (OCF) [OCF]: The foundation develops standards and certifications primarily for IoT devices that use Constrained Application Protocol (CoAP) as the application layer protocol.
6. Fairhair Alliance [Fairhair]: Specifies an IoT middleware to enable interoperability between different application standards used in building automation and lighting systems.
7. OMA LWM2M [LWM2M]: OMA Lightweight M2M is a standard from the Open Mobile Alliance for M2M and IoT device management. LWM2M relies on CoAP as the application layer protocol and uses a RESTful architecture for remote management of IoT devices.

4.2. Existing IP-based Security Protocols and Solutions

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

In the context of the IP-based IoT deployments, consideration of existing Internet security protocols is important. There are a wide

range of specialized as well as general-purpose security solutions for the Internet domain such as IKEv2/IPsec [RFC7296], TLS [RFC5246], DTLS [RFC6347], HIP [RFC7401], PANA [RFC5191], and EAP [RFC3748].

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

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

Transport Layer Security (TLS) and its datagram-oriented variant DTLS secure transport-layer connections. TLS provides security for TCP and requires a reliable transport, while DTLS secures and uses datagram-oriented protocols such as UDP. Both protocols are intentionally kept similar and share the same ideology and cipher suites.

OSCOAP [ID-OSCOAP] is a proposal that protects CoAP messages by wrapping them in the CBOR Object Signing and Encryption (COSE) [RFC8152] format. Thus, OSCOAP falls in the category of object security and it can be applied wherever CoAP can. The advantage of OSCOAP compared with DTLS resides in some more flexibility when dealing with e2e security as it will be discussed in Section 5.1.3.

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

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

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

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

Figure 4 depicts the relationships between the discussed protocols in the context of the security terminology introduced in Section 2.

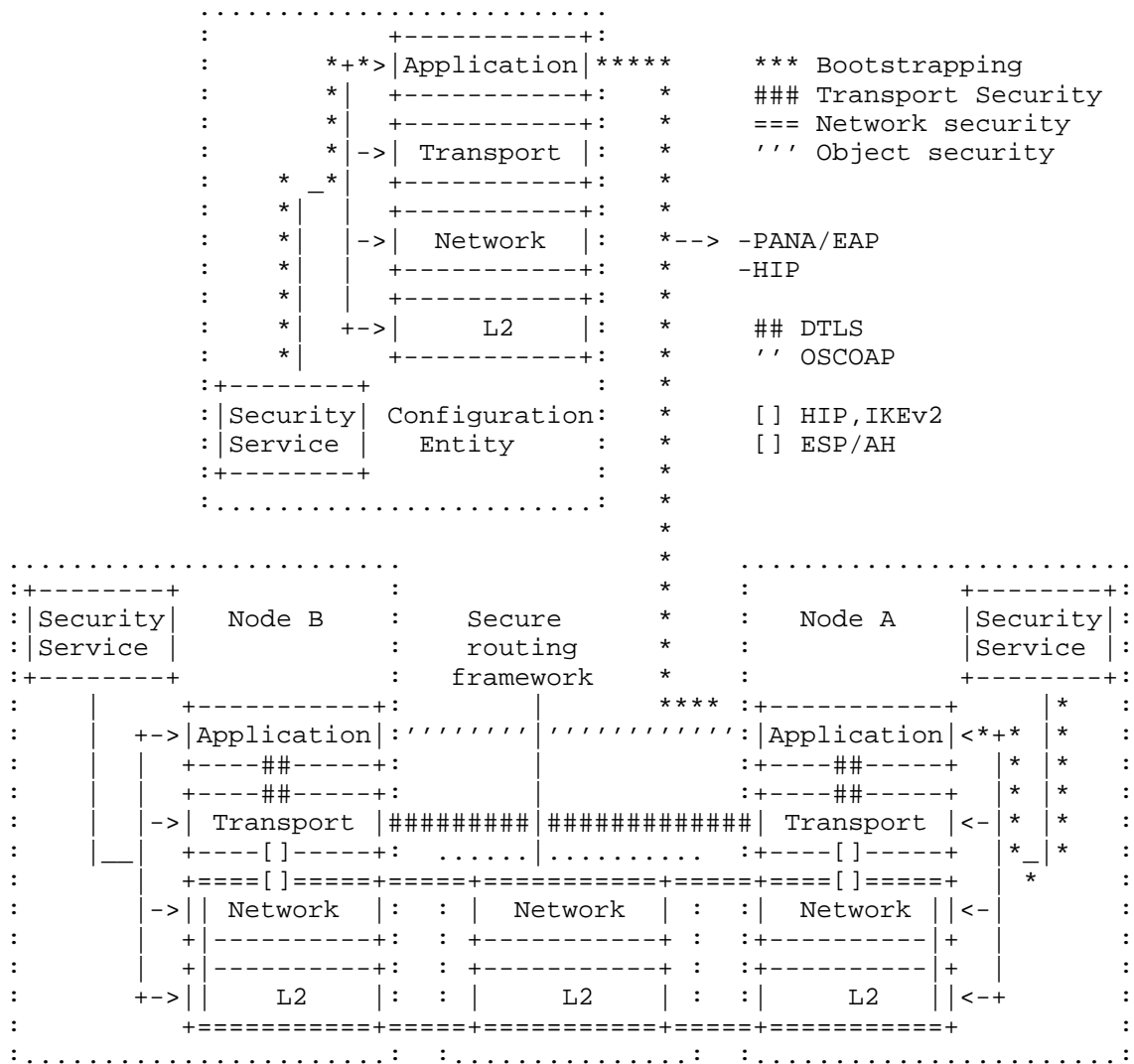


Figure 4: Relationships between IP-based security protocols

4.3. IoT Security Guidelines

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

build secure IoT devices and services. A subset of the different guidelines and ongoing projects are as follows:

1. GSMA IoT security guidelines [GSMAsecurity]: GSMA has published a set of security guidelines for the benefit of new IoT product and service providers. The guidelines are aimed at device manufacturers, service providers, developers and network operators. An enterprise can complete an IoT Security Self-Assessment to demonstrate that its products and services are aligned with the security guidelines of the GSMA.
2. BITAG Internet of Things (IoT) Security and Privacy Recommendations [BITAG]: Broadband Internet Technical Advisory Group (BITAG) has also published recommendations for ensuring security and privacy of IoT device users. BITAG observes that many IoT devices are shipped from the factory with software that is already outdated and vulnerable. The report also states that many devices with vulnerabilities will not be fixed either because the manufacturer does not provide updates or because the user does not apply them. The recommendations include that IoT devices should function without cloud and Internet connectivity, and that all IoT devices should have methods for automatic secure software updates.
3. CSA New Security Guidance for Early Adopters of the IoT [CSA]: The Cloud Security Alliance (CSA) recommendations for early adopters of IoT encourages enterprises to implement security at different layers of the protocol stack. It also recommends implementation of an authentication/authorization framework for IoT deployments. A complete list of recommendations is available in the report [CSA].
4. U.S. Department of Homeland Security [DHS]: DHS has put forth six strategic principles that would enable IoT developers, manufacturers, service providers and consumers to maintain security as they develop, manufacture, implement or use network-connected IoT devices.
5. NIST [NIST-Guide]: The NIST special publication urges enterprise and US federal agencies to address security throughout the systems engineering process. The publication builds upon the ISO/IEC 15288 standard and augments each process in the system lifecycle with security enhancements.
6. NIST [nist_lightweight_project]: NIST is running a project on lightweight cryptography with the purpose of: (i) identifying application areas for which standard cryptographic algorithms are too heavy, classifying them according to some application

profiles to be determined; (ii) determining limitations in those existing cryptographic standards; and (iii) standardizing lightweight algorithms that can be used in specific application profiles.

7. OWASP [OWASP]: Open Web Application Security Project (OWASP) provides security guidance for IoT manufactures, developers and consumers. OWASP also includes guidelines for those who intend to test and analyze IoT devices and applications.
8. IoT Security foundation [IoTSecFoundation]: IoT security foundation has published a document that enlists various considerations that need to be taken into account when developing IoT applications. For example, the document states that IoT devices could use hardware-root of trust to ensure that only authorized software runs on the devices.
9. NHTSA [NHTSA]: The US National Highway Traffic Safety Administration provides a set of non-binding guidance to the automotive industry for improving the cyber security of vehicles. While some of the guidelines are general, the document provides specific recommendations for the automotive industry such as how various automotive manufacturer can share cyber security vulnerabilities discovered.
10. Best Current Practices (BCP) for IoT devices [ID-Moore]: This document provides a list of minimum requirements that vendors of Internet of Things (IoT) devices should take into account while developing applications, services and firmware updates in order to reduce the frequency and severity of security incidents that arise from compromised IoT devices.
11. ENISA [ENISA_ICS]: The European Union Agency for Network and Information Security published a document on communication network dependencies for ICS/SCADA systems in which security vulnerabilities, guidelines and general recommendations are summarized.

Other guideline and recommendation documents may exist or may later be published. This list should be considered non-exhaustive. Despite the acknowledgment that security in the Internet is needed and the existence of multiple guidelines, the fact is that many IoT devices and systems have very limited security. There are multiple reasons for this. For instance, some manufactures focus on delivering a product without paying enough attention to security. This may be because of lack of expertise or limited budget. However, the deployment of such insecure devices poses a severe threat on the privacy and safety of users. The vast amount of devices and their

inherent mobile nature also implies that an initially secure system can become insecure if a compromised device gains access to the system at some point in time. Even if all other devices in a given environment are secure, this does not prevent external (passive) attacks caused by insecure devices.

Recently the Federal Communications Commission (FCC) [FCC] has stated the need for additional regulation of IoT systems. FCC identifies this as a missing component, especially for Federal Information Systems (FIS). Today, security in the US FIS is regulated according to Federal Information Security Management Act (FISMA). From this law, NIST has derived a number of new documents to categorize FIS and determine minimum security requirements for each category. These minimum security requirements are specified in NIST SP 800-53r4 [NIST-SP80053].

Even with strong regulations in place, the question remains as to how such regulations can be applied in practice to non-federal deployments, such as industrial, homes, offices, or smart cities. Each of them exhibits unique features, involves very diverse types of users, has different operational requirements, and combines IoT devices from multiple manufacturers. Future regulations should therefore consider such diverse deployment scenarios.

5. Challenges for a Secure IoT

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

5.1. Constraints and Heterogeneous Communication

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

5.1.1.1. Resource Constraints

IoT deployments are often characterized by lossy and low-bandwidth communication channels. IoT devices are also often constrained in terms of CPU, memory, and energy budget available [RFC7228]. These characteristics directly impact the threats to and the design of security protocols for the IoT domain. First, the use of small packets, for example, IEEE 802.15.4 supports 127-byte sized packets at the physical layer, may result in fragmentation of larger packets required by security protocols. This may open new attack vectors for state exhaustion DoS attacks, which is especially tragic, for example, if the fragmentation is caused by large key exchange messages of security protocols. Moreover, packet fragmentation commonly downgrades the overall system performance due to fragment losses and the need for retransmissions. For instance, fate-sharing packet flight as implemented by DTLS might aggravate the resulting performance loss.

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

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

Independently from the development in the IoT domain, all discussed security protocols show efforts to reduce the cryptographic cost of the required public-key-based key exchanges and signatures with Elliptic Curve Cryptography (ECC) [RFC5246], [RFC5903], [RFC7401], and [ID-HIP-DEX]. Moreover, all protocols have been revised in the last years to enable crypto agility, making cryptographic primitives interchangeable. However, these improvements are only a first step in reducing the computation and communication overhead of Internet protocols. The question remains if other approaches can be applied to leverage key agreement in these heavily resource-constrained environments.

A further fundamental need refers to the limited energy budget available to IoT nodes. Careful protocol (re)design and usage is

required to reduce not only the energy consumption during normal operation, but also under DoS attacks. Since the energy consumption of IoT devices differs from other device classes, judgments on the energy consumption of a particular protocol cannot be made without tailor-made IoT implementations.

5.1.2. Denial-of-Service Resistance

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

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

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

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

a) the relevant information is encrypted and inaccessible to the gateway or b) rewriting integrity-protected parts of the packet would invalidate the end-to-end integrity protection.

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

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

1. Sharing credentials with middleboxes enables them to transform (for example, decompress, convert, etc.) packets and re-apply the security measures after transformation. This method abandons end-to-end security and is only applicable to simple scenarios with a rudimentary security model.
2. Reusing the Internet wire format for IoT makes conversion between IoT and Internet protocols unnecessary. However, it can lead to poor performance in some use cases because IoT specific optimizations (for example, stateful or stateless compression) are not possible.
3. Selectively protecting vital and immutable packet parts with a message authentication code or with encryption requires a careful balance between performance and security. Otherwise this approach might either result in poor performance or poor security depending on which parts are selected for protection, where they are located in the original packet, and how they are processed. [ID-OSCOAP] proposes a solution in this direction by encrypting and integrity protecting most of the message fields except those parts that a middlebox needs to read or change.
4. Homomorphic encryption techniques can be used in the middlebox to perform certain operations. However, this is limited to data processing involving arithmetic operations. Furthermore, performance of existing libraries, for example, SEAL [SEAL] is still too limited and it is not widely applicable yet.

5. Message authentication codes that sustain transformation can be realized by considering the order of transformation and protection (for example, by creating a signature before compression so that the gateway can decompress the packet without recalculating the signature). Such an approach enables IoT specific optimizations but is more complex and may require application-specific transformations before security is applied. Moreover, the usage of encrypted or integrity-protected data prevents middleboxes from transforming packets.
6. Mechanisms based on object security can bridge the protocol worlds, but still require that the two worlds use the same object security formats. Currently the object security format based on CBOR Object Signing and Encryption (COSE) [RFC8152] (IoT protocol) is different from JSON Object Signing and Encryption (JOSE) [RFC7520] or Cryptographic Message Syntax (CMS) [RFC5652]. Legacy devices relying on traditional Internet protocols will need to update to the newer protocols for constrained environments to enable real end-to-end security. Furthermore, middleboxes do not have any access to the data and this approach does not prevent an attacker from modifying relevant fields in CoAP.

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

We finally note that end-to-end security can also be considered in the context of availability: making sure that the messages are delivered. In this case, the end-points cannot control this, but the middleboxes play a fundamental role to make sure that exchanged messages are not dropped, for example, due to a DDoS attack.

5.1.4. New network architectures and paradigm

There is a multitude of new link layer protocols that aim to address the resource-constrained nature of IoT devices. For example, the IEEE 802.11 ah [IEEE802ah] has been specified for extended range and lower energy consumption to support Internet of Things (IoT) devices. Similarly, Low-Power Wide-Area Network (LPWAN) protocols such as LoRa [lora], Sigfox [sigfox], NarrowBand IoT (NB-IoT) [nbiot] are all designed for resource-constrained devices that require long range and low bit rates. While these protocols allow IoT devices to conserve energy and operate efficiently, they also add additional security challenges. For example, the relatively small MTU can make security handshakes with large X509 certificates a significant overhead. At the same time, new communication paradigms also allow IoT devices to

communicate directly amongst themselves with or without support from the network. This communication paradigm is also referred to as Device-to-Device (D2D) or Machine-to-Machine (M2M) or Thing-to-Thing (T2T) communication and it is motivated by a number of features such as improved network performance, lower latency and lower energy requirements.

5.2. Bootstrapping of a Security Domain

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

5.3. Operational Challenges

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

5.3.1. Group Membership and Security

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

Symmetric and asymmetric keys can be used in group communication. Asymmetric keys have the advantage that they can provide source authentication. However, doing broadcast encryption with a single public/private key pair is also not feasible. Although a single symmetric key can be used to encrypt the communication, it has inherent risks since the capture of a single node can compromise the key shared throughout the network. The usage of symmetric-keys also does not provide source authentication. Another factor to consider is that asymmetric cryptography is more resource-intensive than symmetric key solutions. Thus, the security risks and performance trade-offs of applying either symmetric or asymmetric keys to a given

IoT use case need to be well-analyzed according to risk and usability assessments.

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

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

5.3.2. Mobility and IP Network Dynamics

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

HIP offers a simple yet effective mobility management by allowing hosts to signal changes to their associations [RFC8046]. However, slight adjustments might be necessary to reduce the cryptographic costs, for example, by making the public-key signatures in the mobility messages optional. Diet HIP does not define mobility yet but it is sufficiently similar to HIP and can use the same mechanisms. TLS and DTLS do not have native mobility support, however, work on DTLS mobility exists in the form of an Internet draft [ID-Williams]. The specific need for IP-layer mobility mainly depends on the scenario in which the nodes operate. In many cases, mobility supported by means of a mobile gateway may suffice to enable mobile IoT networks, such as body sensor networks.

5.4. Secure software update

Software updates are required not only to add new functionality to IoT devices but also to eliminate security vulnerabilities due to software bugs, design flaws, or deprecated algorithms. Software bugs might remain even after careful code review. Implementations of security protocols might contain (design) flaws. Lastly, cryptographic algorithms can become insecure due to advances in cryptanalysis.

Due to all these reasons, IoT devices have a reputation for being insecure, and yet, they are expected to stay functional for years and even decades. Additionally, these devices typically operate unattended with direct Internet connectivity. Therefore, a remote software update mechanism to fix vulnerabilities, to update configuration settings, and for adding new functionality is needed.

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

The FTC staff report on Internet of Things - Privacy & Security in a Connected World [FTCreport] and the Article 29 Working Party Opinion 8/2014 on the Recent Developments on the Internet of Things [Article29] also document the challenges for secure remote software update of IoT devices. They note that even providing such a software

update capability may add new vulnerabilities for constrained devices. For example, a buffer overflow vulnerability in the implementation of a software update protocol (TR69) [TR69] and an expired certificate in a hub device [wink] demonstrate how the software update process itself can introduce vulnerabilities.

Powerful IoT devices that run general purpose operating systems can make use of sophisticated software update mechanisms known from the desktop world. However, resource-constrained devices typically do not have any operating system and are often not equipped with a memory management unit or similar tools. Therefore, they might require more specialized solutions. An important requirement for secure software and firmware updates is source authentication. Source authentication requires the resource-constrained things to implement public-key signature verification algorithms. As stated in Section 5.1.1, resource-constrained things have limited amount of computational capabilities and energy supply available which can hinder the amount and frequency of cryptographic processing that they can perform. In addition to source authentication, software updates might require confidential delivery over a secure (encrypted) channel. The complexity of broadcast encryption can force the usage of point-to-point secure links - however, this increases the duration of a software update in a large system. Alternatively, it may force the usage of solutions in which the software update is delivered to a gateway, and then distributed to the rest of the system with a network key. Sending large amounts of data that later needs to be assembled and verified over a secure channel can consume a lot of energy and computational resources. Correct scheduling of the software updates is also a crucial design challenge. For example, a user of connected light bulbs would not want them to update and restart at night. More importantly, the user would not want all the lights to update at the same time.

