

Network Working Group
Internet-Draft
Intended status: Informational
Expires: 15 July 2022

J. Hong
ETRI
Y-G. Hong
Daejeon University
X. de Foy
InterDigital Communications, LLC
M. Kovatsch
Huawei Technologies Duesseldorf GmbH
E. Schooler
Intel
D. Kutscher
University of Applied Sciences Emden/Leer
11 January 2022

IoT Edge Challenges and Functions
draft-irtf-t2trg-iot-edge-04

Abstract

Many IoT applications have requirements that cannot be met by the traditional Cloud (aka cloud computing). These include time sensitivity, data volume, connectivity cost, operation in the face of intermittent services, privacy, and security. As a result, the IoT is driving the Internet toward Edge computing. This document outlines the requirements of the emerging IoT Edge and its challenges. It presents a general model, and major components of the IoT Edge, to provide a common base for future discussions in T2TRG and other IRTF and IETF groups.

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 15 July 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. Background	3
2.1. Internet of Things (IoT)	3
2.2. Cloud Computing	4
2.3. Edge Computing	4
2.4. Examples of IoT Edge Computing Use Cases	6
3. IoT Challenges Leading Towards Edge Computing	9
3.1. Time Sensitivity	9
3.2. Connectivity Cost	10
3.3. Resilience to Intermittent Services	10
3.4. Privacy and Security	10
4. IoT Edge Computing Functions	11
4.1. Overview of IoT Edge Computing Today	11
4.2. General Model	13
4.3. OAM Components	16
4.3.1. Resource Discovery and Authentication	16
4.3.2. Edge Organization and Federation	17
4.3.3. Multi-Tenancy and Isolation	18
4.4. Functional Components	18
4.4.1. In-Network Computation	18
4.4.2. Edge Storage and Caching	20
4.4.3. Northbound/Southbound Communication	20
4.4.4. Communication Brokering	21
4.5. Application Components	21
4.5.1. IoT End Devices Management	21
4.5.2. Data Management and Analytics	22
4.6. Simulation and Emulation Environments	23
5. Security Considerations	23
6. Acknowledgements	24
7. Informative References	24
Authors' Addresses	31

1. Introduction

Currently, many IoT services leverage the Cloud, since it can provide virtually unlimited storage and processing power. The reliance of IoT on back-end cloud computing brings additional advantages such as flexibility and efficiency. Today's IoT systems are fairly static with respect to integrating and supporting computation. It's not that there is no computation, but systems are often limited to static configurations (edge gateways, cloud services).

However, IoT devices are creating vast amounts of data at the network edge. To meet IoT use case requirements, that data increasingly is being stored, processed, analyzed, and acted upon close to the data producers. These requirements include time sensitivity, data volume, connectivity cost, resiliency in the face of intermittent connectivity, privacy, and security, which cannot be addressed by today's centralized cloud computing. These requirements suggest a more flexible way to distribute computing (and storage) and to integrate it in the edge-cloud continuum. We will refer to this integration of edge computing and IoT as "IoT edge computing". Our draft describes related background, uses cases, challenges, system models, and functional components.

Due to the dynamic nature of the IoT edge computing landscape, this document does not list existing projects in this field. However, Section 4.1 presents a high-level overview of the field, based on a limited review of standards, research, open-source and proprietary products in [I-D-defoy-t2trg-iot-edge-computing-background].

2. Background

2.1. Internet of Things (IoT)

Since the term "Internet of Things" (IoT) was coined by Kevin Ashton in 1999 working on Radio-Frequency Identification (RFID) technology [Ashton], the concept of IoT has evolved. It now reflects a vision of connecting the physical world to the virtual world of computers using (wireless) networks over which things can send and receive information without human intervention. Recently, the term has become more literal by actually connecting things to the Internet and converging on Internet and Web technology.

Things are usually embedded systems of various kinds, such as home appliances, mobile equipment, wearable devices, etc. Things are widely distributed, but typically have limited storage and processing power, which raise concerns regarding reliability, performance, energy consumption, security, and privacy [Lin]. This limited storage and processing power leads to complementing IoT with cloud computing.

2.2. Cloud Computing

Cloud computing has been defined in [NIST]: "cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction". Low cost and massive availability of storage and processing power enabled the realization of another computing model, in which virtualized resources can be leased in an on-demand fashion, being provided as general utilities. Companies like Amazon, Google, Facebook, etc. widely adopted this paradigm for delivering services over the Internet, gaining both economical and technical benefits [Botta].

Today, an unprecedented volume and variety of data is generated by things, and applications deployed at the network edge consume this data. In this context, cloud-based service models are not suitable for some classes of applications, which for example need very short response times, access to local personal data, or generate vast amounts of data. Those applications may instead leverage edge computing.

2.3. Edge Computing

Edge computing, also referred to as fog computing in some settings, is a new paradigm in which substantial computing and storage resources are placed at the edge of the Internet, that is, close to mobile devices, sensors, actuators, or machines. Edge computing happens near data sources [Mahadev], or closer (topologically, physically, in terms of latency, etc.) to where decisions or interactions with the physical world are happening. It processes both downstream data, e.g. originated from cloud services, and upstream data, e.g. originated from end devices or network elements. The term fog computing usually represents the notion of a multi-tiered edge computing, that is, several layers of compute infrastructure between the end devices and cloud services.

An edge device is any computing or networking resource residing between end-devices' data sources and cloud-based data centers. In edge computing, end devices not only consume data but also produce data. And at the network edge, devices not only request services and information from the Cloud, but also handle computing tasks including processing, storage, caching, and load balancing on data sent to and from the Cloud [Shi]. This does not preclude end devices from hosting computation themselves when possible, independently or as part of a distributed edge computing platform (this is also referred to as Mist Computing).

Several standards developing organization (SDO) and industry forums have provided definitions of edge and fog computing:

- * ISO defines edge computing as a "form of distributed computing in which significant processing and data storage takes place on nodes which are at the edge of the network" [ISO_TR].
- * ETSI defines multi-access edge computing as a "system which provides an IT service environment and cloud-computing capabilities at the edge of an access network which contains one or more type of access technology, and in close proximity to its users" [ETSI_MEC_01].
- * The Industrial IoT Consortium (IIC, formerly OpenFog) defines fog computing as "a horizontal, system-level architecture that distributes computing, storage, control and networking functions closer to the users along a cloud-to-thing continuum" [OpenFog].

Based on these definitions, we can summarize a general philosophy of edge computing as to distribute the required functions close to users and data, while the difference to classic local systems is the usage of management and orchestration features adopted from cloud computing.

Actors from various industries approach edge computing using different terms and reference models, although in practice these approaches are not incompatible and may integrate with each other:

- * The telecommunication industry tends to use a model where edge computing services are deployed over Network Function Virtualization (NFV) infrastructure, at aggregation points or in proximity to the user equipment (e.g., gNodeBs) [ETSI_MEC_03].
- * Enterprise and campus solutions often interpret edge computing as an "edge cloud", that is, a smaller data center directly connected to the local network (often referred to as "on-premise").

- * The automation industry defines the edge as the connection point between IT from OT (Operational Technology). Hence, here edge computing sometimes refers to applying IT solutions to OT problems such as analytics, more flexible user interfaces, or simply having more computing power than an automation controller.

2.4. Examples of IoT Edge Computing Use Cases

IoT edge computing can be used in home, industry, grid, healthcare, city, transportation, agriculture, and/or education scenarios. We discuss here only a few examples of such use cases, to point out differentiating requirements. These examples are followed with references to other use cases.

Smart Factory

As part of the 4th industrial revolution, smart factories run real-time processes based on IT technologies such as artificial intelligence and big data. In a smart factory, even a very small environmental change can lead to a situation in which production efficiency decreases or product quality problems occur. Therefore, simple but time-sensitive processing can be performed at the edge: for example, controlling temperature and humidity in the factory, or operating machines based on the real-time collection of the operational status of each machine. On the other hand, data requiring highly precise analysis, such as machine lifecycle management or accident risk prediction, can be transferred to a central data center for processing.

The use of edge computing in a smart factory can reduce the cost of network and storage resources by reducing the communication load to the central data center or server. It is also possible to improve process efficiency and facility asset productivity through the real-time prediction of failures, and to reduce the cost of failure through preliminary measures. In the existing manufacturing field, production facilities are manually run according to a program entered in advance, but edge computing in a smart factory enables tailoring solutions by analyzing data at each production facility and machine level. Digital twins [Jones] of IoT devices have been use jointly with edge computing in industrial IoT scenarios [Chen].

Smart Grid

In future smart city scenarios, the Smart Grid will be critical in ensuring highly available/efficient energy control in city-wide electricity management. Edge computing is expected to play a significant role in those systems to improve transmission efficiency of electricity; to react and restore power after a disturbance; to

reduce operation costs and reuse renewable energy effectively, since these operations involve local decision-making. In addition, edge computing can help to monitor power generation and power demand, and making local electrical energy storage decisions in the smart grid system.