Finally, it is also important to mention previous and ongoing work in the area of secure software and firmware updates at the IETF. [RFC4108] describes how Cryptographic Message Syntax (CMS) [RFC5652] can be used to protect firmware packages. The IAB has also organized a workshop to understand the challenges for secure software update of IoT devices. A summary of the workshop and the proposed next steps have been documented [iotsu]. Finally, a new working group called Firmware UpDate (fud) [WG-FUD] is currently being chartered at the IETF. The working group aims to standardize a new version [RFC4108] that reflects the best current practices for firmware update based on experience with IoT deployments. It will specifically work on describing an IoT firmware update architecture and specifying a manifest format that contains meta-data about the firmware update package.

5.5. End-of-Life

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

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

5.6. Verifying device behavior

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

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

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

and how this monitoring feature is implemented still remains an open question.

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

5.7. Testing: bug hunting and vulnerabilities

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

1. It remains to be seen how the software testing and quality assurance mechanisms used from the desktop and mobile world will be applied to IoT devices to give end users the confidence that the purchased devices are robust.
2. It is also an open question how the combination of devices from multiple vendors might actually lead to dangerous network configurations, for example, if combination of specific devices can trigger unexpected behavior.

5.8. Quantum-resistance

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

The above scenario becomes more urgent when we consider the so called "harvest and decrypt" attack in which an attacker can start to harvest (store) encrypted data today, before a quantum-computer is available, and decrypt it years later, once a quantum computer is available.

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

We refer to [ETSI_GR_QSC_001] for an extensive overview of existing quantum-resistant cryptography. [RFC7696] provides guidelines for cryptographic algorithm agility.

5.9. Privacy protection

Users will be surrounded by hundreds of connected IoT devices. Even if the communication links are encrypted and protected, information about the users might be collected for different purposes affecting their privacy. In [Ziegeldorf], privacy in IoT is defined as the threefold guarantee to the user for: 1. awareness of privacy risks imposed by smart things and services surrounding the data subject, 2. individual control over the collection and processing of personal information by the surrounding smart things, 3. awareness and control of subsequent use and dissemination of personal information by those entities to any entity outside the subject's personal control sphere.

Based on this definition, several privacy threats and challenges have been documented [Ziegeldorf] and [RFC6973]:

1. Identification - refers to the identification of the users and their objects.
2. Localization - relates to the capability of locating a user and even tracking them.
3. Profiling - is about creating a profile of the user and their preferences.

4. Interaction - occurs when a user has been profiled and a given interaction is preferred, presenting (for example, visually) some information that discloses private information.
5. Lifecycle transitions - take place when devices are, for example, sold without properly removing private data.
6. Inventory attacks - happen if specific information about (smart) objects in possession of a user is disclosed.
7. Linkage - is about when information of two or more IoT systems is combined so that a broader view on the personal data is created.

When IoT systems are deployed, the above issues should be considered to ensure that private data remains private. How to achieve this in practice is still an area of ongoing research.

5.10. Data leakage

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

1. long term keys. These long term keys can be extracted by means of a side-channel attack or reverse engineering. If these keys are exposed, then they might be used to perform attacks on devices deployed in other locations.
2. source code that might allow the attacker to determine bugs or find exploits to perform other types of attacks. The attacker might also just sell the source code.
3. proprietary algorithms. The attacker can analyze these algorithms gaining valuable know-how. The attacker can also create copies of the product (based on those proprietary algorithms) or modify the algorithms to perform more advanced attacks.

Protection against such data leakage patterns is not trivial since devices are inherently resource-constrained. An open question is whether there are any viable techniques to protect IoT devices and the data in the devices in such an adversarial model.

5.11. Trustworthy IoT Operation

Flaws in the design and implementation of a secure IoT device and network can lead to security vulnerabilities. For instance, a flaw is the distribution of an Internet-connected IoT device in which a default password is used in all devices. Many IoT devices can be found in the Internet by means of tools such as Shodan [shodan], and if they have any vulnerability, it can then be exploited at scale, for example, to launch DDoS attacks. For instance, Dyn, a major DNS, was attacked by means of a DDoS attack originated from a large IoT botnet composed of thousands of compromised IP-cameras [dyn-attack]. Open questions in this area are:

1. How to prevent large scale vulnerabilities in IoT devices?
2. How to prevent attackers from exploiting vulnerabilities in IoT devices at large scale?
3. If the vulnerability has been exploited, how do we stop a large scale attack before any damage is caused?

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

6. Conclusions and Next Steps

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

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

problems are important; however, different deployment environments have different operational and security demands. Thus, a potential approach is the definition and standardization of security profiles, each with specific mitigation strategies according to the risk assessment associated with the security profile. Such an approach would ensure minimum security capabilities in different environments while ensuring interoperability.

7. Security Considerations

This document reflects upon the requirements and challenges of the security architectural framework for the Internet of Things.

8. IANA Considerations

This document contains no request to IANA.

9. Acknowledgments

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

10. Informative References

- [Article29] "Opinion 8/2014 on the on Recent Developments on the Internet of Things", Web http://ec.europa.eu/justice/data-protection/article-29/documentation/opinion-recommendation/files/2014/wp223_en.pdf, n.d..
- [AUTO-ID] "AUTO-ID LABS", Web <http://www.autoidlabs.org/>, September 2010.
- [BACNET] "BACnet", Web <http://www.bacnet.org/>, February 2011.
- [BITAG] "Internet of Things (IoT) Security and Privacy Recommendations", Web <http://www.bitag.org/report-internet-of-things-security-privacy-recommendations.php>, n.d..

- [cctv] "Backdoor In MVPower DVR Firmware Sends CCTV Stills To an Email Address In China", Web <https://hardware.slashdot.org/story/16/02/17/0422259/backdoor-in-mvpower-dvr-firmware-sends-cctv-stills-to-an-email-address-in-china>, n.d..
- [CSA] "Security Guidance for Early Adopters of the Internet of Things (IoT)", Web https://downloads.cloudsecurityalliance.org/whitepapers/Security_Guidance_for_Early_Adopters_of_the_Internet_of_Things.pdf, n.d..
- [DALI] "DALI", Web <http://www.dalibydesign.us/dali.html>, February 2011.
- [DHS] "Strategic Principles For Securing the Internet of Things (IoT)", Web https://www.dhs.gov/sites/default/files/publications/Strategic_Principles_for_Securing_the_Internet_of_Things-2016-1115-FINAL....pdf, n.d..
- [dyn-attack] "Dyn Analysis Summary Of Friday October 21 Attack", Web <https://dyn.com/blog/dyn-analysis-summary-of-friday-october-21-attack/>, n.d..
- [ENISA_ICS] "Communication network dependencies for ICS/SCADA Systems", European Union Agency For Network And Information Security , February 2017.
- [ETSI_GR_QSC_001] "Quantum-Safe Cryptography (QSC);Quantum-safe algorithmic framework", European Telecommunications Standards Institute (ETSI) , June 2016.
- [Fairhair] "Fairhair Alliance", Web <https://www.fairhair-alliance.org/>, n.d..
- [FCC] "Federal Communications Comssion Response 12-05-2016", FCC , February 2016.

[FTCreport]

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

[GSMAsecurity]

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

[ID-6lonfc]

Choi, Y., Hong, Y., Youn, J., Kim, D., and J. Choi, "Transmission of IPv6 Packets over Near Field Communication", draft-ietf-6lo-nfc-07 (work in progress), June 2017.

[ID-6tisch]

Thubert, P., "An Architecture for IPv6 over the TSCH mode of IEEE 802.15.4", draft-ietf-6tisch-architecture-12 (work in progress), August 2017.

[ID-acedtls]

Gerdes, S., Bergmann, O., Bormann, C., Selander, G., and L. Seitz, "Datagram Transport Layer Security (DTLS) Profile for Authentication and Authorization for Constrained Environments (ACE)", draft-ietf-ace-dtls-authorize-01 (work in progress), July 2017.

[ID-aceoauth]

Seitz, L., Selander, G., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "Authentication and Authorization for Constrained Environments (ACE)", draft-ietf-ace-oauth-authz-08 (work in progress), October 2017.

[ID-bootstrap]

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

[ID-c2pq]

Hoffman, P., "The Transition from Classical to Post-Quantum Cryptography", draft-hoffman-c2pq-02 (work in progress), August 2017.

- [ID-Daniel] Park, S., Kim, K., Haddad, W., Chakrabarti, S., and J. Laganier, "IPv6 over Low Power WPAN Security Analysis", draft-daniel-6lowpan-security-analysis-05 (work in progress), March 2011.
- [ID-dietesp] Migault, D., Guggemos, T., and C. Bormann, "Diet-ESP: a flexible and compressed format for IPsec/ESP", draft-mglt-6lo-diet-esp-02 (work in progress), July 2016.
- [ID-HIP-DEX] Moskowitz, R., "HIP Diet EXchange (DEX)", draft-moskowitz-hip-rg-dex-06 (work in progress), May 2012.
- [ID-Moore] Moore, K., Barnes, R., and H. Tschofenig, "Best Current Practices for Securing Internet of Things (IoT) Devices", draft-moore-iot-security-bcp-01 (work in progress), July 2017.
- [ID-MUD] Lear, E., Droms, R., and D. Romascanu, "Manufacturer Usage Description Specification", draft-ietf-opsawg-mud-13 (work in progress), October 2017.
- [ID-Nikander] Nikander, P. and J. Melen, "A Bound End-to-End Tunnel (BEET) mode for ESP", draft-nikander-esp-beet-mode-09 (work in progress), August 2008.
- [ID-OSCOAP] Selander, G., Mattsson, J., Palombini, F., and L. Seitz, "Object Security for Constrained RESTful Environments (OSCORE)", draft-ietf-core-object-security-06 (work in progress), October 2017.
- [ID-rd] Shelby, Z., Koster, M., Bormann, C., Stok, P., and C. Amsuess, "CoRE Resource Directory", draft-ietf-core-resource-directory-11 (work in progress), July 2017.
- [ID-senml] Jennings, C., Shelby, Z., Arkko, J., Keranen, A., and C. Bormann, "Media Types for Sensor Measurement Lists (SenML)", draft-ietf-core-senml-10 (work in progress), July 2017.

- [ID-Williams]
Williams, M. and J. Barrett, "Mobile DTLS", draft-barrett-mobile-dtls-00 (work in progress), March 2009.
- [IEEE802ah]
"Status of Project IEEE 802.11ah, IEEE P802.11- Task Group AH-Meeting Update.",
Web http://www.ieee802.org/11/Reports/tgah_update.htm,
n.d..
- [IIoT]
"Industrial Internet Consortium",
Web <http://www.iiconsortium.org/>, n.d..
- [IoTSecFoundation]
"Establishing Principles for Internet of Things Security",
Web <https://iotsecurityfoundation.org/establishing-principles-for-internet-of-things-security/>, n.d..
- [iotsu]
"Patching the Internet of Things: IoT Software Update Workshop 2016", Web
<https://www.ietf.org/blog/2016/07/patching-the-internet-of-things-iot-software-update-workshop-2016/>, n.d..
- [IPSO]
"IPSO Alliance", Web <http://www.ipso-alliance.org>, n.d..
- [loral]
"LoRa - Wide Area Networks for IoT", Web <https://www.lora-alliance.org/>, n.d..
- [LWM2M]
"OMA LWM2M", Web <http://openmobilealliance.org/iot/lightweight-m2m-lwm2m>, n.d..
- [nbiot]
"NarrowBand IoT", Web
http://www.3gpp.org/ftp/tsg_ran/TSG_RAN/TSGR_69/Docs/RP-151621.zip, n.d..
- [NHTSA]
"Cybersecurity Best Practices for Modern Vehicles", Web
https://www.nhtsa.gov/staticfiles/nvs/pdf/812333_CybersecurityForModernVehicles.pdf, n.d..
- [NIST-Guide]
Ross, R., McEvelley, M., and J. Oren, "Systems Security Engineering", Web
<http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-160.pdf>, n.d..

- [NIST-SP80053] "Security and Privacy Controls for Federal Information Systems and Organizations",
Web <http://dx.doi.org/10.6028/NIST.SP.800-53r4>, n.d..
- [nist_lightweight_project] "NIST lightweight Project", Web www.nist.gov/programs-projects/lightweight-cryptography,
www.nist.gov/sites/default/files/documents/2016/10/17/sonmez-turan-presentation-lwc2016.pdf, n.d..
- [OCF] "Open Connectivity Foundation",
Web <https://openconnectivity.org/>, n.d..
- [OneM2M] "OneM2M", Web <http://www.onem2m.org/>, n.d..
- [OWASP] "IoT Security Guidance",
Web https://www.owasp.org/index.php/IoT_Security_Guidance, n.d..
- [RFC2818] Rescorla, E., "HTTP Over TLS", RFC 2818,
DOI 10.17487/RFC2818, May 2000, <<https://www.rfc-editor.org/info/rfc2818>>.
- [RFC3748] Aboba, B., Blunk, L., Vollbrecht, J., Carlson, J., and H. Levkowitz, Ed., "Extensible Authentication Protocol (EAP)", RFC 3748, DOI 10.17487/RFC3748, June 2004,
<<https://www.rfc-editor.org/info/rfc3748>>.
- [RFC3756] Nikander, P., Ed., Kempf, J., and E. Nordmark, "IPv6 Neighbor Discovery (ND) Trust Models and Threats",
RFC 3756, DOI 10.17487/RFC3756, May 2004,
<<https://www.rfc-editor.org/info/rfc3756>>.
- [RFC3833] Atkins, D. and R. Austein, "Threat Analysis of the Domain Name System (DNS)", RFC 3833, DOI 10.17487/RFC3833, August 2004,
<<https://www.rfc-editor.org/info/rfc3833>>.
- [RFC4016] Parthasarathy, M., "Protocol for Carrying Authentication and Network Access (PANA) Threat Analysis and Security Requirements", RFC 4016, DOI 10.17487/RFC4016, March 2005,
<<https://www.rfc-editor.org/info/rfc4016>>.
- [RFC4108] Housley, R., "Using Cryptographic Message Syntax (CMS) to Protect Firmware Packages", RFC 4108,
DOI 10.17487/RFC4108, August 2005, <<https://www.rfc-editor.org/info/rfc4108>>.

- [RFC4555] Eronen, P., "IKEv2 Mobility and Multihoming Protocol (MOBIKE)", RFC 4555, DOI 10.17487/RFC4555, June 2006, <<https://www.rfc-editor.org/info/rfc4555>>.
- [RFC4621] Kivinen, T. and H. Tschofenig, "Design of the IKEv2 Mobility and Multihoming (MOBIKE) Protocol", RFC 4621, DOI 10.17487/RFC4621, August 2006, <<https://www.rfc-editor.org/info/rfc4621>>.
- [RFC4738] Ignjatic, D., Dondeti, L., Audet, F., and P. Lin, "MIKEY-RSA-R: An Additional Mode of Key Distribution in Multimedia Internet KEYing (MIKEY)", RFC 4738, DOI 10.17487/RFC4738, November 2006, <<https://www.rfc-editor.org/info/rfc4738>>.
- [RFC4919] Kushalnagar, N., Montenegro, G., and C. Schumacher, "IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs): Overview, Assumptions, Problem Statement, and Goals", RFC 4919, DOI 10.17487/RFC4919, August 2007, <<https://www.rfc-editor.org/info/rfc4919>>.
- [RFC4944] Montenegro, G., Kushalnagar, N., Hui, J., and D. Culler, "Transmission of IPv6 Packets over IEEE 802.15.4 Networks", RFC 4944, DOI 10.17487/RFC4944, September 2007, <<https://www.rfc-editor.org/info/rfc4944>>.
- [RFC5191] Forsberg, D., Ohba, Y., Ed., Patil, B., Tschofenig, H., and A. Yegin, "Protocol for Carrying Authentication for Network Access (PANA)", RFC 5191, DOI 10.17487/RFC5191, May 2008, <<https://www.rfc-editor.org/info/rfc5191>>.
- [RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <<https://www.rfc-editor.org/info/rfc5246>>.
- [RFC5652] Housley, R., "Cryptographic Message Syntax (CMS)", STD 70, RFC 5652, DOI 10.17487/RFC5652, September 2009, <<https://www.rfc-editor.org/info/rfc5652>>.
- [RFC5713] Moustafa, H., Tschofenig, H., and S. De Cnodder, "Security Threats and Security Requirements for the Access Node Control Protocol (ANCP)", RFC 5713, DOI 10.17487/RFC5713, January 2010, <<https://www.rfc-editor.org/info/rfc5713>>.

- [RFC5903] Fu, D. and J. Solinas, "Elliptic Curve Groups modulo a Prime (ECP Groups) for IKE and IKEv2", RFC 5903, DOI 10.17487/RFC5903, June 2010, <<https://www.rfc-editor.org/info/rfc5903>>.
- [RFC6272] Baker, F. and D. Meyer, "Internet Protocols for the Smart Grid", RFC 6272, DOI 10.17487/RFC6272, June 2011, <<https://www.rfc-editor.org/info/rfc6272>>.
- [RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, DOI 10.17487/RFC6347, January 2012, <<https://www.rfc-editor.org/info/rfc6347>>.
- [RFC6550] Winter, T., Ed., Thubert, P., Ed., Brandt, A., Hui, J., Kelsey, R., Levis, P., Pister, K., Struik, R., Vasseur, JP., and R. Alexander, "RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks", RFC 6550, DOI 10.17487/RFC6550, March 2012, <<https://www.rfc-editor.org/info/rfc6550>>.
- [RFC6551] Vasseur, JP., Ed., Kim, M., Ed., Pister, K., Dejean, N., and D. Barthel, "Routing Metrics Used for Path Calculation in Low-Power and Lossy Networks", RFC 6551, DOI 10.17487/RFC6551, March 2012, <<https://www.rfc-editor.org/info/rfc6551>>.
- [RFC6568] Kim, E., Kaspar, D., and JP. Vasseur, "Design and Application Spaces for IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs)", RFC 6568, DOI 10.17487/RFC6568, April 2012, <<https://www.rfc-editor.org/info/rfc6568>>.
- [RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, DOI 10.17487/RFC6690, August 2012, <<https://www.rfc-editor.org/info/rfc6690>>.
- [RFC6749] Hardt, D., Ed., "The OAuth 2.0 Authorization Framework", RFC 6749, DOI 10.17487/RFC6749, October 2012, <<https://www.rfc-editor.org/info/rfc6749>>.
- [RFC6973] Cooper, A., Tschofenig, H., Aboba, B., Peterson, J., Morris, J., Hansen, M., and R. Smith, "Privacy Considerations for Internet Protocols", RFC 6973, DOI 10.17487/RFC6973, July 2013, <<https://www.rfc-editor.org/info/rfc6973>>.

- [RFC7049] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", RFC 7049, DOI 10.17487/RFC7049, October 2013, <<https://www.rfc-editor.org/info/rfc7049>>.
- [RFC7159] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7159, DOI 10.17487/RFC7159, March 2014, <<https://www.rfc-editor.org/info/rfc7159>>.
- [RFC7228] Bormann, C., Ersue, M., and A. Keranen, "Terminology for Constrained-Node Networks", RFC 7228, DOI 10.17487/RFC7228, May 2014, <<https://www.rfc-editor.org/info/rfc7228>>.
- [RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <<https://www.rfc-editor.org/info/rfc7252>>.
- [RFC7296] Kaufman, C., Hoffman, P., Nir, Y., Eronen, P., and T. Kivinen, "Internet Key Exchange Protocol Version 2 (IKEv2)", STD 79, RFC 7296, DOI 10.17487/RFC7296, October 2014, <<https://www.rfc-editor.org/info/rfc7296>>.
- [RFC7401] Moskowitz, R., Ed., Heer, T., Jokela, P., and T. Henderson, "Host Identity Protocol Version 2 (HIPv2)", RFC 7401, DOI 10.17487/RFC7401, April 2015, <<https://www.rfc-editor.org/info/rfc7401>>.
- [RFC7416] Tsao, T., Alexander, R., Dohler, M., Daza, V., Lozano, A., and M. Richardson, Ed., "A Security Threat Analysis for the Routing Protocol for Low-Power and Lossy Networks (RPLs)", RFC 7416, DOI 10.17487/RFC7416, January 2015, <<https://www.rfc-editor.org/info/rfc7416>>.
- [RFC7515] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Signature (JWS)", RFC 7515, DOI 10.17487/RFC7515, May 2015, <<https://www.rfc-editor.org/info/rfc7515>>.
- [RFC7517] Jones, M., "JSON Web Key (JWK)", RFC 7517, DOI 10.17487/RFC7517, May 2015, <<https://www.rfc-editor.org/info/rfc7517>>.
- [RFC7519] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Token (JWT)", RFC 7519, DOI 10.17487/RFC7519, May 2015, <<https://www.rfc-editor.org/info/rfc7519>>.

- [RFC7520] Miller, M., "Examples of Protecting Content Using JSON Object Signing and Encryption (JOSE)", RFC 7520, DOI 10.17487/RFC7520, May 2015, <<https://www.rfc-editor.org/info/rfc7520>>.
- [RFC7668] Nieminen, J., Savolainen, T., Isomaki, M., Patil, B., Shelby, Z., and C. Gomez, "IPv6 over BLUETOOTH(R) Low Energy", RFC 7668, DOI 10.17487/RFC7668, October 2015, <<https://www.rfc-editor.org/info/rfc7668>>.
- [RFC7696] Housley, R., "Guidelines for Cryptographic Algorithm Agility and Selecting Mandatory-to-Implement Algorithms", BCP 201, RFC 7696, DOI 10.17487/RFC7696, November 2015, <<https://www.rfc-editor.org/info/rfc7696>>.
- [RFC7744] Seitz, L., Ed., Gerdes, S., Ed., Selander, G., Mani, M., and S. Kumar, "Use Cases for Authentication and Authorization in Constrained Environments", RFC 7744, DOI 10.17487/RFC7744, January 2016, <<https://www.rfc-editor.org/info/rfc7744>>.
- [RFC7815] Kivinen, T., "Minimal Internet Key Exchange Version 2 (IKEv2) Initiator Implementation", RFC 7815, DOI 10.17487/RFC7815, March 2016, <<https://www.rfc-editor.org/info/rfc7815>>.
- [RFC7925] Tschofenig, H., Ed. and T. Fossati, "Transport Layer Security (TLS) / Datagram Transport Layer Security (DTLS) Profiles for the Internet of Things", RFC 7925, DOI 10.17487/RFC7925, July 2016, <<https://www.rfc-editor.org/info/rfc7925>>.
- [RFC8046] Henderson, T., Ed., Vogt, C., and J. Arkko, "Host Mobility with the Host Identity Protocol", RFC 8046, DOI 10.17487/RFC8046, February 2017, <<https://www.rfc-editor.org/info/rfc8046>>.
- [RFC8105] Mariager, P., Petersen, J., Ed., Shelby, Z., Van de Logt, M., and D. Barthel, "Transmission of IPv6 Packets over Digital Enhanced Cordless Telecommunications (DECT) Ultra Low Energy (ULE)", RFC 8105, DOI 10.17487/RFC8105, May 2017, <<https://www.rfc-editor.org/info/rfc8105>>.
- [RFC8152] Schaad, J., "CBOR Object Signing and Encryption (COSE)", RFC 8152, DOI 10.17487/RFC8152, July 2017, <<https://www.rfc-editor.org/info/rfc8152>>.

- [RG-T2TRG] "IRTF Thing-to-Thing (T2TRG) Research Group",
Web <https://datatracker.ietf.org/rg/t2trg/charter/>, n.d..
- [SchneierSecurity] "The Internet of Things Is Wildly Insecure--And Often Unpatchable", Web
https://www.schneier.com/essays/archives/2014/01/the_internet_of_thin.html, n.d..
- [SEAL] "Simple Encrypted Arithmetic Library - SEAL",
Web <https://sealcrypto.codeplex.com/>, n.d..
- [shodan] "Shodan", Web <https://www.shodan.io/>, n.d..
- [sigfox] "Sigfox - The Global Communications Service Provider for the Internet of Things (IoT)",
Web <https://www.sigfox.com/>, n.d..
- [Thread] "Thread Group", Web <http://threadgroup.org/>, n.d..
- [TR69] "Too Many Cooks - Exploiting the Internet-of-TR-069-Things", Web https://media.ccc.de/v/31c3_-_6166_-_en_-_saal_6_-_201412282145_-_too_many_cooks_-_exploiting_the_internet-of-tr-069-things_-_lior_oppenheim_-_shahar_tal, n.d..
- [WG-6lo] "IETF IPv6 over Networks of Resource-constrained Nodes (6lo) Working Group",
Web <https://datatracker.ietf.org/wg/6lo/charter/>, n.d..
- [WG-6LoWPAN] "IETF IPv6 over Low power WPAN (6lowpan) Working Group",
Web <http://tools.ietf.org/wg/6lowpan/>, n.d..
- [WG-ACE] "IETF Authentication and Authorization for Constrained Environments (ACE) Working Group",
Web <https://datatracker.ietf.org/wg/ace/charter/>, n.d..
- [WG-ACME] "Automated Certificate Management Environment Working Group", Web <https://datatracker.ietf.org/wg/acme/about/>, n.d..
- [WG-CoRE] "IETF Constrained RESTful Environment (CoRE) Working Group", Web <https://datatracker.ietf.org/wg/core/charter/>, n.d..

- [WG-FUD] "IETF Firmware UpDate (fud)",
Web <https://datatracker.ietf.org/wg/fud/about/>, n.d..
- [WG-LWIG] "IETF Light-Weight Implementation Guidance (LWIG) Working Group", Web <https://datatracker.ietf.org/wg/lwig/charter/>, n.d..
- [WG-MSEC] "IETF MSEC Working Group",
Web <https://datatracker.ietf.org/wg/msec/>, n.d..
- [wink] "Wink's Outage Shows Us How Frustrating Smart Homes Could Be",
Web <http://www.wired.com/2015/04/smart-home-headaches/>, n.d..
- [ZB] "ZigBee Alliance", Web <http://www.zigbee.org/>, February 2011.
- [Ziegeldorf]
Ziegeldorf, J., Garcia-Morchon, O., and K. Wehrle,,
"Privacy in the Internet of Things: Threats and Challenges", Security and Communication Networks - Special Issue on Security in a Completely Interconnected World , 2013.

Authors' Addresses

Oscar Garcia-Morchon
Philips IP&S
High Tech Campus 5
Eindhoven, 5656 AA
The Netherlands

Email: oscar.garcia-morchon@philips.com

Sandeep S. Kumar
Philips Research
High Tech Campus
Eindhoven, 5656 AA
The Netherlands

Email: sandeep.kumar@philips.com

Mohit Sethi
Ericsson
Hirsalantie 11
Jorvas, 02420
Finland

Email: mohit@piuha.net

Network Working Group
Internet-Draft
Intended status: Informational
Expires: June 16, 2019

O. Garcia-Morchon
Philips IP&S
S. Kumar
Philips Research
M. Sethi
Ericsson
December 13, 2018

State-of-the-Art and Challenges for the Internet of Things Security
draft-irtf-t2trg-iot-secons-16

Abstract

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

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

Status of This Memo

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 June 16, 2019.

Copyright Notice

Copyright (c) 2018 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. The Thing Lifecycle	4
3. Security Threats and Managing Risk	7
4. State-of-the-Art	11
4.1. IP-based IoT Protocols and Standards	11
4.2. Existing IP-based Security Protocols and Solutions	14
4.3. IoT Security Guidelines	16
5. Challenges for a Secure IoT	19
5.1. Constraints and Heterogeneous Communication	19
5.1.1. Resource Constraints	19
5.1.2. Denial-of-Service Resistance	20
5.1.3. End-to-end security, protocol translation, and the role of middleboxes	21
5.1.4. New network architectures and paradigm	23
5.2. Bootstrapping of a Security Domain	23
5.3. Operational Challenges	24
5.3.1. Group Membership and Security	24
5.3.2. Mobility and IP Network Dynamics	25
5.4. Secure software update and cryptographic agility	26
5.5. End-of-Life	28
5.6. Verifying device behavior	28
5.7. Testing: bug hunting and vulnerabilities	29
5.8. Quantum-resistance	30
5.9. Privacy protection	31
5.10. Reverse engineering considerations	32
5.11. Trustworthy IoT Operation	33

6. Conclusions and Next Steps	34
7. Security Considerations	34
8. IANA Considerations	34
9. Acknowledgments	35
10. Informative References	35
Authors' Addresses	47

1. Introduction

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

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

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

specifications as a reference for details about security considerations applying to the specified system or protocol.

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

2. The Thing Lifecycle

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

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

The thing is later installed and commissioned within a network by an installer during the bootstrapping phase. Specifically, the device

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

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

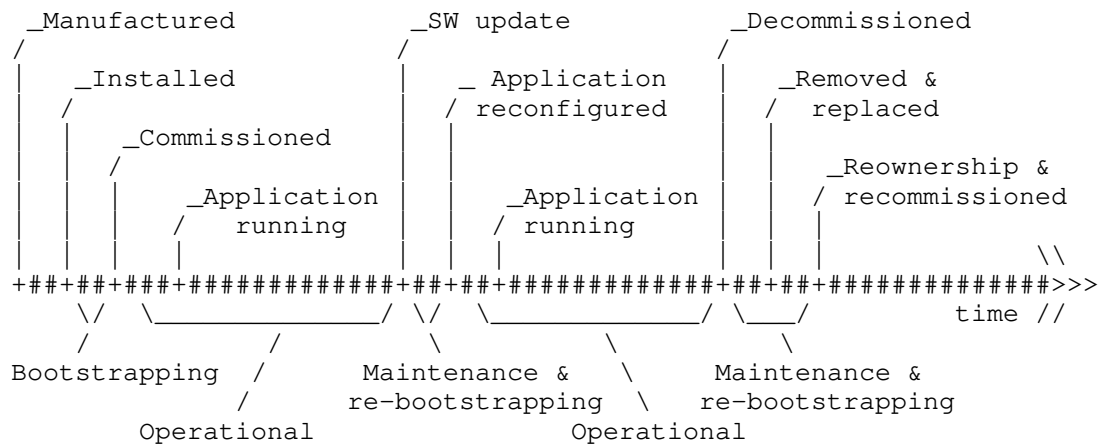


Figure 1: The lifecycle of a thing in the Internet of Things

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

The term security subsumes a wide range of primitives, protocols, and procedures. For instance, the term security includes services such as confidentiality, authentication, integrity, authorization, source authentication, and availability. The term security often also includes augmented services such as duplicate detection and detection of stale packets (timeliness). These security services can be implemented through a combination of cryptographic mechanisms such as block ciphers, hash functions, and signature algorithms; as well as

non-cryptographic mechanisms that implement authorization and other security policy enforcement aspects. For ensuring security in IoT networks, one should not only focus on the required security services, but also pay special attention to how the services are realized in the overall system.

3. Security Threats and Managing Risk

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

1. **Vulnerable Software/Code:** Things in the Internet of Things rely on software that might contain severe bugs and/or bad design choices. This makes the things vulnerable to many different types of attacks, depending on the criticality of the bugs, e.g., buffer overflows or lack of authentication. This can be considered as one of the most important security threat. The large-scale distributed denial-of-service (DDoS) attack, popularly known as the Mirai botnet [mirai], was caused by things that had well-known or easy-to-guess passwords for configuration.
2. **Privacy threat:** The tracking of a thing's location and usage may pose a privacy risk to people around it. For instance, an attacker can infer privacy sensitive information from the data gathered and communicated by individual things. Such information may subsequently be sold to interested parties for marketing purposes and targeted advertising. In extreme cases, such information might be used to track dissidents in oppressive regimes. Unlawful surveillance and interception of traffic to/from a thing by intelligence agencies is also a privacy threat.
3. **Cloning of things:** During the manufacturing process of a thing, an untrusted factory can easily clone the physical characteristics, firmware/software, or security configuration of

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

4. Malicious substitution of things: During the installation of a thing, a genuine thing may be substituted with a similar variant (of lower quality) without being detected. The main motivation may be cost savings, where the installation of lower-quality things (for example, non-certified products) may significantly reduce the installation and operational costs. The installers can subsequently resell the genuine things to gain further financial benefits. Another motivation may be to inflict damage to the reputation of a competitor's offerings.
5. Eavesdropping attack: During the commissioning of a thing into a network, it may be susceptible to eavesdropping, especially if operational keying materials, security parameters, or configuration settings, are exchanged in clear using a wireless medium or if used cryptographic algorithms are not suitable for the envisioned lifetime of the device and the system. After obtaining the keying material, the attacker might be able to recover the secret keys established between the communicating entities, thereby compromising the authenticity and confidentiality of the communication channel, as well as the authenticity of commands and other traffic exchanged over this communication channel. When the network is in operation, T2T communication can be eavesdropped if the communication channel is not sufficiently protected or if a session key is compromised due to protocol weaknesses. An adversary may also be able to eavesdrop if keys are not renewed or updated appropriately. Lastly, messages can also be recorded and decrypted offline at a later point of time. The Venona project [venona-project] is one such example where messages were recorded for offline decryption.
6. Man-in-the-middle attack: Both the commissioning phase and operational phases may also be vulnerable to man-in-the-middle

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

7. **Firmware attacks:** When a thing is in operation or maintenance phase, its firmware or software may be updated to allow for new functionality or new features. An attacker may be able to exploit such a firmware upgrade by maliciously replacing the thing's firmware, thereby influencing its operational behavior. For example, an attacker could add a piece of malicious code to the firmware that will cause it to periodically report the energy usage of the thing to a data repository for analysis. The attacker can then use this information to determine when a home or enterprise (where the thing is installed) is unoccupied and break in. Similarly, devices whose software has not been properly maintained and updated might contain vulnerabilities that might be exploited by attackers to replace the firmware on the device.
8. **Extraction of private information:** IoT devices (such as sensors, actuators, etc.) are often physically unprotected in their ambient environment and they could easily be captured by an attacker. An attacker with physical access may then attempt to extract private information such as keys (for example, device's key, private-key, group key), sensed data (for example, healthcare status of a user), configuration parameters (for example, the Wi-Fi key), or proprietary algorithms (for example, algorithm performing some data analytics task). Even when the data originating from a thing is encrypted, attackers can perform traffic analysis to deduce meaningful information which might compromise the privacy of the thing's owner and/or user.
9. **Routing attack:** As highlighted in [ID-Daniel], routing information in IoT networks can be spoofed, altered, or replayed, in order to create routing loops, attract/repel network traffic, extend/shorten source routes, etc. A non-exhaustive list of routing attacks includes 1) Sinkhole attack (or blackhole attack), where an attacker declares himself to have a high-quality route/path to the base station, thus allowing him to do manipulate all packets passing through it. 2) Selective forwarding, where an attacker may selectively forward

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

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

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

1. A Business Impact Analysis (BIA) assesses the consequences of the loss of basic security attributes: confidentiality, integrity and availability in an IoT system. These consequences might include the impact from lost data, reduced sales, increased expenses, regulatory fines, customer dissatisfaction, etc. Performing a business impact analysis allows a business to determine the relevance of having a proper security design.
2. A Risk Assessment (RA) analyzes security threats to an IoT system while considering their likelihood and impact. It also includes categorizing each of them with a risk level. Risks classified as

moderate or high must be mitigated, i.e., the security architecture should be able to deal with those threat.

3. A privacy impact assessment (PIA) aims at assessing the Personally Identifiable Information (PII) that is collected, processed, or used in an IoT system. By doing so, the goal is to fulfill applicable legal requirements, determine risks and effects of manipulation and loss of PII.
4. Procedures for incident reporting and mitigation refer to the methodologies that allow becoming aware of any security issues that affect an IoT system. Furthermore, this includes steps towards the actual deployment of patches that mitigate the identified vulnerabilities.

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

4. State-of-the-Art

This section is organized as follows. Section 4.1 summarizes state-of-the-art on IP-based IoT systems, within IETF and in other standardization bodies. Section 4.2 summarizes state-of-the-art on IP-based security protocols and their usage. Section 4.3 discusses guidelines and regulations for securing IoT as proposed by other bodies. Note that the references included in this section are a representative of the state-of-the-art at the point of writing and they are by no means exhaustive. The references are also at varying levels of maturity, and thus, it is advisable to review their specific status.

4.1. IP-based IoT Protocols and Standards

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

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

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

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

While CoAP defines a standard communication protocol, a format for representing sensor measurements and parameters over CoAP is required. The Sensor Measurement Lists (SenML) [RFC8428] is a specification that defines media types for simple sensor measurements and parameters. It has a minimalistic design so that constrained devices with limited computational capabilities can easily encode their measurements and, at the same time, servers can efficiently collect large number of measurements.