Smart Agriculture

Smart agriculture integrates information and communication technology with farming technology. Intelligent farms use IoT technology to measure and analyze temperature, humidity, sunlight, carbon dioxide, soil, etc. in crop cultivation facilities. Depending on analysis results, control devices are used to set environmental parameters to an appropriate state. Remote management is also possible through mobile devices such as smartphones.

In existing farms, simple systems such as management according to temperature and humidity can easily and inexpensively be implemented with IoT technology. Sensors in fields are gathering data on field and crop condition. This data is then transmitted to cloud servers, which process data and recommend actions. Usage of edge computing can reduce by a large amount data transmitted up and down the network, resulting in saving cost and bandwidth. Locally generated data can be processed at the edge, and local computing and analytics can drive local actions. With edge computing, it is also easy for farmers to select large amounts of data for processing, and data can be analyzed even in remote areas with poor access conditions. As the number of people working on farming decreases over time, increasing automation enabled by edge computing can be a driving force for future smart agriculture.

Smart Construction

Safety is critical on a construction site. Every year, many construction workers lose their lives due to falls, collisions, electric shocks, and other accidents. Therefore, solutions have been developed in order to improve construction site safety, including real-time identification of workers, monitoring of equipment location, and predictive accident prevention. To deploy these solutions, many cameras and IoT sensors were installed on construction sites, that measure noise, vibration, gas concentration, etc. Typically, data generated from these measurements has been collected in an on-site gateway and sent to a remote cloud server for storage and analysis. Thus, an inspector can check the information stored on the cloud server to investigate an incident. However, this approach can be expensive, due to transmission costs, e.g., of video streams over an LTE connection, and due to usage fees of private cloud services such as Amazon Web Services.

Using edge computing, data generated on the construction site can be processed and analyzed on an edge server located within or near the site. Only the result of this processing needs to be transferred to a cloud server, thus saving transmission costs. It is also possible to locally generate warnings to prevent accident in real-time.

Self-Driving Car

The self-driving car, with its focus on safety, is a system where edge computing has an essential role. Autonomous vehicles are equipped with high-resolution cameras, radars, laser scanners (LIDAR), sonar sensors, and GPS systems. Edge computing nodes collect and analyze vast amounts of data generated in real-time by these sensors to keep track of distances between vehicles in front, surrounding road conditions, vehicle flow, and to quickly respond to unexpected situations. For example, if the speed of the car running in front decreases, speed should be adjusted to maintain the distance between the cars, and when a roadside signal changes, a self-driving car should operate according to the new signal. If such processing is performed in a central data center, network delays or data transmission errors can lead to accidents. Applying edge computing can minimize these network delays and data transmission errors, thereby improving safety. In the shorter term we can expect edge computing nodes to be at the base station or in road-side units. However, to further reduce reaction times, some edge computing nodes should be located in the vehicle itself.

AR/VR

Augmented Reality (AR) and Virtual Reality (VR) are likely to strongly influence the Information and Communication Technology (ICT) market in the future, since they can support innovative products in most other use cases including smart factories, self-driving cars, etc. In AR/VR, due to large amounts of data generated at endpoints such as mobile devices and PCs, user immersion can be significantly decreased by a latency of only a few hundred milliseconds. Therefore, using an edge computing infrastructure built close to endpoints can not only reduce the cost and latency of data transmission but also maximize user immersion. For example, in AR using edge computing, streaming video can be displayed realistically in higher quality, giving users the best possible experience.

Other Use Cases

Additionally, oneM2M recently studied several use cases related to edge computing, including: smart factories, smart transportation, an accident notification service, a high-precision road map service, a vulnerable road user service and a vehicular data service. These use

cases are documented in [oneM2M-TR0001], [oneM2M-TR0018] and [oneM2M-TR0026]. Edge computing related requirements raised through the analysis of these use cases are captured in [oneM2M-TS0002].

3. IoT Challenges Leading Towards Edge Computing

This section describes challenges met by IoT, that are motivating the adoption of edge computing for IoT. Those are distinct from research challenges applicable to IoT edge computing, some of which will be mentioned in Section 4.3.

IoT technology is used with more and more demanding applications, e.g. in industrial, automotive or healthcare domains, leading to new challenges. For example, industrial machines such as laser cutters already produce over 1 terabyte per hour, and similar amounts can be generated in autonomous cars [NVIDIA]. 90% of IoT data is expected to be stored, processed, analyzed, and acted upon close to the source [Kelly], as cloud computing models alone cannot address the new challenges [Chiang].

Below we discuss IoT use case requirements that are moving cloud capabilities to be more proximate and more distributed and disaggregated.

3.1. Time Sensitivity

Many industrial control systems, such as manufacturing systems, smart grids, oil and gas systems, etc., often require stringent end-to-end latency between the sensor and control node. While some IoT applications may require latency below a few tens of milliseconds [Weiner], industrial robots and motion control systems have use cases for cycle times in the order of microseconds [_60802]. In some cases speed-of-light limitations may simply prevent a solution based on remote cloud, however it is not the only challenge relative to time sensitivity. Guarantees for jitter are also important to those industrial IoT applications. This means control packets need to arrive with as little variation as possible and within a strict deadline. Given the best-effort characteristics of the Internet, this challenge is virtually impossible to address, without using end-to-end guarantees for individual message delivery and continuous data flows.

3.2. Connectivity Cost

Some IoT deployments are not challenged by a constrained network bandwidth to the Cloud. The fifth generation mobile networks (5G) and Wi-Fi 6 both theoretically top out at 10 gigabits per second (i.e., 4.5 terabytes per hour), which enables high-bandwidth uplinks. However, the resulting cost for high-bandwidth connectivity to upload all data to the Cloud is unjustifiable and impractical for most IoT applications. In some settings, e.g. in aeronautical communication, higher communication costs reduce the amount of data that can be practically uploaded even further.

3.3. Resilience to Intermittent Services

Many IoT devices such as sensors, data collectors, actuators, controllers, etc. have very limited hardware resources and cannot rely solely on their limited resources to meet all their computing and/or storage needs. They require reliable, uninterrupted, or resilient services to augment their capabilities in order to fulfill their application tasks. This is hard and partly impossible to achieve with cloud services for systems such as vehicles, drones, or oil rigs that have intermittent network connectivity. The dual is also true, a cloud back-end might want to have a reading of the device even if it's currently asleep.

3.4. Privacy and Security

When IoT services are deployed at home, personal information can be learned from detected usage data. For example, one can extract information about employment, family status, age, and income by analyzing smart meter data [ENERGY]. Policy-makers started to provide frameworks that limit the usage of personal data and put strict requirements on data controllers and processors. However, data stored indefinitely in the Cloud also increases the risk of data leakage, for instance, through attacks on rich targets.

Industrial systems are often argued to not have privacy implications, as no personal data is gathered. Yet data from such systems is often highly sensitive, as one might be able to infer trade secrets such as the setup of production lines. Hence, the owners of these systems are generally reluctant to upload IoT data to the Cloud.

Furthermore, passive observers can perform traffic analysis on the device-to-cloud path. Hiding traffic patterns associated with sensor networks can therefore be another requirement for edge computing.

4. IoT Edge Computing Functions

In this section, we first look at the current state of IoT edge computing Section 4.1, and then define a general system model Section 4.2. This provides context for IoT edge computing functions, which are listed in Section 4.3.

4.1. Overview of IoT Edge Computing Today

This section provides an overview of today's IoT edge computing field, based on a limited review of standards, research, open-source and proprietary products in [I-D-defoy-t2trg-iot-edge-computing-background].

IoT gateways, both open-source (such as EdgeX Foundry or Home Edge) and proprietary (such as Amazon Greengrass, Microsoft Azure IoT Edge, Google Cloud IoT Core, and gateways from Bosh, Siemens), represent a common class of IoT edge computing products, where the gateway is providing a local service on customer premises and is remotely managed through a cloud service. IoT communication protocols are typically used between IoT devices and the gateway, including CoAP, MQTT, and many specialized IoT protocols (such as OPC UA and DDS in the Industrial IoT space), while the gateway communicates with the distant cloud typically using HTTPS. Virtualization platforms enable the deployment of virtual edge computing functions (using VMs, application containers, etc.), including IoT gateway software, on servers in the mobile network infrastructure (at base stations and concentration points), in edge data centers (in central offices) or regional data centers located near central offices. End devices are envisioned to become computing devices in forward-looking projects, but they are not commonly used as such today.

Besides open-source and proprietary solutions, a horizontal IoT service layer is standardized by the oneM2M standards body, to reduce fragmentation, increase interoperability and promote reuse in the IoT ecosystem.

Physical or virtual IoT gateways can host application programs, which are typically built using an SDK to access local services through a programmatic API. Edge cloud system operators host their customers' applications VMs or containers on servers located in or near access networks, which can implement local edge services. For example, mobile networks can provide edge services for radio network information, location, and bandwidth management.

Life cycle management of services and applications on physical IoT gateways is often cloud-based. Edge cloud management platforms and products (such as StarlingX, Akraino Edge Stack, Mobile EdgeX) adapt

cloud management technologies (e.g., Kubernetes) to the edge cloud, i.e., to smaller, distributed computing devices running outside a controlled data center. Services and application life-cycle is typically using an NFV-like management and orchestration model.