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

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

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

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

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

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

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

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

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

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

1. Thread [Thread]: Specifies the Thread protocol that is intended for a variety of IoT devices. It is an IPv6-based network protocol that runs over IEEE 802.15.4.
2. Industrial Internet Consortium [IIoT]: The consortium defines reference architectures and security frameworks for development, adoption and widespread use of Industrial Internet technologies based on existing IETF standards.
3. Internet Protocol for Smart Objects IPSO [IPSO]: The alliance specifies a common object model that enables application software on any device to interoperate with other conforming devices.
4. OneM2M [OneM2M]: The standards body defines technical and API specifications for IoT devices. It aims to create a service layer that can run on any IoT device hardware and software.
5. Open Connectivity Foundation (OCF) [OCF]: The foundation develops standards and certifications primarily for IoT devices that use Constrained Application Protocol (CoAP) as the application layer protocol.
6. Fairhair Alliance [Fairhair]: Specifies an IoT middleware to enable a common IP network infrastructure between different application standards used in building automation and lighting systems such as BACnet, KNX and ZigBee.
7. OMA LWM2M [LWM2M]: OMA Lightweight M2M is a standard from the Open Mobile Alliance for M2M and IoT device management. LWM2M relies on CoAP as the application layer protocol and uses a RESTful architecture for remote management of IoT devices.

4.2. Existing IP-based Security Protocols and Solutions

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

In the context of the IP-based IoT deployments, consideration of existing Internet security protocols is important. There are a wide range of specialized as well as general-purpose security solutions for the Internet domain such as IKEv2/IPsec [RFC7296], Transport

Layer Security (TLS) [RFC8446], Datagram Transport Layer Security (DTLS) [RFC6347], Host Identity Protocol (HIP) [RFC7401], PANA [RFC5191], Kerberos ([RFC4120]), Simple Authentication and Security Layer (SASL) [RFC4422], and Extensible Authentication Protocol (EAP) [RFC3748].

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

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

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

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

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

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

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

4.3. IoT Security Guidelines

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

1. Global System for Mobile Communications (GSM) Association (GSMA) IoT security guidelines [GSMAsecurity]: GSMA has published a set of security guidelines for the benefit of new IoT product and service providers. The guidelines are aimed at device manufacturers, service providers, developers and network operators. An enterprise can complete an IoT Security Self-Assessment to demonstrate that its products and services are aligned with the security guidelines of the GSMA.
2. Broadband Internet Technical Advisory Group (BITAG) IoT Security and Privacy Recommendations [BITAG]: BITAG has published recommendations for ensuring security and privacy of IoT device users. BITAG observes that many IoT devices are shipped from the factory with software that is already outdated and

vulnerable. The report also states that many devices with vulnerabilities will not be fixed either because the manufacturer does not provide updates or because the user does not apply them. The recommendations include that IoT devices should function without cloud and Internet connectivity, and that all IoT devices should have methods for automatic secure software updates.

3. United Kingdom Department for Digital, Culture, Media and Sport (DCMS) [DCMS]: UK DCMS has released a report that includes a list of 13 steps for improving IoT security. These steps, for example, highlight the need for implementing a vulnerability disclosure policy and keeping software updated. The report is aimed at device manufacturers, IoT service providers, mobile application developers and retailers.
4. Cloud Security Alliance (CSA) New Security Guidance for Early Adopters of the IoT [CSA]: CSA recommendations for early adopters of IoT encourages enterprises to implement security at different layers of the protocol stack. It also recommends implementation of an authentication/authorization framework for IoT deployments. A complete list of recommendations is available in the report [CSA].
5. United States Department of Homeland Security [DHS]: DHS has put forth six strategic principles that would enable IoT developers, manufacturers, service providers and consumers to maintain security as they develop, manufacture, implement or use network-connected IoT devices.
6. National Institute of Standards and Technology (NIST) [NIST-Guide]: The NIST special publication urges enterprise and US federal agencies to address security throughout the systems engineering process. The publication builds upon the International Organization for Standardization (ISO)/International Electrotechnical Commission (IEC) 15288 standard and augments each process in the system lifecycle with security enhancements.
7. National Institute of Standards and Technology (NIST) [nist-lightweight-project]: NIST is running a project on lightweight cryptography with the purpose of: (i) identifying application areas for which standard cryptographic algorithms are too heavy, classifying them according to some application profiles to be determined; (ii) determining limitations in those existing cryptographic standards; and (iii) standardizing lightweight algorithms that can be used in specific application profiles.

8. Open Web Application Security Project (OWASP) [OWASP]: OWASP provides security guidance for IoT manufactures, developers and consumers. OWASP also includes guidelines for those who intend to test and analyze IoT devices and applications.
9. IoT Security foundation [IoTSecFoundation]: IoT security foundation has published a document that enlists various considerations that need to be taken into account when developing IoT applications. For example, the document states that IoT devices could use hardware-root of trust to ensure that only authorized software runs on the devices.
10. National Highway Traffic Safety Administration (NHTSA) [NHTSA]: The US NHTSA provides guidance to the automotive industry for improving the cyber security of vehicles. While some of the guidelines are general, the document provides specific recommendations for the automotive industry such as how various automotive manufacturer can share cyber security vulnerabilities discovered.
11. Best Current Practices (BCP) for IoT devices [ID-Moore]: This document provides a list of minimum requirements that vendors of Internet of Things (IoT) devices should to take into account while developing applications, services and firmware updates in order to reduce the frequency and severity of security incidents that arise from compromised IoT devices.
12. European Union Agency for Network and Information Security (ENISA) [ENISA-ICS]: ENISA published a document on communication network dependencies for Industrial Control Systems (ICS)/Supervisory Control And Data Acquisition (SCADA) systems in which security vulnerabilities, guidelines and general recommendations are summarized.
13. Internet Society Online Trust Alliance [ISOC-OTA]: The Internet Society's IoT Trust Framework identifies the core requirements manufacturers, service providers, distributors, purchasers and policymakers need to understand, assess and embrace for effective security and privacy as part of the Internet of Things.

Other guideline and recommendation documents may exist or may later be published. This list should be considered non-exhaustive. Despite the acknowledgment that security in the Internet is needed and the existence of multiple guidelines, the fact is that many IoT devices and systems have very limited security. There are multiple reasons for this. For instance, some manufactures focus on delivering a product without paying enough attention to security.

This may be because of lack of expertise or limited budget. However, the deployment of such insecure devices poses a severe threat on the privacy and safety of users. The vast amount of devices and their inherent mobile nature also implies that an initially secure system can become insecure if a compromised device gains access to the system at some point in time. Even if all other devices in a given environment are secure, this does not prevent external attacks caused by insecure devices. Recently the Federal Communications Commission (FCC) [FCC] has stated the need for additional regulation of IoT systems. It is possible that we may see other such regional regulations in the future.

5. Challenges for a Secure IoT

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

5.1. Constraints and Heterogeneous Communication

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

5.1.1. Resource Constraints

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

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

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

There are ongoing efforts to reduce the resource consumption of security protocols by using more efficient underlying cryptographic primitives such as Elliptic Curve Cryptography [RFC8446]. The specification of elliptic curve X25519 [ecc25519], stream ciphers such as ChaCha [ChaCha], Diet HIP [ID-HIP-DEX], and ECC groups for IKEv2 [RFC5903] are all examples of efforts to make security protocols more resource efficient. Additionally, most modern security protocols have been revised in the last few years to enable cryptographic agility, making cryptographic primitives interchangeable. However, these improvements are only a first step in reducing the computation and communication overhead of Internet protocols. The question remains if other approaches can be applied to leverage key agreement in these heavily resource-constrained environments.

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

5.1.2. Denial-of-Service Resistance

The tight memory and processing constraints of things naturally alleviate resource exhaustion attacks. Especially in unattended T2T communication, such attacks are difficult to notice before the service becomes unavailable (for example, because of battery or memory exhaustion). As a DoS countermeasure, DTLS, IKEv2, HIP, and Diet HIP implement return routability checks based on a cookie mechanism to delay the establishment of state at the responding host

until the address of the initiating host is verified. The effectiveness of these defenses strongly depend on the routing topology of the network. Return routability checks are particularly effective if hosts cannot receive packets addressed to other hosts and if IP addresses present meaningful information as is the case in today's Internet. However, they are less effective in broadcast media or when attackers can influence the routing and addressing of hosts (for example, if hosts contribute to the routing infrastructure in ad-hoc networks and meshes).

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

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

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

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

If access to data or messages by the middleboxes is required or acceptable, then a diverse set of approaches for handling such a scenario are available. Note that some of these approaches affect

the meaning of end-to-end security in terms of integrity and confidentiality since the middleboxes will be able to either decrypt or modify partially the exchanged messages:

1. Sharing credentials with middleboxes enables them to transform (for example, decompress, convert, etc.) packets and re-apply the security measures after transformation. This method abandons end-to-end security and is only applicable to simple scenarios with a rudimentary security model.
2. Reusing the Internet wire format for IoT makes conversion between IoT and Internet protocols unnecessary. However, it can lead to poor performance in some use cases because IoT specific optimizations (for example, stateful or stateless compression) are not possible.
3. Selectively protecting vital and immutable packet parts with a message authentication code or with encryption requires a careful balance between performance and security. Otherwise this approach might either result in poor performance or poor security depending on which parts are selected for protection, where they are located in the original packet, and how they are processed. [ID-OSCORE] proposes a solution in this direction by encrypting and integrity protecting most of the message fields except those parts that a middlebox needs to read or change.
4. Homomorphic encryption techniques can be used in the middlebox to perform certain operations. However, this is limited to data processing involving arithmetic operations. Furthermore, performance of existing libraries, for example, SEAL [SEAL] is still too limited and homomorphic encryption techniques are not widely applicable yet.
5. Message authentication codes that sustain transformation can be realized by considering the order of transformation and protection (for example, by creating a signature before compression so that the gateway can decompress the packet without recalculating the signature). Such an approach enables IoT specific optimizations but is more complex and may require application-specific transformations before security is applied. Moreover, the usage of encrypted or integrity-protected data prevents middleboxes from transforming packets.
6. Mechanisms based on object security can bridge the protocol worlds, but still require that the two worlds use the same object security formats. Currently the object security format based on CBOR Object Signing and Encryption (COSE) [RFC8152] is different from JSON Object Signing and Encryption (JOSE) [RFC7520] or

Cryptographic Message Syntax (CMS) [RFC5652]. Legacy devices relying on traditional Internet protocols will need to update to the newer protocols for constrained environments to enable real end-to-end security. Furthermore, middleboxes do not have any access to the data and this approach does not prevent an attacker who is capable of modifying relevant message header fields that are not protected.

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

5.1.4. New network architectures and paradigm

There is a multitude of new link layer protocols that aim to address the resource-constrained nature of IoT devices. For example, the IEEE 802.11 ah [IEEE802ah] has been specified for extended range and lower energy consumption to support Internet of Things (IoT) devices. Similarly, Low-Power Wide-Area Network (LPWAN) protocols such as LoRa [loras], Sigfox [sigfox], NarrowBand IoT (NB-IoT) [nbiot] are all designed for resource-constrained devices that require long range and low bit rates. [RFC8376] provides an informational overview of the set of LPWAN technologies being considered by the IETF. It also identifies the potential gaps that exist between the needs of those technologies and the goal of running IP in such networks. While these protocols allow IoT devices to conserve energy and operate efficiently, they also add additional security challenges. For example, the relatively small MTU can make security handshakes with large X509 certificates a significant overhead. At the same time, new communication paradigms also allow IoT devices to communicate directly amongst themselves with or without support from the network. This communication paradigm is also referred to as Device-to-Device (D2D) or Machine-to-Machine (M2M) or Thing-to-Thing (T2T) communication and it is motivated by a number of features such as improved network performance, lower latency and lower energy requirements.

5.2. Bootstrapping of a Security Domain

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

5.3. Operational Challenges

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

5.3.1. Group Membership and Security

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

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

Conceptually, solutions that provide secure group communication at the network layer (IPsec/IKEv2, HIP/Diet HIP) may have an advantage in terms of the cryptographic overhead when compared to application-focused security solutions (TLS/ DTLS). This is due to the fact that application-focused solutions require cryptographic operations per group application, whereas network layer approaches may allow sharing secure group associations between multiple applications (for example, for neighbor discovery and routing or service discovery). Hence, implementing shared features lower in the communication stack can

avoid redundant security measures. However, it is important to note that sharing security contexts among different applications involves potential security threats, e.g., if one of the applications is malicious and monitors exchanged messages or injects fake messages. In the case of OSCORE, it provides security for CoAP group communication as defined in RFC7390, i.e., based on multicast IP. If the same security association is reused for each application, then this solution does not seem to have more cryptographic overhead compared to IPsec.

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

5.3.2. Mobility and IP Network Dynamics

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

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

mobile IoT networks, such as body sensor networks. Using message based application-layer security solutions such as OSCORE [ID-OSCORE] can also alleviate the problem of re-establishing lower-layer sessions for mobile nodes.

5.4. Secure software update and cryptographic agility

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

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

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

Powerful IoT devices that run general purpose operating systems can make use of sophisticated software update mechanisms known from the desktop world. However, resource-constrained devices typically do not have any operating system and are often not equipped with a

memory management unit or similar tools. Therefore, they might require more specialized solutions.

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

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

1. Would it be secure to use the existing cryptographic algorithms available on the device for updating with new cryptographic algorithms that are more secure?
2. Will the new software-based implementation fit on the device given the limited resources?
3. Would the normal operation of existing IoT applications on the device be severely hindered by the update?

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

5.5. End-of-Life

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

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

5.6. Verifying device behavior

Users using new IoT appliances such as Internet-connected smart televisions, speakers and cameras are often unaware that these devices can undermine their privacy. Recent revelations have shown that many IoT device vendors have been collecting sensitive private

data through these connected appliances with or without appropriate user warnings [cctv].

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

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

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

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

5.7. Testing: bug hunting and vulnerabilities

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

1. It remains to be seen how the software testing and quality assurance mechanisms used from the desktop and mobile world will be applied to IoT devices to give end users the confidence that the purchased devices are robust. Bodies such as the European Cyber Security Organization (ECSO) [ECSO] are working on processes for security certification of IoT devices.
2. It is also an open question how the combination of devices from multiple vendors might actually lead to dangerous network configurations. For example, if combination of specific devices can trigger unexpected behavior. It is needless to say that the security of the whole system is limited by its weakest point.

5.8. Quantum-resistance

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

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

This situation would require us to move to quantum-resistant alternatives, in particular, for those functionalities involving key exchange, public-key encryption and signatures. [ID-c2pq] describes when quantum computers may become widely available and what steps are necessary for transition to cryptographic algorithms that provide security even in presence of quantum computers. While future planning is hard, it may be a necessity in certain critical IoT deployments which are expected to last decades or more. Although increasing the key-size of the different algorithms is definitely an

option, it would also incur additional computational overhead and network traffic. This would be undesirable in most scenarios. There have been recent advancements in quantum-resistant cryptography. We refer to [ETSI-GR-QSC-001] for an extensive overview of existing quantum-resistant cryptography and [RFC7696] provides guidelines for cryptographic algorithm agility.

5.9. Privacy protection

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

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

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

1. Awareness of the privacy risks imposed by IoT devices and services. This awareness is achieved by means of transparent practices by the data controller, i.e., the entity that is providing IoT devices and/or services.
2. Individual control over the collection and processing of personal information by IoT devices and services.
3. Awareness and control of the subsequent use and dissemination of personal information by data controllers to any entity outside the subject's personal control sphere. This point implies that the data controller must be accountable for its actions on the personal information.

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

1. Identification - refers to the identification of the users, their IoT devices, and generated data.
2. Localization - relates to the capability of locating a user and even tracking them, e.g., by tracking MAC addresses in Wi-Fi or Bluetooth.
3. Profiling - is about creating a profile of the user and their preferences.
4. Interaction - occurs when a user has been profiled and a given interaction is preferred, presenting (for example, visually) some information that discloses private information.
5. Lifecycle transitions - take place when devices are, for example, sold without properly removing private data.
6. Inventory attacks - happen if specific information about IoT devices in possession of a user is disclosed.
7. Linkage - is about when information of two or more IoT systems (or other data sets) is combined so that a broader view of the personal data captured can be created.

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

5.10. Reverse engineering considerations

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

1. long term keys. These long term keys can be extracted by means of a side-channel attack or reverse engineering. If these keys are exposed, then they might be used to perform attacks on devices deployed in other locations.
2. source code. Extraction of source code might allow the attacker to determine bugs or find exploits to perform other types of attacks. The attacker might also just sell the source code.
3. proprietary algorithms. The attacker can analyze these algorithms gaining valuable know-how. The attacker can also create copies of the product (based on those proprietary algorithms) or modify the algorithms to perform more advanced attacks.
4. configuration or personal data. The attacker might be able to read personal data, e.g., healthcare data, that has been stored on a device.

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

5.11. Trustworthy IoT Operation

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

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

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

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

6. Conclusions and Next Steps

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

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

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

7. Security Considerations

This entire memo deals with security issues.

8. IANA Considerations

This document contains no request to IANA.

9. Acknowledgments

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

10. Informative References

- [Article29] "Opinion 8/2014 on the on Recent Developments on the Internet of Things", Web http://ec.europa.eu/justice/data-protection/article-29/documentation/opinion-recommendation/files/2014/wp223_en.pdf, n.d..
- [AUTO-ID] "AUTO-ID LABS", Web <http://www.autoidlabs.org/>, September 2010.
- [BACNET] "BACnet", Web <http://www.bacnet.org/>, February 2011.
- [BITAG] "Internet of Things (IoT) Security and Privacy Recommendations", Web <http://www.bitag.org/report-internet-of-things-security-privacy-recommendations.php>, n.d..
- [cctv] "Backdoor In MVPower DVR Firmware Sends CCTV Stills To an Email Address In China", Web <https://hardware.slashdot.org/story/16/02/17/0422259/backdoor-in-mvpower-dvr-firmware-sends-cctv-stills-to-an-email-address-in-china>, n.d..
- [ChaCha] Bernstein, D., "ChaCha, a variant of Salsa20", Web <http://cr.yp.to/chacha/chacha-20080128.pdf>, n.d..
- [CSA] "Security Guidance for Early Adopters of the Internet of Things (IoT)", Web https://downloads.cloudsecurityalliance.org/whitepapers/Security_Guidance_for_Early_Adopters_of_the_Internet_of_Things.pdf, n.d..
- [DALI] "DALI", Web <http://www.dalibydesign.us/dali.html>, February 2011.

- [DCMS] "Secure by Design: Improving the cyber security of consumer Internet of Things Report", Web <https://www.gov.uk/government/publications/secure-by-design>, n.d..
- [DHS] "Strategic Principles For Securing the Internet of Things (IoT)", Web https://www.dhs.gov/sites/default/files/publications/Strategic_Principles_for_Securing_the_Internet_of_Things-2016-1115-FINAL....pdf, n.d..
- [dyn-attack] "Dyn Analysis Summary Of Friday October 21 Attack", Web <https://dyn.com/blog/dyn-analysis-summary-of-friday-october-21-attack/>, n.d..
- [ecc25519] Bernstein, D., "Curve25519: new Diffie-Hellman speed records", Web <https://cr.yp.to/ecdh/curve25519-20060209.pdf>, n.d..
- [ECSO] "European Cyber Security Organization", Web <https://www.ecs-org.eu/>, n.d..
- [ENISA-ICS] "Communication network dependencies for ICS/SCADA Systems", European Union Agency For Network And Information Security , February 2017.
- [ETSI-GR-QSC-001] "Quantum-Safe Cryptography (QSC);Quantum-safe algorithmic framework", European Telecommunications Standards Institute (ETSI) , June 2016.
- [Fairhair] "Fairhair Alliance", Web <https://www.fairhair-alliance.org/>, n.d..
- [FCC] "Federal Communications Comssion Response 12-05-2016", FCC , February 2016.
- [FTCreport] "FTC Report on Internet of Things Urges Companies to Adopt Best Practices to Address Consumer Privacy and Security Risks", Web <https://www.ftc.gov/news-events/press-releases/2015/01/ftc-report-internet-things-urges-companies-adopt-best-practices>, n.d..

- [GDPR] "The EU General Data Protection Regulation",
Web <https://www.eugdpr.org/>, n.d..
- [GSMAsecurity]
"GSMA IoT Security Guidelines", Web
<http://www.gsma.com/connectedliving/future-iot-networks/iot-security-guidelines/>, n.d..
- [ID-6lonfc]
Choi, Y., Hong, Y., Youn, J., Kim, D., and J. Choi,
"Transmission of IPv6 Packets over Near Field
Communication", draft-ietf-6lo-nfc-12 (work in progress),
November 2018.
- [ID-6tisch]
Thubert, P., "An Architecture for IPv6 over the TSCH mode
of IEEE 802.15.4", draft-ietf-6tisch-architecture-18 (work
in progress), December 2018.
- [ID-acedtls]
Gerdes, S., Bergmann, O., Bormann, C., Selander, G., and
L. Seitz, "Datagram Transport Layer Security (DTLS)
Profile for Authentication and Authorization for
Constrained Environments (ACE)", draft-ietf-ace-dtls-
authorize-05 (work in progress), October 2018.
- [ID-aceoauth]
Seitz, L., Selander, G., Wahlstroem, E., Erdtman, S., and
H. Tschofenig, "Authentication and Authorization for
Constrained Environments (ACE) using the OAuth 2.0
Framework (ACE-OAuth)", draft-ietf-ace-oauth-authz-17
(work in progress), November 2018.
- [ID-bootstrap]
Sarikaya, B., Sethi, M., and D. Garcia-Carillo, "Secure
IoT Bootstrapping: A Survey", draft-sarikaya-t2trg-
sbootstrapping-05 (work in progress), September 2018.
- [ID-c2pq]
Hoffman, P., "The Transition from Classical to Post-
Quantum Cryptography", draft-hoffman-c2pq-04 (work in
progress), August 2018.
- [ID-Daniel]
Park, S., Kim, K., Haddad, W., Chakrabarti, S., and J.
Laganier, "IPv6 over Low Power WPAN Security Analysis",
draft-daniel-6lowpan-security-analysis-05 (work in
progress), March 2011.

- [ID-dietesp] Migault, D., Guggemos, T., and C. Bormann, "Diet-ESP: a flexible and compressed format for IPsec/ESP", draft-mglt-6lo-diet-esp-02 (work in progress), July 2016.
- [ID-HIP-DEX] Moskowitz, R., "HIP Diet EXchange (DEX)", draft-moskowitz-hip-rg-dex-06 (work in progress), May 2012.
- [ID-Moore] Moore, K., Barnes, R., and H. Tschofenig, "Best Current Practices for Securing Internet of Things (IoT) Devices", draft-moore-iot-security-bcp-01 (work in progress), July 2017.
- [ID-MUD] Lear, E., Droms, R., and D. Romascanu, "Manufacturer Usage Description Specification", draft-ietf-opsawg-mud-25 (work in progress), June 2018.
- [ID-multicast] Tiloca, M., Selander, G., Palombini, F., and J. Park, "Group OSCORE - Secure Group Communication for CoAP", draft-ietf-core-oscore-groupcomm-03 (work in progress), October 2018.
- [ID-OSCORE] Selander, G., Mattsson, J., Palombini, F., and L. Seitz, "Object Security for Constrained RESTful Environments (OSCORE)", draft-ietf-core-object-security-15 (work in progress), August 2018.
- [ID-rd] Shelby, Z., Kostner, M., Bormann, C., Stok, P., and C. Amsuess, "CoRE Resource Directory", draft-ietf-core-resource-directory-17 (work in progress), October 2018.
- [ID-Williams] Williams, M. and J. Barrett, "Mobile DTLS", draft-barrett-mobile-dtls-00 (work in progress), March 2009.
- [IEEE802ah] "Status of Project IEEE 802.11ah, IEEE P802.11- Task Group AH-Meeting Update.",
Web http://www.ieee802.org/11/Reports/tgah_update.htm, n.d..
- [IIoT] "Industrial Internet Consortium",
Web <http://www.iiconsortium.org/>, n.d..

- [IoTSecFoundation]
"Establishing Principles for Internet of Things Security",
Web <https://iotsecurityfoundation.org/establishing-principles-for-internet-of-things-security/>, n.d..
- [IPSO] "IPSO Alliance", Web <http://www.ipso-alliance.org>, n.d..
- [ISOC-OTA]
"Internet Society's Online Trust Alliance (OTA)",
Web <https://www.internetsociety.org/ota/>, n.d..
- [loral] "LoRa - Wide Area Networks for IoT", Web <https://www.lora-alliance.org/>, n.d..
- [LWM2M] "OMA LWM2M", Web
<http://openmobilealliance.org/iot/lightweight-m2m-lwm2m>,
n.d..
- [mirai] Koliass, C., Kambourakis, G., Stavrou, A., and J. Voas,,
"DDoS in the IoT: Mirai and Other Botnets", IEEE
Computer , 2017.
- [nbiot] "NarrowBand IoT", Web
http://www.3gpp.org/ftp/tsg_ran/TSG_RAN/TSGR_69/Docs/RP-151621.zip, n.d..
- [NHTSA] "Cybersecurity Best Practices for Modern Vehicles", Web
https://www.nhtsa.gov/staticfiles/nvs/pdf/812333_CybersecurityForModernVehicles.pdf, n.d..
- [NIST-Guide]
Ross, R., McEvelley, M., and J. Oren, "Systems Security
Engineering", Web
<http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-160.pdf>, n.d..
- [nist-lightweight-project]
"NIST lightweight Project", Web www.nist.gov/programs-projects/lightweight-cryptography,
www.nist.gov/sites/default/files/documents/2016/10/17/sonmez-turan-presentation-lwc2016.pdf, n.d..
- [NISTSP800-122]
Erika McCallister, ., Tim Grance, ., and . Karen Scarfone,
"NIST SP800-122 - Guide to Protecting the Confidentiality
of Personally Identifiable Information", Web
<https://nvlpubs.nist.gov/nistpubs/legacy/sp/nistspecialpublication800-122.pdf>, n.d..

- [NISTSP800-30r1] "NIST SP 800-30r1 - Guide for Conducting Risk Assessments", Web <https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-30r1.pdf>, n.d..
- [NISTSP800-34r1] Marianne Swanson, ., Pauline Bowen, ., Amy Wohl Phillips, ., Dean Gallup, ., and . David Lynes, "NIST SP800-34r1 - Contingency Planning Guide for Federal Information Systems", Web <https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-34r1.pdf>, n.d..
- [OCF] "Open Connectivity Foundation", Web <https://openconnectivity.org/>, n.d..
- [OneM2M] "OneM2M", Web <http://www.onem2m.org/>, n.d..
- [OWASP] "IoT Security Guidance", Web https://www.owasp.org/index.php/IoT_Security_Guidance, n.d..
- [RFC2818] Rescorla, E., "HTTP Over TLS", RFC 2818, DOI 10.17487/RFC2818, May 2000, <<https://www.rfc-editor.org/info/rfc2818>>.
- [RFC3748] Aboba, B., Blunk, L., Vollbrecht, J., Carlson, J., and H. Levkowetz, Ed., "Extensible Authentication Protocol (EAP)", RFC 3748, DOI 10.17487/RFC3748, June 2004, <<https://www.rfc-editor.org/info/rfc3748>>.
- [RFC3756] Nikander, P., Ed., Kempf, J., and E. Nordmark, "IPv6 Neighbor Discovery (ND) Trust Models and Threats", RFC 3756, DOI 10.17487/RFC3756, May 2004, <<https://www.rfc-editor.org/info/rfc3756>>.
- [RFC3833] Atkins, D. and R. Austein, "Threat Analysis of the Domain Name System (DNS)", RFC 3833, DOI 10.17487/RFC3833, August 2004, <<https://www.rfc-editor.org/info/rfc3833>>.
- [RFC4016] Parthasarathy, M., "Protocol for Carrying Authentication and Network Access (PANA) Threat Analysis and Security Requirements", RFC 4016, DOI 10.17487/RFC4016, March 2005, <<https://www.rfc-editor.org/info/rfc4016>>.

- [RFC4108] Housley, R., "Using Cryptographic Message Syntax (CMS) to Protect Firmware Packages", RFC 4108, DOI 10.17487/RFC4108, August 2005, <<https://www.rfc-editor.org/info/rfc4108>>.
- [RFC4120] Neuman, C., Yu, T., Hartman, S., and K. Raeburn, "The Kerberos Network Authentication Service (V5)", RFC 4120, DOI 10.17487/RFC4120, July 2005, <<https://www.rfc-editor.org/info/rfc4120>>.
- [RFC4422] Melnikov, A., Ed. and K. Zeilenga, Ed., "Simple Authentication and Security Layer (SASL)", RFC 4422, DOI 10.17487/RFC4422, June 2006, <<https://www.rfc-editor.org/info/rfc4422>>.
- [RFC4555] Eronen, P., "IKEv2 Mobility and Multihoming Protocol (MOBIKE)", RFC 4555, DOI 10.17487/RFC4555, June 2006, <<https://www.rfc-editor.org/info/rfc4555>>.
- [RFC4621] Kivinen, T. and H. Tschofenig, "Design of the IKEv2 Mobility and Multihoming (MOBIKE) Protocol", RFC 4621, DOI 10.17487/RFC4621, August 2006, <<https://www.rfc-editor.org/info/rfc4621>>.
- [RFC4738] Ignjatic, D., Dondeti, L., Audet, F., and P. Lin, "MIKEY-RSA-R: An Additional Mode of Key Distribution in Multimedia Internet KEYing (MIKEY)", RFC 4738, DOI 10.17487/RFC4738, November 2006, <<https://www.rfc-editor.org/info/rfc4738>>.
- [RFC4919] Kushalnagar, N., Montenegro, G., and C. Schumacher, "IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs): Overview, Assumptions, Problem Statement, and Goals", RFC 4919, DOI 10.17487/RFC4919, August 2007, <<https://www.rfc-editor.org/info/rfc4919>>.
- [RFC4944] Montenegro, G., Kushalnagar, N., Hui, J., and D. Culler, "Transmission of IPv6 Packets over IEEE 802.15.4 Networks", RFC 4944, DOI 10.17487/RFC4944, September 2007, <<https://www.rfc-editor.org/info/rfc4944>>.
- [RFC5191] Forsberg, D., Ohba, Y., Ed., Patil, B., Tschofenig, H., and A. Yegin, "Protocol for Carrying Authentication for Network Access (PANA)", RFC 5191, DOI 10.17487/RFC5191, May 2008, <<https://www.rfc-editor.org/info/rfc5191>>.

- [RFC5652] Housley, R., "Cryptographic Message Syntax (CMS)", STD 70, RFC 5652, DOI 10.17487/RFC5652, September 2009, <<https://www.rfc-editor.org/info/rfc5652>>.
- [RFC5713] Moustafa, H., Tschofenig, H., and S. De Cnodder, "Security Threats and Security Requirements for the Access Node Control Protocol (ANCP)", RFC 5713, DOI 10.17487/RFC5713, January 2010, <<https://www.rfc-editor.org/info/rfc5713>>.
- [RFC5903] Fu, D. and J. Solinas, "Elliptic Curve Groups modulo a Prime (ECP Groups) for IKE and IKEv2", RFC 5903, DOI 10.17487/RFC5903, June 2010, <<https://www.rfc-editor.org/info/rfc5903>>.
- [RFC6272] Baker, F. and D. Meyer, "Internet Protocols for the Smart Grid", RFC 6272, DOI 10.17487/RFC6272, June 2011, <<https://www.rfc-editor.org/info/rfc6272>>.
- [RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, DOI 10.17487/RFC6347, January 2012, <<https://www.rfc-editor.org/info/rfc6347>>.
- [RFC6550] Winter, T., Ed., Thubert, P., Ed., Brandt, A., Hui, J., Kelsey, R., Levis, P., Pister, K., Struik, R., Vasseur, JP., and R. Alexander, "RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks", RFC 6550, DOI 10.17487/RFC6550, March 2012, <<https://www.rfc-editor.org/info/rfc6550>>.
- [RFC6551] Vasseur, JP., Ed., Kim, M., Ed., Pister, K., Dejean, N., and D. Barthel, "Routing Metrics Used for Path Calculation in Low-Power and Lossy Networks", RFC 6551, DOI 10.17487/RFC6551, March 2012, <<https://www.rfc-editor.org/info/rfc6551>>.
- [RFC6568] Kim, E., Kaspar, D., and JP. Vasseur, "Design and Application Spaces for IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs)", RFC 6568, DOI 10.17487/RFC6568, April 2012, <<https://www.rfc-editor.org/info/rfc6568>>.
- [RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, DOI 10.17487/RFC6690, August 2012, <<https://www.rfc-editor.org/info/rfc6690>>.
- [RFC6749] Hardt, D., Ed., "The OAuth 2.0 Authorization Framework", RFC 6749, DOI 10.17487/RFC6749, October 2012, <<https://www.rfc-editor.org/info/rfc6749>>.

- [RFC6973] Cooper, A., Tschofenig, H., Aboba, B., Peterson, J., Morris, J., Hansen, M., and R. Smith, "Privacy Considerations for Internet Protocols", RFC 6973, DOI 10.17487/RFC6973, July 2013, <<https://www.rfc-editor.org/info/rfc6973>>.
- [RFC7049] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", RFC 7049, DOI 10.17487/RFC7049, October 2013, <<https://www.rfc-editor.org/info/rfc7049>>.
- [RFC7228] Bormann, C., Ersue, M., and A. Keranen, "Terminology for Constrained-Node Networks", RFC 7228, DOI 10.17487/RFC7228, May 2014, <<https://www.rfc-editor.org/info/rfc7228>>.
- [RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <<https://www.rfc-editor.org/info/rfc7252>>.
- [RFC7296] Kaufman, C., Hoffman, P., Nir, Y., Eronen, P., and T. Kivinen, "Internet Key Exchange Protocol Version 2 (IKEv2)", STD 79, RFC 7296, DOI 10.17487/RFC7296, October 2014, <<https://www.rfc-editor.org/info/rfc7296>>.
- [RFC7401] Moskowitz, R., Ed., Heer, T., Jokela, P., and T. Henderson, "Host Identity Protocol Version 2 (HIPv2)", RFC 7401, DOI 10.17487/RFC7401, April 2015, <<https://www.rfc-editor.org/info/rfc7401>>.
- [RFC7515] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Signature (JWS)", RFC 7515, DOI 10.17487/RFC7515, May 2015, <<https://www.rfc-editor.org/info/rfc7515>>.
- [RFC7516] Jones, M. and J. Hildebrand, "JSON Web Encryption (JWE)", RFC 7516, DOI 10.17487/RFC7516, May 2015, <<https://www.rfc-editor.org/info/rfc7516>>.
- [RFC7517] Jones, M., "JSON Web Key (JWK)", RFC 7517, DOI 10.17487/RFC7517, May 2015, <<https://www.rfc-editor.org/info/rfc7517>>.
- [RFC7519] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Token (JWT)", RFC 7519, DOI 10.17487/RFC7519, May 2015, <<https://www.rfc-editor.org/info/rfc7519>>.

- [RFC7520] Miller, M., "Examples of Protecting Content Using JSON Object Signing and Encryption (JOSE)", RFC 7520, DOI 10.17487/RFC7520, May 2015, <<https://www.rfc-editor.org/info/rfc7520>>.
- [RFC7668] Nieminen, J., Savolainen, T., Isomaki, M., Patil, B., Shelby, Z., and C. Gomez, "IPv6 over BLUETOOTH(R) Low Energy", RFC 7668, DOI 10.17487/RFC7668, October 2015, <<https://www.rfc-editor.org/info/rfc7668>>.
- [RFC7696] Housley, R., "Guidelines for Cryptographic Algorithm Agility and Selecting Mandatory-to-Implement Algorithms", BCP 201, RFC 7696, DOI 10.17487/RFC7696, November 2015, <<https://www.rfc-editor.org/info/rfc7696>>.
- [RFC7744] Seitz, L., Ed., Gerdes, S., Ed., Selander, G., Mani, M., and S. Kumar, "Use Cases for Authentication and Authorization in Constrained Environments", RFC 7744, DOI 10.17487/RFC7744, January 2016, <<https://www.rfc-editor.org/info/rfc7744>>.
- [RFC7815] Kivinen, T., "Minimal Internet Key Exchange Version 2 (IKEv2) Initiator Implementation", RFC 7815, DOI 10.17487/RFC7815, March 2016, <<https://www.rfc-editor.org/info/rfc7815>>.
- [RFC7925] Tschofenig, H., Ed. and T. Fossati, "Transport Layer Security (TLS) / Datagram Transport Layer Security (DTLS) Profiles for the Internet of Things", RFC 7925, DOI 10.17487/RFC7925, July 2016, <<https://www.rfc-editor.org/info/rfc7925>>.
- [RFC8046] Henderson, T., Ed., Vogt, C., and J. Arkko, "Host Mobility with the Host Identity Protocol", RFC 8046, DOI 10.17487/RFC8046, February 2017, <<https://www.rfc-editor.org/info/rfc8046>>.
- [RFC8105] Mariager, P., Petersen, J., Ed., Shelby, Z., Van de Logt, M., and D. Barthel, "Transmission of IPv6 Packets over Digital Enhanced Cordless Telecommunications (DECT) Ultra Low Energy (ULE)", RFC 8105, DOI 10.17487/RFC8105, May 2017, <<https://www.rfc-editor.org/info/rfc8105>>.
- [RFC8152] Schaad, J., "CBOR Object Signing and Encryption (COSE)", RFC 8152, DOI 10.17487/RFC8152, July 2017, <<https://www.rfc-editor.org/info/rfc8152>>.