Resilience in IoT often entails the ability to operate autonomously in periods of disconnectedness in order to preserve the integrity and safety of the controlled system, possibly in a degraded mode. IoT devices and gateways are often expected to operate in the always-on and unattended mode, using fault detection and unassisted recovery functions.

The platform typically includes services to advertise or consume APIs (e.g., Mpl interface in ETSI MEC supports service discovery and communication), and enables communicating with local and remote endpoints (e.g., message routing function in IoT gateways). The service platform is typically extensible by edge applications, since they can advertise an API that other edge applications can consume. IoT communication services include protocols translation, analytics, and transcoding. Communication between edge computing devices is enabled in tiered deployments or distributed deployments.

An edge cloud platform may enable pass-through without storage or local storage (e.g., on IoT gateways). Some edge cloud platforms use a distributed form of storage such as an ICN network, e.g., Named Function Networking (NFN) nodes can store data in a Named Data Networking (NDN) system, or a distributed storage platform (e.g., IPFS, EdgeFS, Ceph). External storage, e.g., on databases in distant or local IT cloud, is typically used for filtered data deemed worthy of long-term storage, although in some cases it may be for all data, for example when required for regulatory reasons.

Stateful computing is supported on platforms hosting native programs, VMs or containers. Stateless computing is supported on platforms providing a "serverless computing" service (a.k.a. function-as-a-service, e.g., using stateless containers), or on systems based on named function networking.

In many IoT use cases, a typical network usage pattern is high volume uplink with some form of traffic reduction enabled by processing over edge computing devices. Alternatives to traffic reduction include deferred transmission (to off-peak hours or using physical shipping). Downlink traffic includes application control and software updates. Other, downlink-heavy traffic patterns are not excluded but are more often associated with non-IoT usage (e.g., video CDNs).

4.2. General Model

Edge computing is expected to play an important role in deploying new IoT services integrated with Big Data and AI, enabled by flexible in-network computing platforms. Although there are lots of approaches to edge computing, we attempt to lay out a general model and list associated logical functions in this section. In practice, this model can map to different architectures, such as:

- * A single IoT gateway, or a hierarchy of IoT gateways, typically connected to the cloud (e.g., to extend the traditional cloud-based management of IoT devices and data to the edge). A common role of an IoT Gateway is to provide access to a heterogeneous set of IoT devices/sensors; handle IoT data; and deliver IoT data to its final destination in a cloud network. Whereas an IoT gateway needs interactions with the cloud, it can also operate independently in a disconnected mode.
- * A set of distributed computing nodes, e.g., embedded in switches, routers, edge cloud servers, or mobile devices. Some IoT end devices can have enough computing capabilities to participate in such distributed systems due to advances in hardware technology. In this model, edge computing nodes can collaborate to share their resources.

In the general model described in Figure 1, the edge computing domain is interconnected with IoT end devices (southbound connectivity) and possibly with a remote/cloud network (northbound connectivity), and with a service operator's system. Edge computing nodes provide multiple logical functions, or components, which may not all be present in a given system. They may be implemented in a centralized or distributed fashion, at the network edge, or through some interworking between edge network and remote cloud network.

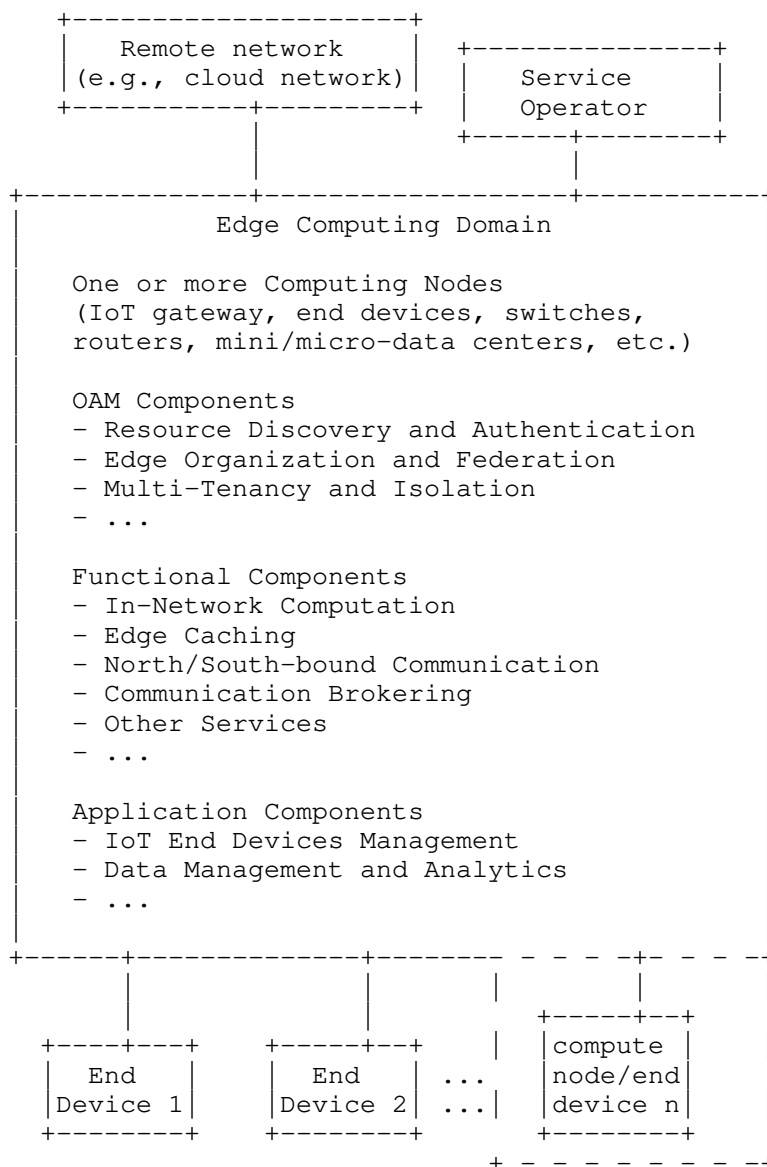


Figure 1: Model of IoT Edge Computing

In the distributed model described in Figure 2, the edge computing domain is composed of IoT edge gateways and IoT end devices which are also used as computing nodes. Edge computing domains are connected with a remote/cloud network, and with their respective service operator's system. IoT end devices/computing nodes provide logical

functions, as part of a distributed machine learning application. The processing capabilities in IoT end devices being limited, they require the support of other nodes: the training process for AI services is executed at IoT edge gateways or cloud networks and the prediction (inference) service is executed in the IoT end devices.

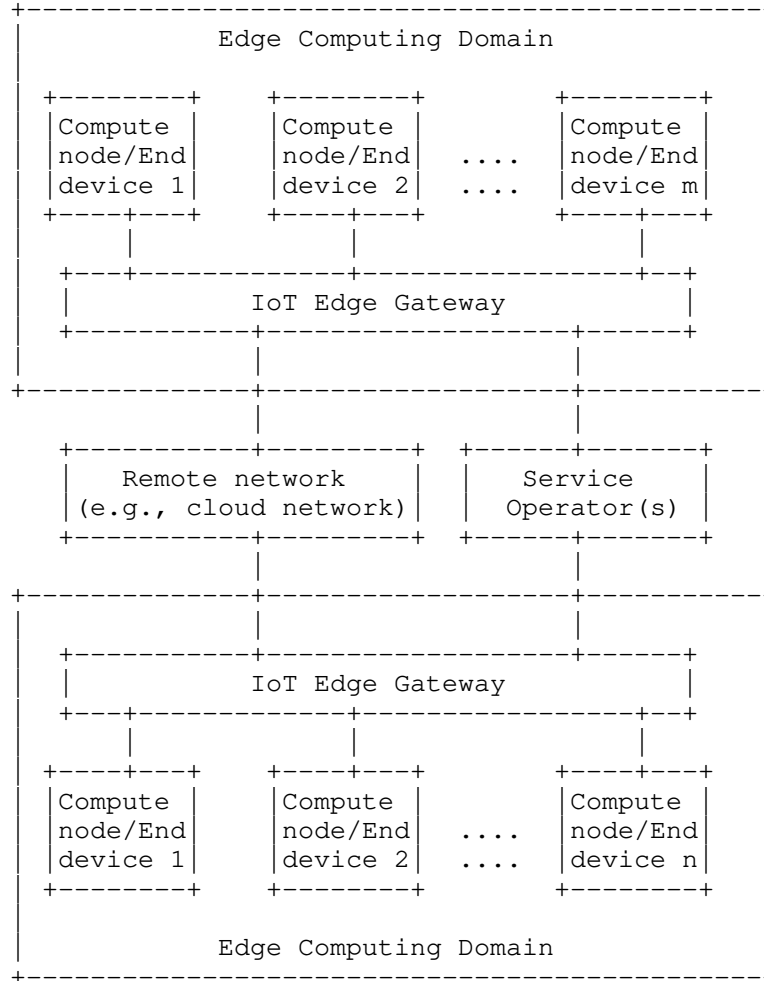


Figure 2: Example: Machine Learning over a Distributed IoT Edge Computing System

We now attempt to enumerate major edge computing domain components. They are here loosely organized into OAM (Operations, Administration, and Maintenance), functional and application components, with the understanding that the distinction between these classes may not

always be clear, depending on actual system architectures. Some representative research challenges are associated with those functions. We used input from co-authors, IRTF attendees, and some comprehensive reviews of the field ([Yousefpour], [Zhang2], [Khan]).