- [RFC8240] Tschofenig, H. and S. Farrell, "Report from the Internet of Things Software Update (IoTSU) Workshop 2016", RFC 8240, DOI 10.17487/RFC8240, September 2017, <<https://www.rfc-editor.org/info/rfc8240>>.
- [RFC8259] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", STD 90, RFC 8259, DOI 10.17487/RFC8259, December 2017, <<https://www.rfc-editor.org/info/rfc8259>>.
- [RFC8376] Farrell, S., Ed., "Low-Power Wide Area Network (LPWAN) Overview", RFC 8376, DOI 10.17487/RFC8376, May 2018, <<https://www.rfc-editor.org/info/rfc8376>>.
- [RFC8428] Jennings, C., Shelby, Z., Arkko, J., Keranen, A., and C. Bormann, "Sensor Measurement Lists (SenML)", RFC 8428, DOI 10.17487/RFC8428, August 2018, <<https://www.rfc-editor.org/info/rfc8428>>.
- [RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <<https://www.rfc-editor.org/info/rfc8446>>.
- [RG-T2TRG] "IRTF Thing-to-Thing (T2TRG) Research Group", Web <https://datatracker.ietf.org/rg/t2trg/charter/>, n.d..
- [SchneierSecurity] "The Internet of Things Is Wildly Insecure--And Often Unpatchable", Web https://www.schneier.com/essays/archives/2014/01/the_internet_of_thin.html, n.d..
- [SEAL] "Simple Encrypted Arithmetic Library - SEAL", Web <https://www.microsoft.com/en-us/research/publication/simple-encrypted-arithmetic-library-seal-v2-0/>, n.d..
- [shodan] "Shodan", Web <https://www.shodan.io/>, n.d..
- [sigfox] "Sigfox - The Global Communications Service Provider for the Internet of Things (IoT)", Web <https://www.sigfox.com/>, n.d..
- [Thread] "Thread Group", Web <http://threadgroup.org/>, n.d..

- [TR69] "Too Many Cooks - Exploiting the Internet-of-TR-069-Things", Web https://media.ccc.de/v/31c3_-_6166_-_en_-_saal_6_-_201412282145_-_too_many_cooks_-_exploiting_the_internet-of-tr-069-things_-_lior_oppenheim_-_shahar_tal, n.d..
- [venona-project] "Venona Project", Web <https://www.nsa.gov/news-features/declassified-documents/venona/index.shtml>, n.d..
- [WG-6lo] "IETF IPv6 over Networks of Resource-constrained Nodes (6lo) Working Group", Web <https://datatracker.ietf.org/wg/6lo/charter/>, n.d..
- [WG-6LoWPAN] "IETF IPv6 over Low power WPAN (6lowpan) Working Group", Web <http://tools.ietf.org/wg/6lowpan/>, n.d..
- [WG-ACE] "IETF Authentication and Authorization for Constrained Environments (ACE) Working Group", Web <https://datatracker.ietf.org/wg/ace/charter/>, n.d..
- [WG-ACME] "Automated Certificate Management Environment Working Group", Web <https://datatracker.ietf.org/wg/acme/about/>, n.d..
- [WG-CoRE] "IETF Constrained RESTful Environment (CoRE) Working Group", Web <https://datatracker.ietf.org/wg/core/charter/>, n.d..
- [WG-LPWAN] "IETF Low Power Wide-Area Networks Working Group", Web <https://datatracker.ietf.org/wg/lpwan/>, n.d..
- [WG-LWIG] "IETF Light-Weight Implementation Guidance (LWIG) Working Group", Web <https://datatracker.ietf.org/wg/lwig/charter/>, n.d..
- [WG-MSEC] "IETF MSEC Working Group", Web <https://datatracker.ietf.org/wg/msec/>, n.d..
- [WG-SUIT] "IETF Software Updates for Internet of Things (suit)", Web <https://datatracker.ietf.org/group/suit/about/>, n.d..
- [WG-TEEP] "IETF Trusted Execution Environment Provisioning (teep)", Web <https://datatracker.ietf.org/wg/teep/about/>, n.d..

- [wink] "Wink's Outage Shows Us How Frustrating Smart Homes Could Be", Web <http://www.wired.com/2015/04/smart-home-headaches/>, n.d..
- [ZB] "ZigBee Alliance", Web <http://www.zigbee.org/>, February 2011.
- [Ziegeldorf]
Ziegeldorf, J., Garcia-Morchon, O., and K. Wehrle,,
"Privacy in the Internet of Things: Threats and
Challenges", Security and Communication Networks - Special
Issue on Security in a Completely Interconnected World ,
2013.

Authors' Addresses

Oscar Garcia-Morchon
Philips IP&S
High Tech Campus 5
Eindhoven, 5656 AA
The Netherlands

Email: oscar.garcia-morchon@philips.com

Sandeep S. Kumar
Philips Research
High Tech Campus
Eindhoven, 5656 AA
The Netherlands

Email: sandeep.kumar@philips.com

Mohit Sethi
Ericsson
Hirsalantie 11
Jorvas, 02420
Finland

Email: mohit@piuha.net

Network Working Group
Internet-Draft
Intended status: Informational
Expires: May 3, 2018

A. Keranen
Ericsson
M. Kovatsch
ETH Zurich
K. Hartke
Universitaet Bremen TZI
October 30, 2017

RESTful Design for Internet of Things Systems
draft-irtf-t2trg-rest-iot-00

Abstract

This document gives guidance for designing Internet of Things (IoT) systems that follow the principles of the Representational State Transfer (REST) architectural style.

Status of This Memo

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

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

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

This Internet-Draft will expire on May 3, 2018.

Copyright Notice

Copyright (c) 2017 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of

the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

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

1. Introduction

The Representational State Transfer (REST) architectural style [REST] is a set of guidelines and best practices for building distributed hypermedia systems. At its core is a set of constraints, which when fulfilled enable desirable properties for distributed software systems such as scalability and modifiability. When REST principles are applied to the design of a system, the result is often called RESTful and in particular an API following these principles is called a RESTful API.

Different protocols can be used with RESTful systems, but at the time of writing the most common protocols are HTTP [RFC7230] and CoAP [RFC7252]. Since RESTful APIs are often simple and lightweight, they are a good fit for various IoT applications. The goal of this document is to give basic guidance for designing RESTful systems and APIs for IoT applications and give pointers for more information. Design of a good RESTful IoT system has naturally many commonalities with other Web systems. Compared to other systems, the key characteristics of many IoT systems include:

- o data formats, interaction patterns, and other mechanisms that minimize, or preferably avoid, the need for human interaction
- o preference for compact and simple data formats to facilitate efficient transfer over (often) constrained networks and lightweight processing in constrained nodes

2. Terminology

This section explains some of the common terminology that is used in the context of RESTful design for IoT systems. For terminology of constrained nodes and networks, see [RFC7228].

Cache: A local store of response messages and the subsystem that controls storage, retrieval, and deletion of messages in it.

Client: A node that sends requests to servers and receives responses. In RESTful IoT systems it's common for nodes to have more than one role (e.g., both server and client; see Section 3.1).

Client State: The state kept by a client between requests. This typically includes the currently processed representation, the set of active requests, the history of requests, bookmarks (URIs stored for later retrieval), and application-specific state (e.g., local variables). (Note that this is called "Application State" in [REST], which has some ambiguity in modern (IoT) systems where

the overall state of the distributed application (i.e., application state) is reflected in the union of all Client States and Resource States of all clients and servers involved.)

Content Negotiation: The practice of determining the "best" representation for a client when examining the current state of a resource. The most common forms of content negotiation are Proactive Content Negotiation and Reactive Content Negotiation.

Form: A hypermedia control that enables a client to change the state of a resource or to construct a query locally.

Forward Proxy: An intermediary that is selected by a client, usually via local configuration rules, and that can be tasked to make requests on behalf of the client. This may be useful, for example, when the client lacks the capability to make the request itself or to service the response from a cache in order to reduce response time, network bandwidth, and energy consumption.

Gateway: A reverse proxy that provides an interface to a non-RESTful system such as legacy systems or alternative technologies such as Bluetooth ATT/GATT. See also "Reverse Proxy".

Hypermedia Control: A component, such as a link or a form, embedded in a representation that identifies a resource for future hypermedia interactions. If the client engages in an interaction with the identified resource, the result may be a change to resource state and/or client state.

Idempotent Method: A method where multiple identical requests with that method lead to the same visible resource state as a single such request.

Link: A hypermedia control that enables a client to navigate between resources and thereby change the client state.

Link Relation Type: An identifier that describes how the link target resource relates to the current resource (see [RFC5988]).

Media Type: A string such as "text/html" or "application/json" that is used to label representations so that it is known how the representation should be interpreted and how it is encoded.

Method: An operation associated with a resource. Common methods include GET, PUT, POST, and DELETE (see Section 3.5 for details).

Origin Server: A server that is the definitive source for representations of its resources and the ultimate recipient of any

request that intends to modify its resources. In contrast, intermediaries (such as proxies caching a representation) can assume the role of a server, but are not the source for representations as these are acquired from the origin server.

Proactive Content Negotiation: A content negotiation mechanism where the server selects a representation based on the expressed preference of the client. For example, an IoT application could send a request to a sensor with preferred media type "application/senml+json".

Reactive Content Negotiation: A content negotiation mechanism where the client selects a representation from a list of available representations. The list may, for example, be included by a server in an initial response. If the user agent is not satisfied by the initial response representation, it can request one or more of the alternative representations, selected based on metadata (e.g., available media types) included in the response.

Representation: A serialization that represents the current or intended state of a resource and that can be transferred between clients and servers. REST requires representations to be self-describing, meaning that there must be metadata that allows peers to understand which representation format is used. Depending on the protocol needs and capabilities, there can be additional metadata that is transmitted along with the representation.

Representation Format: A set of rules for serializing resource state. On the Web, the most prevalent representation format is HTML. Other common formats include plain text and formats based on JSON [RFC7159], XML, or RDF. Within IoT systems, often compact formats based on JSON, CBOR [RFC7049], and EXI [W3C.REC-exi-20110310] are used.

Representational State Transfer (REST): An architectural style for Internet-scale distributed hypermedia systems.

Resource: An item of interest identified by a URI. Anything that can be named can be a resource. A resource often encapsulates a piece of state in a system. Typical resources in an IoT system can be, e.g., a sensor, the current value of a sensor, the location of a device, or the current state of an actuator.

Resource State: A model of a resource's possible states that is represented in a supported representation type, typically a media type. Resources can change state because of REST interactions with them, or they can change state for reasons outside of the REST model.

Resource Type: An identifier that annotates the application-
semantics of a resource (see Section 3.1 of [RFC6690]).

Reverse Proxy: An intermediary that appears as a server towards the
client but satisfies the requests by forwarding them to the actual
server (possibly via one or more other intermediaries). A reverse
proxy is often used to encapsulate legacy services, to improve
server performance through caching, and to enable load balancing
across multiple machines.

Safe Method: A method that does not result in any state change on
the origin server when applied to a resource.

Server: A node that listens for requests, performs the requested
operation and sends responses back to the clients.

Uniform Resource Identifier (URI): A global identifier for
resources. See Section 3.3 for more details.

3. Basics

3.1. Architecture

The components of a RESTful system are assigned one or both of two
roles: client or server. Note that the terms "client" and "server"
refer only to the roles that the nodes assume for a particular
message exchange. The same node might act as a client in some
communications and a server in others. Classic user agents (e.g.,
Web browsers) are always in the client role and have the initiative
to issue requests. Origin servers always have the server role and
govern over the resources they host.

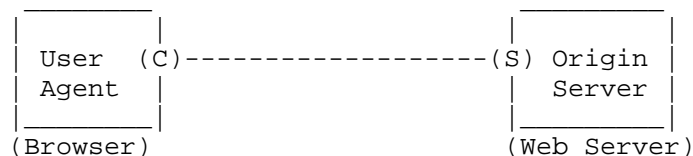


Figure 1: Client-Server Communication

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.

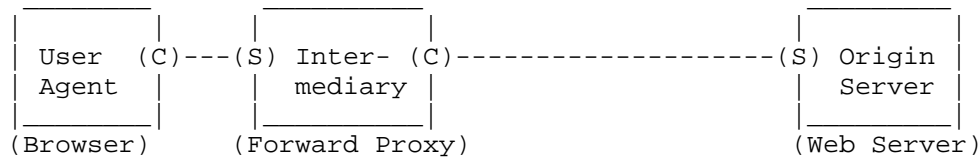


Figure 2: Communication with Forward Proxy

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).

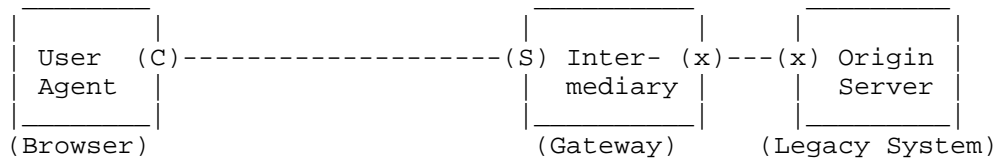


Figure 3: Communication with Reverse Proxy

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).

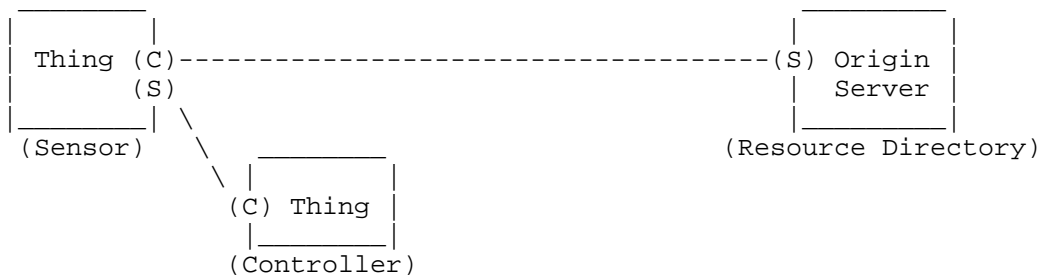


Figure 4: Constrained RESTful environments

3.2. System design

When designing a RESTful system, the primary effort goes into modeling the state of the distributed application and assigning it to the different components (i.e., clients and servers). How clients can navigate through the resources and modify state to achieve their goals is defined through hypermedia controls, that is, links and forms. Hypermedia controls span a kind of a state machine where the nodes are resources and the transitions are links or forms. Clients run this state machine (i.e., the application) by retrieving representations, processing the data, and following the included hypermedia controls. In REST, remote state is changed by submitting forms. This is usually done by retrieving the current state, modifying the state on the client side, and transferring the new state to the server in the form of new representations - rather than calling a service and modifying the state on the server side.

Client state encompasses the current state of the described state machine and the possible next transitions derived from the hypermedia controls within the currently processed representation (see Section 2). Furthermore, clients can have part of the state of the distributed application in local variables.

Resource state includes the more persistent data of an application (i.e., independent of individual clients). This can be static data such as device descriptions, persistent data such as system configurations, but also dynamic data such as the current value of a sensor on a thing.

It is important to distinguish between "client state" and "resource state" and keep them separate. Following the Stateless constraint, the client state must be kept only on clients. That is, there is no establishment of shared information about past and future interactions between client and server (usually called a session). On the one hand, this makes requests a bit more verbose since every request must contain all the information necessary to process it. On the other hand, this makes servers efficient and scalable, since they do not have to keep any state about their clients. Requests can easily be distributed over multiple worker threads or server instances. For IoT systems, this constraint lowers the memory requirements for server implementations, which is particularly important for constrained servers (e.g., sensor nodes) and servers serving large amount of clients (e.g., Resource Directory).

3.3. Uniform Resource Identifiers (URIs)

An important part of RESTful API design is to model the system as a set of resources whose state can be retrieved and/or modified and where resources can be potentially also created and/or deleted.

Uniform Resource Identifiers (URIs) are used to indicate a resource for interaction, to reference a resource from another resource, to advertise or bookmark a resource, or to index a resource by search engines.

```
foo://example.com:8042/over/there?name=ferret#nose
  \_/      \_____/ \_____/ \_____/ \_____/
  |         |         |         |         |
scheme authority  path    query  fragment
```

A URI is a sequence of characters that matches the syntax defined in [RFC3986]. It consists of a hierarchical sequence of five components: scheme, authority, path, query, and fragment (from most significant to least significant). A scheme creates a namespace for resources and defines how the following components identify a resource within that namespace. The authority identifies an entity that governs part of the namespace, such as the server "www.example.org" in the "http" scheme. A host name (e.g., a fully qualified domain name) or an IP address, potentially followed by a transport layer port number, are usually used in the authority component for the "http" and "coap" schemes. The path and query contain data to identify a resource within the scope of the URI's scheme and naming authority. The fragment allows to refer to some portion of the resource, such as a Record in a SenML Pack. However, fragments are processed only at client side and not sent on the wire. [RFC7320] provides more details on URI design and ownership with best current practices for establishing URI structures, conventions, and formats.

For RESTful IoT applications, typical schemes include "https", "coaps", "http", and "coap". These refer to HTTP and CoAP, with and without Transport Layer Security (TLS) [RFC5246]. (CoAP uses Datagram TLS (DTLS) [RFC6347], the variant of TLS for UDP.) These four schemes also provide means for locating the resource; using the HTTP protocol for "http" and "https", and with the CoAP protocol for "coap" and "coaps". If the scheme is different for two URIs (e.g., "coap" vs. "coaps"), it is important to note that even if the rest of the URI is identical, these are two different resources, in two distinct namespaces.

The query parameters can be used to parametrize the resource. For example, a GET request may use query parameters to request the server

to send only certain kind data of the resource (i.e., filtering the response). Query parameters in PUT and POST requests do not have such established semantics and are not commonly used. Whether the order of the query parameters matters in URIs is unspecified and they can be re-ordered e.g., by proxies. Therefore applications should not rely on their order; see Section 3.3 of [RFC6943] for more details.

3.4. Representations

Clients can retrieve the resource state from an origin server or manipulate resource state on the origin server by transferring resource representations. Resource representations have a media type that tells how the representation should be interpreted by identifying the representation format used.

Typical media types for IoT systems include:

- o "text/plain" for simple UTF-8 text
- o "application/octet-stream" for arbitrary binary data
- o "application/json" for the JSON format [RFC7159]
- o "application/senml+json" [I-D.ietf-core-senml] for Sensor Markup Language (SenML) formatted data
- o "application/cbor" for CBOR [RFC7049]
- o "application/exi" for EXI [W3C.REC-exi-20110310]

A full list of registered Internet Media Types is available at the IANA registry [IANA-media-types] and numerical media types registered for use with CoAP are listed at CoAP Content-Formats IANA registry [IANA-CoAP-media].

3.5. HTTP/CoAP Methods

Section 4.3 of [RFC7231] defines the set of methods in HTTP; Section 5.8 of [RFC7252] defines the set of methods in CoAP. As part of the Uniform Interface constraint, each method can have certain properties that give guarantees to clients.

Safe methods do not cause any state change on the origin server when applied to a resource. For example, the GET method only returns a representation of the resource state but does not change the resource. Thus, it is always safe for a client to retrieve a representation without affecting server-side state.

Idempotent methods can be applied multiple times to the same resource while causing the same visible resource state as a single such request. For example, the PUT method replaces the state of a resource with a new state; replacing the state multiple times with the same new state still results in the same state for the resource. However, the response from the server can be different when the same idempotent method is used multiple times. For example when DELETE is used twice on an existing resource, the first request would remove the association and return success acknowledgement whereas the second request would likely result in error response due to non-existing resource.

The following lists the most relevant methods and gives a short explanation of their semantics.

3.5.1. GET

The GET method requests a current representation for the target resource, while the origin server must ensure that there are no side-effects on the resource state. Only the origin server needs to know how each of its resource identifiers corresponds to an implementation and how each implementation manages to select and send a current representation of the target resource in a response to GET.

A payload within a GET request message has no defined semantics.

The GET method is safe and idempotent.

3.5.2. POST

The POST method requests that the target resource process the representation enclosed in the request according to the resource's own specific semantics.

If one or more resources has been created on the origin server as a result of successfully processing a POST request, the origin server sends a 201 (Created) response containing a Location header field (with HTTP) or Location-Path and/or Location-Query Options (with CoAP) that provide an identifier for the resource created. The server also includes a representation that describes the status of the request while referring to the new resource(s).

The POST method is not safe nor idempotent.

3.5.3. PUT

The PUT method requests that the state of the target resource be created or replaced with the state defined by the representation enclosed in the request message payload. A successful PUT of a given representation would suggest that a subsequent GET on that same target resource will result in an equivalent representation being sent.

The fundamental difference between the POST and PUT methods is highlighted by the different intent for the enclosed representation. The target resource in a POST request is intended to handle the enclosed representation according to the resource's own semantics, whereas the enclosed representation in a PUT request is defined as replacing the state of the target resource. Hence, the intent of PUT is idempotent and visible to intermediaries, even though the exact effect is only known by the origin server.

The PUT method is not safe, but is idempotent.

3.5.4. DELETE

The DELETE method requests that the origin server remove the association between the target resource and its current functionality.

If the target resource has one or more current representations, they might or might not be destroyed by the origin server, and the associated storage might or might not be reclaimed, depending entirely on the nature of the resource and its implementation by the origin server.

The DELETE method is not safe, but is idempotent.

3.6. HTTP/CoAP Status/Response Codes

Section 6 of [RFC7231] defines a set of Status Codes in HTTP that are used by application to indicate whether a request was understood and satisfied, and how to interpret the answer. Similarly, Section 5.9 of [RFC7252] defines the set of Response Codes in CoAP.

The status codes consist of three digits (e.g., "404" with HTTP or "4.04" with CoAP) where the first digit expresses the class of the code. Implementations do not need to understand all status codes, but the class of the code must be understood. Codes starting with 1 are informational; the request was received and being processed. Codes starting with 2 indicate a successful request. Codes starting with 3 indicate redirection; further action is needed to complete the

request. Codes 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]:

- o Performance
- o Scalability
- o Reliability
- o Simplicity
- o Modifiability
- o Visibility
- o Portability

The following sub-sections briefly summarize the REST constraints and explain how they enable the listed properties.

4.1. Client-Server

As explained in the Architecture section, RESTful system components have clear roles in every interaction. Clients have the initiative to issue requests, intermediaries can only forward requests, and servers respond requests, while origin servers are the ultimate recipient of requests that intent to modify resource state.

This improves simplicity and visibility, as it is clear which component started an interaction. Furthermore, it improves modifiability through a clear separation of concerns.

4.2. Stateless

The Stateless constraint requires messages to be self-contained. They must contain all the information to process it, independent from previous messages. This allows to strictly separate the client state from the resource state.

This improves scalability and reliability, since servers or worker threads can be replicated. It also improves visibility because message traces contain all the information to understand the logged interactions.

Furthermore, the Stateless constraint enables caching.

4.3. Cache

This constraint requires responses to have implicit or explicit cache-control metadata. This enables clients and intermediary to store responses and re-use them to locally answer future requests. The cache-control metadata is necessary to decide whether the information in the cached response is still fresh or stale and needs to be discarded.

Cache improves performance, as less data needs to be transferred and response times can be reduced significantly. Less transfers also improves scalability, as origin servers can be protected from too many requests. Local caches furthermore improve reliability, since requests can be answered even if the origin server is temporarily not available.

4.4. Uniform Interface

All RESTful APIs use the same, uniform interface independent of the application. This simple interaction model is enabled by exchanging representations and modifying state locally, which simplifies the interface between clients and servers to a small set of methods to retrieve, update, and delete state - which applies to all applications.

In contrast, in a service-oriented RPC approach, all required ways to modify state need to be modeled explicitly in the interface resulting in a large set of methods - which differs from application to application. Moreover, it is also likely that different parties come up with different ways how to modify state, including the naming of

the procedures, while the state within an application is a bit easier to agree on.

A REST interface is fully defined by:

- o URIs to identify resources
- o representation formats to represent (and retrieve and manipulate) resource state
- o self-descriptive messages with a standard set of methods (e.g., GET, POST, PUT, DELETE with their guaranteed properties)
- o hypermedia controls within representations

The concept of hypermedia controls is also known as HATEOAS: Hypermedia As The Engine Of Application State. The origin server embeds controls for the interface into its representations and thereby informs the client about possible next requests. The mostly used control for RESTful systems is Web Linking [RFC5590]. Hypermedia forms are more powerful controls that describe how to construct more complex requests, including representations to modify resource state.

While this is the most complex constraints (in particular the hypermedia controls), it improves many different key properties. It improves simplicity, as uniform interfaces are easier to understand. The self-descriptive messages improve visibility. The limitation to a known set of representation formats fosters portability. Most of all, however, this constraint is the key to modifiability, as hypermedia-driven, uniform interfaces allow clients and servers to evolve independently, and hence enable a system to evolve.

4.5. Layered System

This constraint enforces that a client cannot see beyond the server with which it is interacting.

A layered system is easier to modify, as topology changes become transparent. Furthermore, this helps scalability, as intermediaries such as load balancers can be introduced without changing the client side. The clean separation of concerns helps with simplicity.

4.6. Code-on-Demand

This principle enables origin servers to ship code to clients.

Code-on-Demand improves modifiability, since new features can be deployed during runtime (e.g., support for a new representation format). It also improves performance, as the server can provide code for local pre-processing before transferring the data.

5. Hypermedia-driven Applications

Hypermedia-driven applications take advantage of hypermedia controls, i.e., links and forms, embedded in the resource representations. A hypermedia client is a client that is capable of processing these hypermedia controls. Hypermedia links can be used to give additional information about a resource representation (e.g., the source URI of the representation) or pointing to other resources. The forms can be used to describe the structure of the data that can be sent (e.g., with a POST or PUT method) to a server, or how a data retrieval (e.g., GET) request for a resource should be formed. In a hypermedia-driven application the client interacts with the server using only the hypermedia controls, instead of selecting methods and/or constructing URIs based on out-of-band information, such as API documentation.

5.1. Motivation

The advantage of this approach is increased evolvability and extensibility. This is important in scenarios where servers exhibit a range of feature variations, where it's expensive to keep evolving client knowledge and server knowledge in sync all the time, or where there are many different client and server implementations. Hypermedia controls serve as indicators in capability negotiation. In particular, they describe available resources and possible operations on these resources using links and forms, respectively.

There are multiple reasons why a server might introduce new links or forms:

- o The server implements a newer version of the application. Older clients ignore the new links and forms, while newer clients are able to take advantage of the new features by following the new links and submitting the new forms.
- o The server offers links and forms depending on the current state. The server can tell the client which operations are currently valid and thus help the client navigate the application state machine. The client does not have to have knowledge which operations are allowed in the current state or make a request just to find out that the operation is not valid.

- o The server offers links and forms depending on the client's access control rights. If the client is unauthorized to perform a certain operation, then the server can simply omit the links and forms for that operation.

5.2. Knowledge

A client needs to have knowledge of a couple of things for successful interaction with a server. This includes what resources are available, what representations of resource states are available, what each representation describes, how to retrieve a representation, what state changing operations on a resource are possible, how to perform these operations, and so on.

Some part of this knowledge, such as how to retrieve the representation of a resource state, is typically hard-coded in the client software. For other parts, a choice can often be made between hard-coding the knowledge or acquiring it on-demand. The key to success in either case is the use in-band information for identifying the knowledge that is required. This enables the client to verify that it has all required knowledge and to acquire missing knowledge on-demand.

A hypermedia-driven application typically uses the following identifiers:

- o URI schemes that identify communication protocols,
- o Internet Media Types that identify representation formats,
- o link relation types or resource types that identify link semantics,
- o form relation types that identify form semantics,
- o variable names that identify the semantics of variables in templated links, and
- o form field names that identify the semantics of form fields in forms.

The knowledge about these identifiers as well as matching implementations have to be shared a priori in a RESTful system.

5.3. Interaction

A client begins interacting with an application through a GET request on an entry point URI. The entry point URI is the only URI a client is expected to know before interacting with an application. From there, the client is expected to make all requests by following links and submitting forms that are provided in previous responses. The entry point URI can be obtained, for example, by manual configuration or some discovery process (e.g., DNS-SD [RFC6763] or Resource Directory [I-D.ietf-core-resource-directory]). For Constrained RESTful environments `"/.well-known/core"` relative URI is defined as a default entry point for requesting the links hosted by servers with known or discovered addresses [RFC6690].

6. Design Patterns

Certain kinds of design problems are often recurring in variety of domains, and often re-usable design patterns can be applied to them. Also some interactions with a RESTful IoT system are straightforward to design; a classic example of reading a temperature from a thermometer device is almost always implemented as a GET request to a resource that represents the current value of the thermometer. However, certain interactions, for example data conversions or event handling, do not have as straightforward and well established ways to represent the logic with resources and REST methods.

The following sections describe how common design problems such as different interactions can be modeled with REST and what are the benefits of different approaches.

6.1. Collections

A common pattern in RESTful systems across different domains is the collection. A collection can be used to combine multiple resources together by providing resources that consist of set of (often partial) representations of resources, called items, and links to resources. The collection resource also defines hypermedia controls for managing and searching the items in the collection.

Examples of the collection pattern in RESTful IoT systems are the CoRE Resource Directory [I-D.ietf-core-resource-directory], CoAP pub/sub broker [I-D.ietf-core-coap-pubsub], and resource discovery via `"/.well-known/core"`. Collection+JSON [CollectionJSON] is an example of a generic collection Media Type.

6.2. Calling a Procedure

To modify resource state, clients usually use GET to retrieve a representation from the server, modify that locally, and transfer the resulting state back to the server with a PUT (see Section 4.4). Sometimes, however, the state can only be modified on the server side, for instance, because representations would be too large to transfer or part of the required information shall not be accessible to clients. In this case, resource state is modified by calling a procedure (or "function"). This is usually modeled with a POST request, as this method leaves the behavior semantics completely to the server. Procedure calls can be divided into two different classes based on how long they are expected to execute: "instantly" returning and long-running.

6.2.1. Instantly Returning Procedures

When the procedure can return within the expected response time of the system, the result can be directly returned in the response. The result can either be actual content or just a confirmation that the call was successful. In either case, the response does not contain a representation of the resource, but a so-called action result. Action results can still have hypermedia controls to provide the possible transitions in the application state machine.

6.2.2. Long-running Procedures

When the procedure takes longer than the expected response time of the system, or even longer than the response timeout, it is a good pattern to create a new resource to track the "task" execution. The server would respond instantly with a "Created" status (HTTP code 201 or CoAP 2.01) and indicate the location of the task resource in the corresponding header field (or CoAP option) or as a link in the action result. The created resource can be used to monitor the progress, to potentially modify queued tasks or cancel tasks, and to eventually retrieve the result.

Monitoring information would be modeled as state of the task resource, and hence be retrievable as representation. The result - when available - can be embedded in the representation or given as a link to another sub-resource. Modifying tasks can be modeled with forms that either update sub-resources via PUT or do a partial write using PATCH or POST. Canceling a task would be modeled with a form that uses DELETE to remove the task resource.

6.2.3. Conversion

A conversion service is a good example where REST resources need to behave more like a procedure call. The knowledge of converting from one representation to another is located only at the server to relieve clients from high processing or storing lots of data. There are different approaches that all depend on the particular conversion problem.

As mentioned in the previous sections, POST request are a good way to model functionality that does not necessarily affect resource state. When the input data for the conversion is small and the conversion result is deterministic, however, it can be better to use a GET request with the input data in the URI query part. The query is parameterizing the conversion resource, so that it acts like a look-up table. The benefit is that results can be cached also for HTTP (where responses to POST are not cacheable). In CoAP, cacheability depends on the response code, so that also a response to a POST request can be made cacheable through a 2.05 Content code.

When the input data is large or has a binary encoding, it is better to use POST requests with a proper Media Type for the input representation. A POST request is also more suitable, when the result is time-dependent and the latest result is expected (e.g., exchange rates).

6.2.4. Events as State

In event-centric paradigms such as pub/sub, events are usually represented by an incoming message that might even be identical for each occurrence. Since the messages are queued, the receiver is aware of each occurrence of the event and can react accordingly. For instance, in an event-centric system, ringing a door bell would result in a message being sent that represents the event that it was rung.

In resource-oriented paradigms such as REST, messages usually carry the current state of the remote resource, independent from the changes (i.e., events) that have lead to that state. In a naive yet natural design, a door bell could be modeled as a resource that can have the states unpressed and pressed. There are, however, a few issues with this approach. Polling is not an option, as it is highly unlikely to be able to observe the pressed state with any realistic polling interval. When using CoAP Observe with Confirmable notifications, the server will usually send two notifications for the event that the door bell was pressed: notification for changing from unpressed to pressed and another one for changing back to unpressed. If the time between the state changes is very short, the server might

drop the first notification, as Observe only guarantees only eventual consistency (see Section 1.3 of [RFC7641]).

The solution is to pick a state model that fits better to the application. In the case of the door bell - and many other event-driven resources - the solution could be a counter that counts how often the bell was pressed. The corresponding action is taken each time the client observes a change in the received representation.

In the case of a network outage, this could lead to a ringing sound 10 minutes after the bell was rung. Also including a timestamp of the last counter increment in the state can help to suppress ringing a sound when the event has become obsolete.

6.3. Server Push

Overall, a universal mechanism for server push, that is, change-of-state notifications and stand-alone event notifications, is still an open issue that is being discussed in the Thing-to-Thing Research Group. It is connected to the state-event duality problem and custody transfer, that is, the transfer of the responsibility that a message (e.g., event) is delivered successfully.

A proficient mechanism for change-of-state notifications is currently only available for CoAP: Observing resources [RFC7641]. It offers 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. Blockwise transfers [RFC7959] can help here. When the links are ordered by freshness of the events, the first block can already contain all links to new events. Then, observers do not need to retrieve the remaining blocks from the server, but only the representations of the new event resources.

An alternative pattern is to exploit the dual roles of IoT devices, in particular when using CoAP: they are usually client and server at

the same time. A client observer would subscribe to events by registering a callback URI at the origin server, e.g., using a POST request and receiving the location of a temporary subscription resource as handle. The origin server would then publish events by sending POST requests containing the event to the observer. The cancellation can be modeled through deleting the subscription resource. This pattern makes the origin server responsible for delivering the event notifications. This goes beyond retransmissions of messages; the origin server is usually supposed to queue all undelivered events and to retry until successful delivery or explicit cancellation. In HTTP, this pattern is known as REST Hooks.

In HTTP, there exist a number of workarounds to enable server push, e.g., long polling and streaming [RFC6202] or server-sent events [W3C.REC-html5-20141028]. Long polling as an extension that both server and client need to be aware of. In IoT systems, long polling can introduce a considerable overhead, as the request has to be repeated for each notification. Streaming and server-sent events (in fact an evolved version of streaming) are more efficient, as only one request is sent. However, there is only one response header and subsequent notifications can only have content. There are no means for individual status and metadata, and hence no means for proficient error handling (e.g., when the resource is deleted).

7. Security Considerations

This document does not define new functionality and therefore does not introduce new security concerns. We assume that system designers apply classic Web security on top of the basic RESTful guidance given in this document. Thus, security protocols and considerations from related specifications apply to RESTful IoT design. These include:

- o Transport Layer Security (TLS): [RFC5246] and [RFC6347]
- o Internet X.509 Public Key Infrastructure: [RFC5280]
- o HTTP security: Section 9 of [RFC7230], Section 9 of [RFC7231], etc.
- o CoAP security: Section 11 of [RFC7252]
- o URI security: Section 7 of [RFC3986]

IoT-specific security is mainly work in progress at the time of writing. First specifications include:

- o (D)TLS Profiles for the Internet of Things: [RFC7925]

Further IoT security considerations are available in [I-D.irtf-t2trg-iot-seccons].

8. Acknowledgement

The authors would like to thank Mert Ocak, Heidi-Maria Back, Tero Kauppinen, Michael Koster, Robby Simpson, Ravi Subramaniam, Dave Thaler, Erik Wilde, and Niklas Widell for the reviews and feedback.

9. References

9.1. Normative References

- [I-D.ietf-core-object-security]
Selandar, G., Mattsson, J., Palombini, F., and L. Seitz,
"Object Security for Constrained RESTful Environments
(OSCORE)", draft-ietf-core-object-security-06 (work in
progress), October 2017.
- [I-D.ietf-core-resource-directory]
Shelby, Z., Koster, M., Bormann, C., Stok, P., and C.
Amsuess, "CoRE Resource Directory", draft-ietf-core-
resource-directory-12 (work in progress), October 2017.
- [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](https://www.rfc-editor.org/info/rfc5246)>.
- [RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S.,
Housley, R., and W. Polk, "Internet X.509 Public Key
Infrastructure Certificate and Certificate Revocation List
(CRL) Profile", RFC 5280, DOI 10.17487/RFC5280, May 2008,
<<https://www.rfc-editor.org/info/rfc5280>>.
- [RFC5590] Harrington, D. and J. Schoenwaelder, "Transport Subsystem
for the Simple Network Management Protocol (SNMP)", STD
78, RFC 5590, DOI 10.17487/RFC5590, June 2009,
<<https://www.rfc-editor.org/info/rfc5590>>.