4.3. OAM Components

Edge computing OAM goes beyond the network-related OAM functions listed in [RFC6291]. Besides infrastructure (network, storage, and computing resources), edge computing systems can also include computing environments (for VMs, software containers, functions), IoT end devices, data, and code.

Operation-related functions include performance monitoring for service level agreement measurement; fault management and provisioning for links, nodes, compute and storage resources, platforms, and services. Administration covers network/compute/storage resources, platforms and services discovery, configuration, and planning. Discovery during normal operation (e.g., discovery of compute nodes by endpoints) would typically not be included in OAM, however in this document we will not address it separately. Management covers monitoring and diagnostics of failures, as well as means to minimize their occurrence and take corrective actions. This may include software updates management, high service availability through redundancy and multipath communication. Centralized (e.g., SDN) and decentralized management systems can be used. Finally, we arbitrarily chose to address data management as an application component, however in some systems data management may be considered to be similar to a network management function.

We further detail a few OAM components.

4.3.1. Resource Discovery and Authentication

Discovery and authentication may target platforms, infrastructure resources, such as compute, network and storage, but also other resources such as IoT end devices, sensors, data, code units, services, applications, or users interacting with the system. Broker-based solutions can be used, e.g., using an IoT gateway as a broker to discover IoT resources. More decentralized solutions can also be used in replacement or complement, e.g., CoAP enables multicast discovery of an IoT device, and CoAP service discovery enables obtaining a list of resources made available by this device [RFC7252]. Today, centralized gateway-based systems rely, for device authentication, on the installation of a secret on IoT end devices and computing devices (e.g., a device certificate stored in a hardware security module, or a combination of code and data stored in a trusted execution environment).

Related challenges include:

- * Discovery, authentication, and trust establishment between end devices, compute nodes, and platforms, with regard to concerns such as mobility, heterogeneous devices and networks, scale, multiple trust domains, constrained devices, anonymity, and traceability
- * Intermittent connectivity to the Internet, preventing relying on a third-party authority [Echeverria]
- * Resiliency to failures [Harchol], denial of service attacks, easier physical access for attackers

4.3.2. Edge Organization and Federation

In a distributed system context, once edge devices have discovered and authenticated each other, they can be organized, or self-organize, into hierarchies or clusters. The organization structure may range from centralized to peer-to-peer, or it may be closely tied with other systems. Such groups can also form federations with other edge or remote clouds.

Related challenges include:

- * Support for scaling, and enabling fault-tolerance or self-healing [Jeong]. Besides using hierarchical organization to cope with scaling, another available and possibly complementary mechanism is multicast ([RFC7390] [I-D.ietf-core-oscore-groupcomm])
- * Integration of edge computing with virtualized Radio Access Networks (Fog RAN) [I-D.bernardos-sfc-fog-ran] and with 5G access networks
- * Sharing resources in multi-vendor/operator scenarios, to optimize criteria such as profit [Anglano], resource usage, latency, or energy consumption
- * Capacity planning, placement of infrastructure nodes to minimize delay [Fan], cost, energy, etc.
- * Incentives for participation, e.g. in peer-to-peer federation schemes

4.3.3. Multi-Tenancy and Isolation

Some IoT edge computing systems make use of virtualized (compute, storage and networking) resources to address the need for secure multi-tenancy at the edge. This leads to "edge clouds" that share properties with the remote Cloud and can reuse some of its ecosystem. Virtualization function management is covered to a large extent by ETSI NFV and MEC standards activities. Projects such as [LFEDGE-EVE] further cover virtualization and its management into distributed edge computing settings.

Related challenges include:

- * Adapting cloud management platforms to the edge, to account for its distributed nature, e.g., using Conflict-free Replicated Data Types (CRDT) [Jeffery], heterogeneity and customization, e.g., using intent-based management mechanisms [Cao], and limited resources.
- * Minimizing virtual function instantiation time and resource usage

4.4. Functional Components

4.4.1. In-Network Computation

A core function of IoT edge computing is to enable local computation on a node at the network edge, e.g. processing input data from sensors, making local decisions, preprocessing data, offloading computation on behalf of a device, service, or user. Related functions include orchestrating computation (in a centralized or distributed manner) and managing applications lifecycle. Support for in-network computation may vary in term of capability, e.g., computing nodes can host virtual machines, software containers, software actors or unikernels able to run stateful or stateless code, or a rules engine providing an API to register actions in response to conditions such as IoT device ID, sensor values to check, thresholds, etc.

Edge offloading include offloading to and from an IoT device, and to and from a network node. [Cloudlets] offer an example of offloading from an end device to a network node. On the other side, oneM2M is an example of a system that allows a cloud-based IoT platform to transfer resources and tasks to a target edge node [oneM2M-TR0052]. Once transferred, the edge node can directly support IoT devices it serves with the service offloaded by the cloud (e.g. group management, location management, etc.)

QoS can be provided in some systems through the combination of network QoS (e.g., traffic engineering or wireless resource scheduling) and compute/storage resource allocations. For example, in some systems, a bandwidth manager service can be exposed to enable allocation of bandwidth to/from an edge computing application instance.

In-network computation may leverage underlying services, provided using data generated by IoT devices and access networks. Such services include end device location, radio network information, bandwidth management and congestion management (e.g., by the congestion management feature of oneM2M [oneM2M-TR0052]).

Related challenges include:

- * (Computation placement) Selecting, in a centralized or distributed/peer-to-peer manner, an appropriate compute device based on available resources, location of data input and data sinks, compute node properties, etc., and with varying goals including for example end-to-end latency, privacy, high availability, energy conservation, or network efficiency, e.g. using load balancing techniques to avoid congestion
- * Onboarding code on a platform or computing device, and invoking remote code execution, possibly as part of a distributed programming model and with respect to similar concerns of latency, privacy, etc. These operations should deal with heterogeneous compute nodes [Schafer], and may in some cases also support end devices, including IoT devices, as compute nodes [Larrea]
- * Adapting Quality of Results (QoR) for applications where a perfect result is not necessary [Li]
- * Assisted or automatic partitioning of code [I-D.sarathchandra-coin-appcentres]
- * Supporting computation across trust domains, e.g. verifying computation results
- * Support for computation mobility: relocating an instance from one compute node to another, while maintaining a given service level. Session continuity when communicating with end devices that are mobile, possibly at high speed (e.g. in vehicular scenarios). Defining lightweight execution environments for secure code mobility, e.g., using WebAssembly [Nieke]
- * Defining, managing, and verifying Service Level Agreements (SLA) for edge computing systems. Pricing is a related challenge

4.4.2. Edge Storage and Caching

Local storage or caching enable local data processing (e.g., pre-processing or analysis), as well as delayed data transfer to the cloud or delayed physical shipping. An edge node may offer local data storage (where persistence is subject to retention policies), caching, or both. Caching generally refers to temporary storage to improve performance with no persistence guarantees. An edge caching component manages data persistence, e.g., it schedules removal of data when it is no longer needed. Other related aspects include authenticating and encrypting data. Edge storage and caching can take the form of a distributed storage system.

Related challenges include

- * (Cache and data placement) Using cache positioning and data placement strategies to minimize data retrieval delay [Liu], energy consumption. Caches may be positioned in the access network infrastructure, or on end devices
- * Maintaining consistency, freshness, and privacy of stored/cached data in systems that are distributed, constrained, and dynamic (e.g. due to end devices and computing nodes churn or mobility). For example, [Mortazavi] exploits a hierarchical storage organization. Freshness-related metrics include the age of information [Yates], that captures the timeliness of information from a sender (e.g. an IoT device).

4.4.3. Northbound/Southbound Communication

An edge cloud may provide a northbound data plane or management plane interface to a remote network, e.g., a cloud, home or enterprise network. This interface does not exist in standalone (local-only) scenarios. To support such an interface when it exists, an edge computing component needs to expose an API, deal with authentication and authorization, support secure communication.

An edge cloud may provide an API or interface to local or mobile users, for example, to provide access to services and applications, or to manage data published by local/mobile devices.

Edge computing nodes communicate with IoT devices over a southbound interface, typically for data acquisition and IoT end device management.

Related challenges include:

- * Defining edge computing abstractions, such as PaaS [Yanguil], suitable for users and cloud systems to interact with edge computing systems, and dealing with interoperability issues such as data models heterogeneity.

4.4.4. Communication Brokering

A typical function of IoT edge computing is to facilitate communication with IoT end devices: for example, enable clients to register as recipients for data from devices, as well as forwarding/routing of traffic to or from IoT end devices, enabling various data discovery and redistribution patterns, e.g., north-south with clouds, east-west with other edge devices [I-D.mcbride-edge-data-discovery-overview]. Another related aspect is dispatching alerts and notifications to interested consumers both inside and outside of the edge computing domain. Protocol translation, analytics, and video transcoding may also be performed when necessary.