- [RFC5988] Nottingham, M., "Web Linking", RFC 5988, DOI 10.17487/RFC5988, October 2010, <<https://www.rfc-editor.org/info/rfc5988>>.
- [RFC6202] Loreto, S., Saint-Andre, P., Salsano, S., and G. Wilkins, "Known Issues and Best Practices for the Use of Long Polling and Streaming in Bidirectional HTTP", RFC 6202, DOI 10.17487/RFC6202, April 2011, <<https://www.rfc-editor.org/info/rfc6202>>.
- [RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, DOI 10.17487/RFC6347, January 2012, <<https://www.rfc-editor.org/info/rfc6347>>.
- [RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, DOI 10.17487/RFC6690, August 2012, <<https://www.rfc-editor.org/info/rfc6690>>.
- [RFC7049] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", RFC 7049, DOI 10.17487/RFC7049, October 2013, <<https://www.rfc-editor.org/info/rfc7049>>.
- [RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <<https://www.rfc-editor.org/info/rfc7230>>.
- [RFC7231] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", RFC 7231, DOI 10.17487/RFC7231, June 2014, <<https://www.rfc-editor.org/info/rfc7231>>.
- [RFC7641] Hartke, K., "Observing Resources in the Constrained Application Protocol (CoAP)", RFC 7641, DOI 10.17487/RFC7641, September 2015, <<https://www.rfc-editor.org/info/rfc7641>>.
- [RFC7959] Bormann, C. and Z. Shelby, Ed., "Block-Wise Transfers in the Constrained Application Protocol (CoAP)", RFC 7959, DOI 10.17487/RFC7959, August 2016, <<https://www.rfc-editor.org/info/rfc7959>>.
- [W3C.REC-exi-20110310]
Schneider, J. and T. Kamiya, "Efficient XML Interchange (EXI) Format 1.0", World Wide Web Consortium Recommendation REC-exi-20110310, March 2011, <<http://www.w3.org/TR/2011/REC-exi-20110310>>.

[W3C.REC-html5-20141028]

Hickson, I., Berjon, R., Faulkner, S., Leithead, T., Navara, E., O'Connor, T., and S. Pfeiffer, "HTML5", World Wide Web Consortium Recommendation REC-html5-20141028, October 2014, <<http://www.w3.org/TR/2014/REC-html5-20141028>>.

9.2. Informative References

[CollectionJSON]

Amundsen, M., "Collection+JSON - Document Format", February 2013, <<http://amundsen.com/media-types/collection/format/>>.

[I-D.ietf-core-coap-pubsub]

Koster, M., Keranen, A., and J. Jimenez, "Publish-Subscribe Broker for the Constrained Application Protocol (CoAP)", draft-ietf-core-coap-pubsub-02 (work in progress), July 2017.

[I-D.ietf-core-senml]

Jennings, C., Shelby, Z., Arkko, J., Keranen, A., and C. Bormann, "Media Types for Sensor Measurement Lists (SenML)", draft-ietf-core-senml-10 (work in progress), July 2017.

[I-D.irtf-t2trg-iot-secons]

Garcia-Morchon, O., Kumar, S., and M. Sethi, "State-of-the-Art and Challenges for the Internet of Things Security", draft-irtf-t2trg-iot-secons-08 (work in progress), October 2017.

[IANA-CoAP-media]

"CoAP Content-Formats", n.d., <<http://www.iana.org/assignments/core-parameters/core-parameters.xhtml#content-formats>>.

[IANA-media-types]

"Media Types", n.d., <<http://www.iana.org/assignments/media-types/media-types.xhtml>>.

[RFC6763]

Cheshire, S. and M. Krochmal, "DNS-Based Service Discovery", RFC 6763, DOI 10.17487/RFC6763, February 2013, <<https://www.rfc-editor.org/info/rfc6763>>.

[RFC6943]

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

- [RFC7159] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7159, DOI 10.17487/RFC7159, March 2014, <<https://www.rfc-editor.org/info/rfc7159>>.
- [RFC7228] Bormann, C., Ersue, M., and A. Keranen, "Terminology for Constrained-Node Networks", RFC 7228, DOI 10.17487/RFC7228, May 2014, <<https://www.rfc-editor.org/info/rfc7228>>.
- [RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <<https://www.rfc-editor.org/info/rfc7252>>.
- [RFC7320] Nottingham, M., "URI Design and Ownership", BCP 190, RFC 7320, DOI 10.17487/RFC7320, July 2014, <<https://www.rfc-editor.org/info/rfc7320>>.
- [RFC7925] Tschofenig, H., Ed. and T. Fossati, "Transport Layer Security (TLS) / Datagram Transport Layer Security (DTLS) Profiles for the Internet of Things", RFC 7925, DOI 10.17487/RFC7925, July 2016, <<https://www.rfc-editor.org/info/rfc7925>>.

Appendix A. Future Work

- o Interface semantics: shared knowledge among system components (URI schemes, media types, relation types, well-known locations; see core-apps)
- o Unreliable (best effort) communication, robust communication in network with high packet loss, 3-way commit
- o Discuss directories, such as CoAP Resource Directory
- o More information on how to design resources; choosing what is modeled as a resource, etc.

Authors' Addresses

Ari Keranen
Ericsson
Jorvas 02420
Finland

Email: ari.keranen@ericsson.com

Matthias Kovatsch
ETH Zurich
Universitaetstrasse 6
Zurich CH-8092
Switzerland

Email: kovatsch@inf.ethz.ch

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

Email: hartke@tzi.org

Network Working Group
Internet-Draft
Intended status: Informational
Expires: 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.

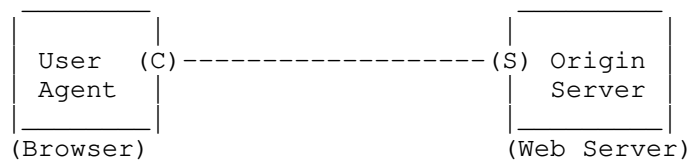


Figure 1: Client-Server Communication

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].

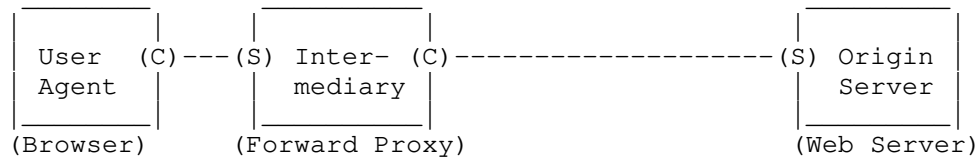


Figure 2: Communication with Forward Proxy

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).

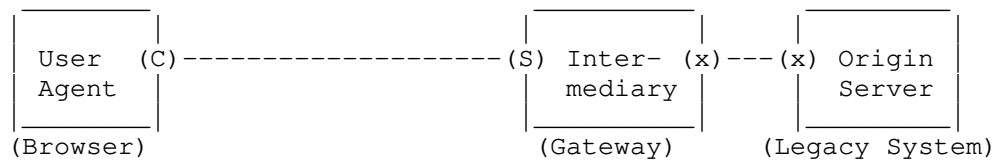


Figure 3: Communication with Reverse Proxy

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).

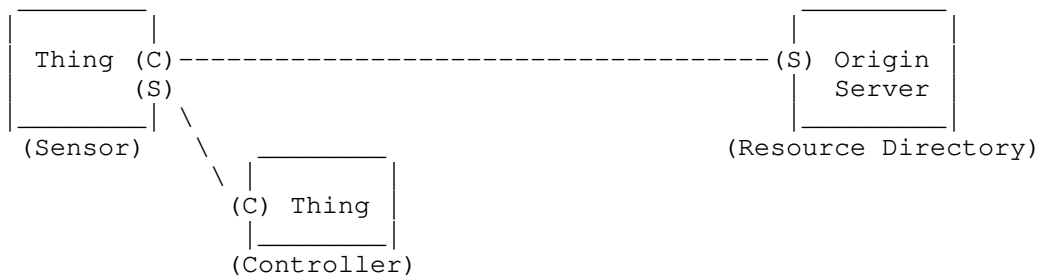


Figure 4: Constrained RESTful environments

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

CURDLE
Internet-Draft
Intended status: Informational
Expires: May 3, 2018

R. Moskowitz
L. Xia
Huawei
October 30, 2017

Small Crypto for Small IOT
draft-moskowitz-small-crypto-00

Abstract

This memo proposes to leverage the Keccak algorithm at a function "width" of b=400 to provide a set of "small" cryptographic functions well suited to the IOT constrained environment. As such, only 128 bit security level is provided here. The full set of NIST approved Keccak derived functions that can work within the b=400 constraint, plus the 3rd round candidate in the CAESAR competition, Ketje, are defined.

Status of This Memo

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

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

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

This Internet-Draft will expire on May 3, 2018.

Copyright Notice

Copyright (c) 2017 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of

the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
1.1. Document status	3
2. Terms and Definitions	3
2.1. Requirements Terminology	3
2.2. Notations	3
2.3. Definitions	3
3. Keccak Basics	4
3.1. Keccak Parameters	4
4. Keccak[144,256]	4
5. The Ketje authenticated encryption scheme	5
6. Using SHAKE128i in Ed25519	5
7. IANA Considerations	5
8. Security Considerations	5
8.1. Defense against future attacks	5
8.2. Security Comparisons	5
8.2.1. SHAKE128i to SHA-256	5
8.2.2. KMAC128i to HMAC(SHA-256)	5
8.2.3. Ketje Sr to AES-CCM	5
8.2.4. Ed25519i to Ed25519	6
9. Acknowledgments	6
10. References	6
10.1. Normative References	6
10.2. Informative References	6
Appendix A. Use Cases	7
A.1. Pacemaker connected via Body Area Network	7
A.2. Automotive CAN FD Sensors	7
A.3. Building Automation and Control Network Sensors	8
Appendix B. Performance Comparisons	8
B.1. SHAKE128i to SHA-256	8
B.2. KMAC128i to HMAC(SHA-256)	8
B.3. Ketje Sr to AES-CCM	8
B.4. Ed25519i to Ed25519	8
B.5. Keccak Hardware to AES Hardware	8
Authors' Addresses	8

1. Introduction

The Keccak [Keccak] algorithm at a width of $b=1600$ was selected by NIST for SHA-3 as defined in FIPS 202 [NIST.FIPS.202]. It is further used to define additional hashing functions in NIST SP 800-185 [NIST.SP.800_185], all with $b=1600$. This selection is well suited for 64 bit processors and large messages. It can take advantage of multiple core CPUs and works well even on 32 bit processors. It is

not well suited for small messages and 8 bit processors that are common in IOT.

A full set of function widths of 25, 50, 100, 200, 400, and 800 are also defined in Keccak. Selection of values of other than 1600 is a risk/design trade off. A width of 400 is used here as the smallest that can provide 128 bits security strength.

Keccak provides more than a secure hash function. FIPS 202 defines the SHAKE function to provide a hash output of arbitrary length, rather than current practice of truncating a longer hash to the needed length. SP 800-185 defines a keyed hash, KMAC, that does not require the HMAC complexity and computational cost. This also can provide a PRF.

Finally, Keccak also provides an AEAD cipher, Ketje, to thus deliver in a single base function, the full suite of symmetric cryptographic functions.

1.1. Document status

Significant portions of this document are stubs. That is not because the information is not available. Rather the information for those sections need to be formatted for inclusion in this document. All the algorithms exist for Keccak and need only be adjusted for B=400. There is considerable performance data and security comparison data from the Keccak team. It needs extraction from other publications and formatted into this document.

2. Terms and Definitions

2.1. Requirements Terminology

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

2.2. Notations

This section will contain notations

2.3. Definitions

TBD

3. Keccak Basics

Keccak is a extendable-output function (XOF) sponge construction hash. It breaks from the 'standard' ARX (addition, rotation and exclusive-or (XOR)) hash approach. Instead it using only bit-level transpositions, bit-level additions and multiplications (in $GF(2)$). This contributes to its superior performance.

This leads to one important nomenclature difference in Keccak. It does not have a block size. Rather in Keccak there is bit-rate which is one of the variable parameters.

3.1. Keccak Parameters

The Keccak primitive is:

Keccak-f[b], where b is 25, 50, 100, 200, 400, 800 or 1600 bits

b=1600 is used in FIPS 202 and SP 800-185. Here, b=400 is used.

Instances are denoted

Keccak[r, c]

capacity c determines the proven security strength against generic attacks
for a security level of n bits, the capacity must be $c = 2n$
and bitrate $r = b - c$

Thus for b=400 and a strength of 128 bits, r=144 and c=256

4. Keccak[144,256]

The instance Keccak[144,256] can be used for SHAKE128 [FIPS 202], cSHAKE128, KMAC128, KMACXOF128, TupleHash128, TupleHashXOF128, ParallelHash128, ParallelHashXOF128 [SP 800-185]. The difference is a bitrate of 144 rather than 1344.

Some of the above variants, such as the parallel hashes are not of value in small devices with small r which is not amendable to tree hashing.

To distinguish the form of the above hashes between the standard r=1344 and r=144, a designation of 'i' is appended to each name so that here SHAKE128i and KMAC128i are used.

5. The Ketje authenticated encryption scheme

Ketje Sr [Ketje] is already defined to use $b=400$.

6. Using SHAKE128i in Ed25519

Ed25519, RFC 8032 [RFC8032], specifies the parameter $H(x)$ as SHA-512. A new form, Ed25519i, will use SHAKE128i for $H(x)$. This one change will bring Ed25519 more into the 'reach' of the constrained environment.

The use of SHAKE128i is the only variant between Ed25519 and Ed25519i.

7. IANA Considerations

TBD. OID assignments

8. Security Considerations

8.1. Defense against future attacks

The safety margin in Keccak can be increased or de-created simply by changing the number of rounds in Keccak-f. This is explained in "Note on Keccak parameters and usage" [NotesPandU], Section 6.

8.2. Security Comparisons

This memo proposes a radical change in the basic crypto-primatives used in protocols. As such, care is called for.

Keccak is not new. It has been well reviewed as explained in "Why Keccak is not ARX" [notARX]. Still, it is important to compare each Keccak function proposed here against the current Best Practice.

8.2.1. SHAKE128i to SHA-256

TBD.

8.2.2. KMAC128i to HMAC(SHA-256)

TBD.

8.2.3. Ketje Sr to AES-CCM

TBD.

8.2.4. Ed25519i to Ed25519

TBD.

9. Acknowledgments

Sections here draw heavily on information available on the Keccak [Keccak] site. I thank the Keccak Team in making this information openly available.

10. References

10.1. Normative References

[NIST.FIPS.202]

Dworkin, M., "SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions", FIPS PUB 202, DOI 10.6028/nist.fips.202, August 2015, <<http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.202.pdf>>.

[NIST.SP.800_185]

Kelsey, J., Change, S., and R. Perlner, "SHA-3 Derived Functions: cSHAKE, KMAC, TupleHash and ParallelHash", Special Publication SP 800-185, DOI 10.6028/nist.sp.800-185, December 2016, <<http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-185.pdf>>.

[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>>.

[RFC8032]

Josefsson, S. and I. Liusvaara, "Edwards-Curve Digital Signature Algorithm (EdDSA)", RFC 8032, DOI 10.17487/RFC8032, January 2017, <<https://www.rfc-editor.org/info/rfc8032>>.

10.2. Informative References

[IEEE.802.15.6_2012]

IEEE, "IEEE Standard for Local and metropolitan area networks - Part 15.6: Wireless Body Area Networks", IEEE 802.15.6-2012, DOI 10.1109/ieeestd.2012.6161600, February 2012, <<http://ieeexplore.ieee.org/servlet/opac?punumber=6161598>>.

- [Keccak] Bertoni, G., Daemen, J., Peeters, M., Van Assche, G., and R. Van Keer, "Team Keccak Home Page", <<https://keccak.team/index.html>>.
- [Ketje] Bertoni, G., Daemen, J., Peeters, M., Van Assche, G., and R. Van Keer, "The Ketje authenticated encryption scheme", <<https://keccak.team/ketje.html>>.
- [notARX] "Why Keccak is not ARX", <https://keccak.team/2017/not_arx.html>.
- [NotesPandU] Bertoni, G., Daemen, J., Peeters, M., and G. Van Assche, "Note on Keccak parameters and usage", February 2010, <<https://keccak.team/files/NoteOnKeccakParametersAndUsage.pdf>>.
- [RFC8163] Lynn, K., Ed., Martocci, J., Neilson, C., and S. Donaldson, "Transmission of IPv6 over Master-Slave/Token-Passing (MS/TP) Networks", RFC 8163, DOI 10.17487/RFC8163, May 2017, <<https://www.rfc-editor.org/info/rfc8163>>.

Appendix A. Use Cases

TBD.

A.1. Pacemaker connected via Body Area Network

In-body sensors, like pacemakers, connected via the Body Area Network (BAN [IEEE.802.15.6_2012]) will be very power conscious. Battery replacement can often require surgery. This will lead to loose interpretation of the HIPAA protection requirements to the detriment of the patient. In-body sensors are an important class of devices that would benefit from the most power and code efficient security components that still meet a strong security claim.

A.2. Automotive CAN FD Sensors

CAN FD (CAN with Flexible Data-Rate) is an extension to the original CAN bus protocol specified in ISO 11898-1. Developed in 2011 and released in 2012. Its larger data payload of 64 bytes can support a encrypting and authenticating protocol, but various in-vehicle constraints can make this challenging. Sensors may be in a very high temperature environment, making even a marginal CPU temperature increase due to heavy computations catastrophic. Cost is always a automotive component constraint and security tends to come in last in the cost competition. Thus any way that the cost of security can be lowered is a major win.

A.3. Building Automation and Control Network Sensors

BACnet (ISO 16484-5) is the standard for wired, in-building control systems. Recently IPv6 support (RFC 8163 [RFC8163]) was added. Some BACnet devices are extremely constrained; 4-bit processors are still common. Though AES on such 4-bit processors is available, it is extremely slow. A smaller security suite would be a real benefit for this environment.

Appendix B. Performance Comparisons

Most new crypto algorithm changes are to reduce risk. This proposal is to reduce cost and thus enable security in a lower class of devices as currently supported. This will only come about if there is a real performance improvement to justify potential non-interoperability.

Performance improvements can come for lower CPU/power demands, smaller code size, and/or smaller storage/memory requirements.

B.1. SHAKE128i to SHA-256

TBD. Magnitude faster?

B.2. KMAC128i to HMAC(SHA-256)

TBD.

B.3. Ketje Sr to AES-CCM

TBD.

B.4. Ed25519i to Ed25519

TBD.

B.5. Keccak Hardware to AES Hardware

TBD.

Authors' Addresses

Robert Moskowitz
Huawei
Oak Park, MI 48237

Email: rgm@labs.htt-consult.com

Liang Xia
Huawei
No. 101, Software Avenue, Yuhuatai District
Nanjing
China

Email: Frank.xialiang@huawei.com