Communication brokering may be centralized in some systems, e.g., using a hub-and-spoke message broker, or distributed like with message buses, possibly in a layered bus approach. Distributed systems may leverage direct communication between end devices, over device-to-device links. A broker can ensure communication reliability, traceability, and in some cases transaction management.

Related challenges include:

- * Enabling secure and resilient communication between IoT end devices and remote cloud, e.g. through multipath support

4.5. Application Components

IoT edge computing can host applications such as the ones mentioned in Section 2.4. While describing components of individual applications is out of our scope, some of those applications share similar functions, such as IoT end device management, data management, described below.

4.5.1. IoT End Devices Management

IoT end device management includes managing information about the IoT devices, including their sensors, how to communicate with them, etc. Edge computing addresses the scalability challenges from the massive number of IoT end devices by separating the scalability domain into edge/local networks and remote networks. For example, in the context of the oneM2M standard, the software campaign feature enables installing, deleting, activating, and deactivating software

functions/services on a potentially large number of edge nodes [oneM2M-TR0052]. Using a dash board or a management software, a service provider issues those requests through an IoT cloud platform supporting the software campaign functionality.

Challenges listed in Section 4.3.1 may be applicable to IoT end devices management as well.

4.5.2. Data Management and Analytics

Data storage and processing at the edge is a major aspect of IoT edge computing, directly addressing high-level IoT challenges listed in Section 3. Data analysis such as performed in AI/ML tasks performed at the edge may benefit from specialized hardware support on computing nodes.

Related challenges include:

- * Addressing concerns on resource usage, security, and privacy when sharing, processing, discovering, or managing data. For example by presenting data in views composed of an aggregation of related data [Zhang]; protecting data communication between authenticated peers [Basudan]; classifying data (e.g., in terms of privacy, importance, validity, etc.); compressing and encrypting data, e.g., using homomorphic encryption to directly process encrypted data [Stanciu].
- * Other concerns on edge data discovery (e.g., streaming data, metadata, events) include siloization and lack of standard in edge environments that can be dynamic (e.g. vehicular networks) and heterogeneous [I-D.mcbride-edge-data-discovery-overview]
- * Data-driven programming models [Renart], e.g. event-based, including handling of naming and data abstractions
- * Addressing concerns such as limited resources, privacy, dynamic and heterogeneous environment, to deploy machine learning at the edge. For example, making machine learning more lightweight and distributed (e.g., to enable distributed inference at the edge), supporting shorter training time and simplified models, and supporting models that can be compressed for efficient communication [Murshed]
- * While edge computing can support IoT services independently of cloud computing, it can also be connected to cloud computing. Thus, the relationship of IoT edge computing to cloud computing, with regard to data management, is another potential challenge [ISO_TR]

4.6. Simulation and Emulation Environments

IoT Edge Computing brings new challenges to simulation and emulation tools used by researchers and developers. A varied set of applications, network, and computing technologies can coexist in a distributed system, which makes modeling difficult. Scale, mobility, and resource management are additional challenges [SimulatingFog].

Tools include simulators, where simplified application logic runs on top of a fog network model, and emulators, where actual applications can be deployed, typically in software containers, over a cloud infrastructure (e.g. Docker, Kubernetes) itself running over a network emulating network edge conditions such as variable delays, throughput and mobility events. To gain in scale, emulated and simulated systems can be used together in hybrid federation-based approaches [PseudoDynamicTesting], while to gain in realism physical devices can be interconnected with emulated systems. Examples of related work and platforms include the publicly accessible MEC sandbox work recently initiated in ETSI [ETSI_Sandbox], and open source simulators and emulators ([AdvantEDGE] emulator and tools cited in [SimulatingFog]). EdgeNet [Senel] is a globally distributed edge cloud for Internet researchers, using nodes contributed by institutions, and based on Docker for containerization and Kubernetes for deployment and node management.

5. Security Considerations

Privacy and security are drivers for the adoption of edge computing for IoT (Section 3.4). As discussed in Section 4.3.1, authentication and trust (between computing nodes, management nodes, end devices) can be challenging as scale, mobility, and heterogeneity increase. The sometimes disconnected nature of edge resources can prevent relying on a third-party authority. Distributed edge computing is exposed to issues with reliability and denial of service attacks. Personal or proprietary IoT data leakage is also a major threat, especially due to the distributed nature of the systems (Section 4.5.2).

However, edge computing also brings solutions in the security space: maintaining privacy by computing sensitive data closer to data generators is a major use case for IoT edge computing. An edge cloud can be used to take actions based on sensitive data, or anonymizing, aggregating or compressing data prior to transmitting to a remote cloud server. Edge computing communication brokering functions can also be used to secure communication between edge and cloud networks.

6. Acknowledgements

The authors would like to thank Joo-Sang Youn, Akbar Rahman, Michel Roy, Robert Gazda, Rute Sofia, Thomas Fossati, Chonggang Wang, Marie-Jose Montpetit, Carlos J. Bernardos, Milan Milenkovic, Dale Seed, JaeSeung Song and Roberto Morabito for their valuable comments and suggestions on this document.

7. Informative References

- [AdvantEDGE] "Mobile Edge Emulation Platform", Source Code Repository , 2020, <<https://github.com/InterDigitalInc/AdvantEDGE>>.
- [Anglano] Anglano, C., Canonico, M., Castagno, P., Guazzone, M., and M. Sereno, "A game-theoretic approach to coalition formation in fog provider federations", IEEE Third International Conference on Fog and Mobile Edge Computing (FMEC), pages 123-130 , 2018.
- [Ashton] Ashton, K., "That Internet of Things thing", RFID J. vol. 22, no. 7, pp. 97-114 , 2009.
- [Basudan] Basudan, S., Lin, X., and K. Sankaranarayanan, "A privacy-preserving vehicular crowdsensing-based road surface condition monitoring system using fog computing", IEEE Internet of Things Journal, 4(3):772-782 , 2017.
- [Botta] Botta, A., Donato, W., Persico, V., and A. Pescapé, "Integration of Cloud Computing and Internet of Things: A survey", Future Gener. Comput. Syst., vol. 56, pp. 684-700 , 2016.
- [Cao] Cao, L., Merican, A., Zad Tootaghaj, D., Ahmed, F., Sharma, P., and V. Saxena, "ECaaS: A Management Framework of Edge Container as a Service for Business Workload", 4th International Workshop on Edge Systems, Analytics and Networking , 2021, <<https://doi.org/10.1145/3434770.3459741>>.
- [Chen] Baotong Chen, ., Jiafu Wan, ., Antonio Celesti, ., Di Li, ., Haider Abbas, ., and . Qin Zhang, "Edge computing in IoT-based manufacturing", IEEE Communications Magazine , 2018, <<https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8466364>>.

- [Chiang] Chiang, M. and T. Zhang, "Fog and IoT: An overview of research opportunities", IEEE Internet Things J., vol. 3, no. 6, pp. 854-864 , 2016.
- [Cloudlets] Mahadev Satyanarayanan, ., Paramvir Bahl, ., Ramón Cáceres, ., and . Nigel Davies, "The Case for VM-Based Cloudlets in Mobile Computing", IEEE Pervasive Computing , 2009, <<https://ieeexplore.ieee.org/document/5280678>>.
- [Echeverria] Echeverria, S., Klinedinst, D., Williams, K., and G. A Lewis, "Establishing trusted identities in disconnected edge environments", IEEE/ACM Symposium Edge Computing (SEC), pages 51-63. , 2016.
- [ENERGY] Beckel, C., Sadamori, L., Staake, T., and S. Santini, "Revealing Household Characteristics from Smart Meter Data", Energy, vol. 78, pp. 397-410 , 2014.
- [ETSI_MEC_01] ETSI, ., "Multi-access Edge Computing (MEC); Terminology", ETSI GS 001 , 2019, <https://www.etsi.org/deliver/etsi_gs/MEC/001_099/001/02.01.01_60/gs_MEC001v020101p.pdf>.
- [ETSI_MEC_03] ETSI, ., "Mobile Edge Computing (MEC); Framework and Reference Architecture", ETSI GS 003 , 2019, <https://www.etsi.org/deliver/etsi_gs/MEC/001_099/003/02.01.01_60/gs_MEC003v020101p.pdf>.
- [ETSI_Sandbox] "Multi-access Edge Computing (MEC) MEC Sandbox Work Item", Portal , 2020, <https://portal.etsi.org/webapp/WorkProgram/Report_WorkItem.asp?WKI_ID=57671>.
- [Fan] Fan, Q. and N. Ansari, "Cost aware cloudlet placement for big data processing at the edge", IEEE International Conference on Communications (ICC), pages 1-6 , 2017.
- [Harchol] Harchol, Y., Mushtaq, A., McCauley, J., Panda, A., and S. Shenker, "Cessna: Resilient edge-computing", Workshop on Mobile Edge Communications, pages 1-6. ACM , 2018.
- [I-D-defoy-t2trg-iot-edge-computing-background] de Foy, X., Hong, J., Hong, Y., Kovatsch, M., Schooler, E., and D. Kutscher, "Machine learning at the network

edge: A survey", draft-defoy-t2trg-iot-edge-computing-background-00 , 2020, <<http://www.ietf.org/internet-drafts/draft-defoy-t2trg-iot-edge-computing-background-00.txt>>.

[I-D.bernardos-sfc-fog-ran]

Bernardos, C. J. and A. Mourad, "Service Function Chaining Use Cases in Fog RAN", Work in Progress, Internet-Draft, draft-bernardos-sfc-fog-ran-10, 22 October 2021, <<https://datatracker.ietf.org/doc/html/draft-bernardos-sfc-fog-ran-10>>.

[I-D.ietf-core-oscore-groupcomm]

Tiloca, M., Selander, G., Palombini, F., Mattsson, J. P., and J. Park, "Group OSCORE - Secure Group Communication for CoAP", Work in Progress, Internet-Draft, draft-ietf-core-oscore-groupcomm-13, 25 October 2021, <<https://datatracker.ietf.org/doc/html/draft-ietf-core-oscore-groupcomm-13>>.

[I-D.mcbride-edge-data-discovery-overview]

McBride, M., Kutscher, D., Schooler, E., Bernardos, C. J., Lopez, D. R., and X. D. Foy, "Edge Data Discovery for COIN", Work in Progress, Internet-Draft, draft-mcbride-edge-data-discovery-overview-05, 1 November 2020, <<https://datatracker.ietf.org/doc/html/draft-mcbride-edge-data-discovery-overview-05>>.

[I-D.sarathchandra-coin-appcentres]

Trossen, D., Sarathchandra, C., and M. Boniface, "In-Network Computing for App-Centric Micro-Services", Work in Progress, Internet-Draft, draft-sarathchandra-coin-appcentres-04, 26 January 2021, <<https://datatracker.ietf.org/doc/html/draft-sarathchandra-coin-appcentres-04>>.

[ISO_TR] "Information Technology - Cloud Computing - Edge Computing Landscape", ISO/IEC TR 23188 , 2018.

[Jeffery] Jeffery, A., Howard, H., and R. Mortier, "Rearchitecting Kubernetes for the Edge", 4th International Workshop on Edge Systems, Analytics and Networking , 2021, <<https://dl.acm.org/doi/10.1145/3434770.3459730>>.

[Jeong] Jeong, T., Chung, J., Hong, J.W., and S. Ha, "Towards a distributed computing framework for fog", IEEE Fog World Congress (FWC), pages 1-6 , 2017.

- [Jones] David Jones, ., Chris Snider, ., Aydin Nassehi, ., Jason Yon, ., and . Ben Hicks, "Characterising the Digital Twin: A systematic literature review", CIRP Journal of Manufacturing Science and Technology , 2020, <<https://www.sciencedirect.com/science/article/pii/S1755581720300110>>.
- [Kelly] Kelly, R., "Internet of Things Data to Top 1.6 Zettabytes by 2022", 2015, <<https://campustechnology.com/articles/2015/04/15/internet-of-things-data-to-top-1-6-zettabytes-by-2020.aspx>>.
- [Khan] Khan, L.U., Yaqoob, I., Tran, N.H., Kazmi, S.M.A., Dang, T.N., and C.S. Hong, "Edge Computing Enabled Smart Cities: A Comprehensive Survey", arXiv:1909.08747 , 2019.
- [Larrea] Larrea, J. and A. Barbalace, "The serverkernel operating system", Third ACM International Workshop on Edge Systems, Analytics and Networking , 2020, <<https://core.ac.uk/reader/327124532>>.
- [LFEDGE-EVE] Linux Foundation, ., "Project Edge Virtualization Engine (EVE)", Portal , 2020, <<https://www.lfedge.org/projects/eve>>.
- [Li] Li, Y., Chen, Y., Lan, T., and G. Venkataramani, "Mobiqor: Pushing the envelope of mobile edge computing via quality-of-result optimization", IEEE 37th International Conference on Distributed Computing Systems (ICDCS), pages 1261-1270 , 2017.
- [Lin] Lin, J., Yu, W., Zhang, N., Yang, X., Zhang, H., and W. Zhao, "A survey on Internet of Things: Architecture, enabling technologies, security and privacy, and applications", IEEE Internet of Things J., vol. 4, no. 5, pp. 1125-1142 , 2017.
- [Liu] Liu, J., Bai, B., Zhang, J., and K.B. Letaief, "Cache placement in fog-rans: From centralized to distributed algorithms", IEEE Transactions on Wireless Communications, 16(11):7039-7051 , 2017.
- [Mahadev] Satyanarayanan, M., "The Emergence of Edge Computing", Computer, vol. 50, no. 1, pp. 30-39 , 2017.

- [Mortazavi] Hossein Mortazavi, S., Balasubramanian, B., de Lara, E., and S.P. Narayanan, "Toward Session Consistency for the Edge", USENIX, Workshop on Hot Topics in Edge Computing (HotEdge 18) , 2018, <<https://www.usenix.org/conference/hotedge18/presentation/mortazavi>>.
- [Murshed] Murshed, M., Murphy, C., Hou, D., Khan, N., Ananthanarayanan, G., and F. Hussain, "Machine learning at the network edge: A survey", arXiv:1908.00080 , 2019.
- [Nieke] Nieke, M., Almstedt, L., and R. Kapitza, "Edgedancer: Secure Mobile WebAssembly Services on the Edge", 4th International Workshop on Edge Systems, Analytics and Networking , 2021, <<https://doi.org/10.1145/3434770.3459731>>.
- [NIST] Mell, P. and T. Grance, "The NIST definition of Cloud Computing", Natl. Inst. Stand. Technol, vol. 53, no. 6, p. 50 , 2009.
- [NVIDIA] Grzywaczewski, A., "Training AI for Self-Driving Vehicles: the Challenge of Scale", NVIDIA Developer Blog , 2017, <<https://devblogs.nvidia.com/training-self-driving-vehicles-challenge-scale/>>.
- [oneM2M-TR0001] Mladin, C., "TR 0001, Use Cases Collection", oneM2M , October 2018, <<https://member.onem2m.org/Application/documentapp/downloadLatestRevision/default.aspx?docID=28153>>.
- [oneM2M-TR0018] Lu, C. and M. Jiang, "TR 0018, Industrial Domain Enablement", oneM2M , February 2019, <<https://member.onem2m.org/Application/documentapp/downloadLatestRevision/default.aspx?docID=29334>>.
- [oneM2M-TR0026] Yamamoto, K., Mladin, C., and V. Kueh, "TR 0026, Vehicular Domain Enablement", oneM2M , January 2020, <<https://member.onem2m.org/Application/documentapp/downloadLatestRevision/default.aspx?docID=31410>>.

- [oneM2M-TR0052] Yamamoto, K. and C. Mladin, "TR 0052, Study on Edge and Fog Computing in oneM2M systems", oneM2M , September 2020, <<https://member.onem2m.org/Application/documentapp/downloadLatestRevision/default.aspx?docID=32633>>.
- [oneM2M-TS0002] He, S., "TS 0002, Requirements", oneM2M , February 2019, <<https://member.onem2m.org/Application/documentapp/downloadLatestRevision/default.aspx?docID=29274>>.
- [OpenFog] "OpenFog Reference Architecture for Fog Computing", OpenFog Consortium , 2017.
- [PseudoDynamicTesting] Ficco, M., Esposito, C., Xiang, Y., and F. Palmieri, "Pseudo-Dynamic Testing of Realistic Edge-Fog Cloud Ecosystems", IEEE Communications Magazine, Nov. 2017 , 2017.
- [Renart] Renart, E.G., Diaz-Montes, J., and M. Parashar, "Data-driven stream processing at the edge", IEEE 1st International Conference on Fog and Edge Computing (ICFEC), pages 31-40 , 2017.
- [RFC6291] Andersson, L., van Helvoort, H., Bonica, R., Romascanu, D., and S. Mansfield, "Guidelines for the Use of the "OAM" Acronym in the IETF", BCP 161, RFC 6291, DOI 10.17487/RFC6291, June 2011, <<https://doi.org/10.17487/RFC6291>>.
- [RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <<https://doi.org/10.17487/RFC7252>>.
- [RFC7390] Rahman, A., Ed. and E. Dijk, Ed., "Group Communication for the Constrained Application Protocol (CoAP)", RFC 7390, DOI 10.17487/RFC7390, October 2014, <<https://doi.org/10.17487/RFC7390>>.
- [Schafer] Schafer, D., Edinger, J., VanSyckel, S., Paluska, J.M., and C. Becker, "Tasklets: Overcoming Heterogeneity in Distributed Computing Systems", IEEE 36th International Conference on Distributed Computing Systems Workshops (ICDCSW), Nara, pp. 156-161 , 2016.

- [Senel] Senel, B., Mouchet, M., Cappos, J., Fourmaux, O., Friedman, T., and R. McGeer, "EdgeNet: A Multi-Tenant and Multi-Provider Edge Cloud", 4th International Workshop on Edge Systems, Analytics and Networking , 2021, <<https://dl.acm.org/doi/pdf/10.1145/3434770.3459737>>.
- [Shi] Shi, W., Cao, J., Zhang, Q., Li, Y., and L. Xu, "Edge computing: vision and challenges", IEEE Internet of Things J., vol. 3, no. 5, pp. 637-646 , 2016.
- [SimulatingFog] Svorobej, S. and . al, "Simulating Fog and Edge Computing Scenarios: An Overview and Research Challenges", MPDI Future Internet 2019 , 2019.
- [Stanciu] Stanciu, V., van Steen, M., Dobre, C., and A. Peter, "Privacy-Preserving Crowd-Monitoring Using Bloom Filters and Homomorphic Encryption", 4th International Workshop on Edge Systems, Analytics and Networking , 2021, <<https://dl.acm.org/doi/10.1145/3434770.3459735>>.
- [Weiner] Weiner, M., Jorgovanovic, M., Sahai, A., and B. Nikolic, "Design of a low-latency, high-reliability wireless communication system for control applications", IEEE Int. Conf. Commun. (ICC), Sydney, NSW, Australia, pp. 3829-3835 , 2014.
- [Yangui] Yangui, S., Ravindran, P., Bibani, O., H Glitho, R., Ben Hadj-Alouane, N., Morrow, M.J., and P.A. Polakos, "A platform as-a-service for hybrid cloud/fog environments", IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN), pages 1-7 , 2016.
- [Yates] Yates, R.D. and S.K. Kaul, "The Age of Information: Real-Time Status Updating by Multiple Sources", IEEE Transactions on Information Theory, vol. 65, no. 3, pp. 1807-1827 , 2019.
- [Yousefpour] Yousefpour, A., Fung, C., Nguyen, T., Kadiyala, K., Jalali, F., Niakanlahiji, A., Kong, J., and J.P. Jue, "All one needs to know about fog computing and related edge computing paradigms: A complete survey", Journal of Systems Architecture, vol. 98, pp. 289-330 , 2019.

- [Zhang] Zhang, Q., Zhang, X., Zhang, Q., Shi, W., and H. Zhong, "Firework: Big data sharing and processing in collaborative edge environment", Fourth IEEE Workshop on Hot Topics in Web Systems and Technologies (HotWeb), pages 20-25 , 2016.
- [Zhang2] Zhang, J., Chen, B., Zhao, Y., Cheng, X., and F. Hu, "Data Security and Privacy-Preserving in Edge Computing Paradigm: Survey and Open Issues", IEEE Access, vol. 6, pp. 18209-18237 , 2018.
- [_60802] IEC/IEEE, ., "Use Cases IEC/IEEE 60802 V1.3", IEC/IEEE 60802 , 2018, <<http://www.ieee802.org/1/files/public/docs2018/60802-industrial-use-cases-0818-v13.pdf>>.

Authors' Addresses

Jungha Hong
ETRI
218 Gajeong-ro, Yuseung-Gu
Daejeon
34129
Republic of Korea

Email: jhong@etri.re.kr

Yong-Geun Hong
Daejeon University
62 Daehak-ro, Dong-gu
Daejeon
300716
Republic of Korea

Email: yonggeun.hong@gmail.com

Xavier de Foy
InterDigital Communications, LLC
1000 Sherbrooke West
Montreal H3A 3G4
Canada

Email: xavier.defoy@interdigital.com

Matthias Kovatsch
Huawei Technologies Duesseldorf GmbH
Riesstr. 25 C // 3.OG
80992 Munich
Germany

Email: ietf@kovatsch.net

Eve Schooler
Intel
2200 Mission College Blvd.
Santa Clara, CA, 95054-1537
United States of America

Email: eve.m.schooler@intel.com

Dirk Kutscher
University of Applied Sciences Emden/Leer
Constantiaplatz 4
26723 Emden
Germany

Email: ietf@dkutscher.net

T2TRG Research Group
Internet-Draft
Intended status: Informational
Expires: 7 August 2022

M. Richardson
Sandelman Software Works
3 February 2022

A Taxonomy of operational security considerations for manufacturer
installed keys and Trust Anchors
draft-richardson-t2trg-idevid-considerations-06

Abstract

This document provides a taxonomy of methods used by manufacturers of silicon and devices to secure private keys and public trust anchors. This deals with two related activities: how trust anchors and private keys are installed into devices during manufacturing, and how the related manufacturer held private keys are secured against disclosure.

This document does not evaluate the different mechanisms, but rather just serves to name them in a consistent manner in order to aid in communication.

RFCEDITOR: please remove this paragraph. This work is occurring in <https://github.com/mcr/idevid-security-considerations>

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 7 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
1.1. Terminology	4
2. Applicability Model	5
2.1. A reference manufacturing/boot process	6
3. Types of Trust Anchors	7
3.1. Secured First Boot Trust Anchor	8
3.2. Software Update Trust Anchor	8
3.3. Trusted Application Manager anchor	9
3.4. Public WebPKI anchors	9
3.5. DNSSEC root	9
3.6. What else?	10
4. Types of Identities	10
4.1. Manufacturer installed IDevID certificates	10
4.1.1. Operational Considerations for Manufacturer IDevID Public Key Infrastructure	11
4.1.2. Key Generation process	11
5. Public Key Infrastructures (PKI)	14
5.1. Number of levels of certification authorities	15
5.2. Protection of CA private keys	17
5.3. Supporting provisioned anchors in devices	17
6. Evaluation Questions	18
6.1. Integrity and Privacy of on-device data	18
6.2. Integrity and Privacy of device identify infrastructure	19
6.3. Integrity and Privacy of included trust anchors	19
7. Privacy Considerations	20
8. Security Considerations	20
9. IANA Considerations	20
10. Acknowledgements	20
11. Changelog	20
12. References	20
12.1. Normative References	20
12.2. Informative References	20
Author's Address	25

1. Introduction

An increasing number of protocols derive a significant part of their security by using trust anchors [RFC4949] that are installed by manufacturers. Disclosure of the list of trust anchors does not usually cause a problem, but changing them in any way does. This includes adding, replacing or deleting anchors. [RFC6024] deals with how trust anchor stores are managed, while this document deals with how the associated PKI which is anchor is managed.

Many protocols also leverage manufacturer installed identities. These identities are usually in the form of [ieee802-1AR] Initial Device Identity certificates (IDevID). The identity has two components: a private key that must remain under the strict control of a trusted part of the device, and a public part (the certificate), which (ignoring, for the moment, personal privacy concerns) may be freely disclosed.

There also situations where identities are tied up in the provision of symmetric shared secrets. A common example is the SIM card ([_3GPP.51.011]), it now comes as a virtual SIM, but which is usually not provisioned at the factory. The provision of an initial, per-device default password also falls into the category of symmetric shared secret.

It is further not unusual for many devices (particularly smartphones) to also have one or more group identity keys. This is used, for instance, in [fidotechnote] to make claims about being a particular model of phone (see [I-D.richardson-rats-usecases]). The key pair that does this is loaded into large batches of phones for privacy reasons.

The trust anchors are used for a variety of purposes. Trust anchors are used to verify:

- * the signature on a software update (as per [I-D.ietf-suit-architecture]),
- * a TLS Server Certificate, such as when setting up an HTTPS connection,
- * the [RFC8366] format voucher that provides proof of an ownership change.

Device identity keys are used when performing enrollment requests (in [RFC8995], and in some uses of [I-D.ietf-emu-eap-noob]). The device identity certificate is also used to sign Evidence by an Attesting Environment (see [I-D.ietf-rats-architecture]).

These security artifacts are used to anchor other chains of information: an EAT Claim as to the version of software/firmware running on a device ([I-D.birkholz-suit-coswid-manifest]), an EAT claim about legitimate network activity (via [I-D.birkholz-rats-mud], or embedded in the IDevID in [RFC8520]).

Known software versions lead directly to vendor/distributor signed Software Bill of Materials (SBOM), such as those described by [I-D.ietf-sacm-coswid] and the NTIA/SBOM work [ntiasbom] and CISQ/OMG SBOM work underway [cisqsbom].

In order to manage risks and assess vulnerabilities in a Supply Chain, it is necessary to determine a degree of trustworthiness in each device. A device may mislead audit systems as to its provenance, about its software load or even about what kind of device it is (see [RFC7168] for a humorous example).

In order to properly assess the security of a Supply Chain it is necessary to understand the kinds and severity of the threats which a device has been designed to resist. To do this, it is necessary to understand the ways in which the different trust anchors and identities are initially provisioned, are protected, and are updated.

To do this, this document details the different trust anchors (TrAnc) and identities (IDs) found in typical devices. The privacy and integrity of the TrAncs and IDs is often provided by a different, superior artifact. This relationship is examined.

While many might desire to assign numerical values to different mitigation techniques in order to be able to rank them, this document does not attempt to do that, as there are too many other (mostly human) factors that would come into play. Such an effort is more properly in the purview of a formal ISO9001 process such as ISO14001.

1.1. Terminology

This document is not a standards track document, and it does not make use of formal requirements language.

This section will be expanded to include needed terminology as required.

The words Trust Anchor are contracted to TrAnc rather than TA, in order not to confuse with [I-D.ietf-teep-architecture]'s "Trusted Application".

This document defines a number of hyphenated terms, and they are summarized here:

device-generated: a private or symmetric key which is generated on the device

infrastructure-generated: a private or symmetric key which is generated by some system, likely located at the factory that built the device

mechanically-installed: when a key or certificate is programmed into non-volatile storage by an out-of-band mechanism such as JTAG [JTAG]

mechanically-transferred: when a key or certificate is transferred into a system via private interface, such as serial console, JTAG managed mailbox, or other physically private interface

network-transferred: when a key or certificate is transferred into a system using a network interface which would be available after the device has shipped. This applies even if the network is physically attached using a bed-of-nails [BedOfNails].

device/infrastructure-co-generated: when a private or symmetric key is derived from a secret previously synchronized between the silicon vendor and the factory using a common algorithm.

2. Applicability Model

There is a wide variety of devices to which this analysis can apply. (See [I-D.bormann-lwig-7228bis].) This document will use a J-group processor as a sample. This class is sufficiently large to experience complex issues among multiple CPUs, packages and operating systems, but at the same time, small enough that this class is often deployed in single-purpose IoT-like uses. Devices in this class often have Secure Enclaves (such as the "Grapeboard"), and can include silicon manufacturer controlled processors in the boot process (the Raspberry PI boots under control of the GPU).

Almost all larger systems (servers, laptops, desktops) include a Baseboard Management Controller (BMC), which ranges from a M-Group Class 3 MCU, to a J-Group Class 10 CPU (see, for instance [openbmc] which uses a Linux kernel and system inside the BMC). As the BMC usually has complete access to the main CPU's memory, I/O hardware and disk, the boot path security of such a system needs to be understood first as being about the security of the BMC.

2.1. A reference manufacturing/boot process

In order to provide for immutability and privacy of the critical TrAnc and IDs, many CPU manufacturers will provide for some kind of private memory area which is only accessible when the CPU is in certain privileged states. See the Terminology section of [I-D.ietf-teep-architecture], notably TEE, REE, and TAM, and also section 4, Architecture.

The private memory that is important is usually non-volatile and rather small. It may be located inside the CPU silicon die, or it may be located externally. If the memory is external, then it is usually encrypted by a hardware mechanism on the CPU, with only the key kept inside the CPU.

The entire mechanism may be external to the CPU in the form of a hardware-TPM module, or it may be entirely internal to the CPU in the form of a firmware-TPM. It may use a custom interface to the rest of the system, or it may implement the TPM 1.2 or TPM 2.0 specifications. Those details are important to performing a full evaluation, but do not matter much to this model (see initial-enclave-location below).

During the manufacturing process, once the components have been soldered to the board, the system is usually put through a system-level test. This is often done as a "bed-of-nails" test [BedOfNails], where the board has key points attached mechanically to a test system. A [JTAG] process tests the System Under Test, and then initializes some firmware into the still empty flash storage.

It is now common for a factory test image to be loaded first: this image will include code to initialize the private memory key described above, and will include a first-stage bootloader and some kind of (primitive) Trusted Application Manager (TAM). (The TAM is a piece of software that lives within the trusted execution environment.)

Embedded in the stage one bootloader will be a Trust Anchor that is able to verify the second-stage bootloader image.

After the system has undergone testing, the factory test image is erased, leaving the first-stage bootloader. One or more second-stage bootloader images are installed. The production image may be installed at that time, or if the second-stage bootloader is able to install it over the network, it may be done that way instead.

There are many variations of the above process, and this section is not attempting to be prescriptive, but to provide enough illustration to motivate subsequent terminology.

The process may be entirely automated, or it may be entirely driven by humans working in the factory, or a combination of the above.

These steps may all occur on an access-controlled assembly line, or the system boards may be shipped from one place to another (maybe another country) before undergoing testing.

Some systems are intended to be shipped in a tamper-proof state, but it is usually not desirable that bed-of-nails testing be possible without tampering, so the initialization process is usually done prior to rendering the system tamper-proof. An example of a one-way tamper-proof, weather resistant treatment might to mount the system board in a case and fill the case with resin.

Quality control testing may be done prior to as well as after the application of tamper-proofing, as systems which do not pass inspection may be reworked to fix flaws, and this should ideally be impossible once the system has been made tamper-proof.

3. Types of Trust Anchors

Trust Anchors (TrAnc) are fundamentally public keys with authorizations implicitly attached through the code that references them.

They are used to validate other digitally signed artifacts. Typically, these are chains of PKIX certificates leading to an End-Entity certificate (EE).

The chains are usually presented as part of an externally provided object, with the term "externally" to be understood as being as close as untrusted flash, to as far as objects retrieved over a network.

There is no requirement that there be any chain at all: the trust anchor can be used to validate a signature over a target object directly.

The trust anchors are often stored in the form of self-signed certificates. The self-signature does not offer any cryptographic assurance, but it does provide a form of error detection, providing verification against non-malicious forms of data corruption. If storage is at a premium (such as inside-CPU non-volatile storage) then only the public key itself need to be stored. For a 256-bit ECDSA key, this is 32 bytes of space.

When evaluating the degree of trust for each trust anchor there are four aspects that need to be determined:

- * can the trust anchor be replaced or modified?
- * can additional trust anchors be added?
- * can trust anchors be removed?
- * how is the private key associated with the trust anchor, maintained by the manufacturer, maintained?

The first three things are device specific properties of how the integrity of the trust anchor is maintained.

The fourth property has nothing to do with the device, but has to do with the reputation and care of the entity that maintains the private key.

Different anchors have different authorizations associated with them.

These are:

3.1. Secured First Boot Trust Anchor

This anchor is part of the first-stage boot loader, and it is used to validate a second-stage bootloader which may be stored in external flash. This is called the initial software trust anchor.

3.2. Software Update Trust Anchor

This anchor is used to validate the main application (or operating system) load for the device.

It can be stored in a number of places. First, it may be identical to the Secure Boot Trust Anchor.

Second, it may be stored in the second-stage bootloader, and therefore its integrity is protected by the Secured First Boot Trust Anchor.

Third, it may be stored in the application code itself, where the application validates updates to the application directly (update in place), or via a double-buffer arrangement. The initial (factory) load of the application code initializes the trust arrangement.

In this situation the application code is not in a secured boot situation, as the second-stage bootloader does not validate the application/operating system before starting it, but it may still provide measured boot mechanism.

3.3. Trusted Application Manager anchor

This anchor is the secure key for the [I-D.ietf-teep-architecture] Trusted Application Manager (TAM). Code which is signed by this anchor will be given execution privileges as described by the manifest which accompanies the code. This privilege may include updating anchors.

3.4. Public WebPKI anchors

These anchors are used to verify HTTPS certificates from web sites. These anchors are typically distributed as part of desktop browsers, and via desktop operating systems.

The exact set of these anchors is not precisely defined: it is usually determined by the browser vendor (e.g., Mozilla, Google, Apple, Safari, Microsoft), or the operating system vendor (e.g., Apple, Google, Microsoft, Ubuntu). In most cases these vendors look to the CA/Browser Forum [CABFORUM] for inclusion criteria.

3.5. DNSSEC root

This anchor is part of the DNS Security extensions. It provides an anchor for securing DNS lookups. Secure DNS lookups may be important in order to get access to software updates. This anchor is now scheduled to change approximately every 3 years, with the new key announced several years before it is used, making it possible to embed keys that will be valid for up to five years.

This trust anchor is typically part of the application/operating system code and is usually updated by the manufacturer when they do updates. However, a system that is connected to the Internet may update the DNSSEC anchor itself through the mechanism described in [RFC5011].

There are concerns that there may be a chicken and egg situation for devices that have remained in a powered off state (or disconnected from the Internet) for some period of years. That upon being reconnected, that the device would be unable to do DNSSEC validation. This failure would result in them being unable to obtain operating system updates that would then include the updates to the DNSSEC key.

3.6. What else?

TBD?

4. Types of Identities

Identities are installed during manufacturing time for a variety of purposes.

Identities require some private component. Asymmetric identities (e.g., RSA, ECDSA, EdDSA systems) require a corresponding public component, usually in the form of a certificate signed by a trusted third party.

This certificate associates the identity with attributes.

The process of making this coordinated key pair and then installing it into the device is called identity provisioning.

4.1. Manufacturer installed IDevID certificates

[ieee802-1AR] defines a category of certificates that are installed by the manufacturer, which contain at the least, a device unique serial number.

A number of protocols depend upon this certificate.

- * [RFC8572] and [RFC8995] introduce mechanisms for new devices (called pledges) to be onboarded into a network without intervention from an expert operator. A number of derived protocols such as [I-D.ietf-anima-brski-async-enroll], [I-D.ietf-anima-constrained-voucher], [I-D.richardson-anima-voucher-delegation], [I-D.friel-anima-brski-cloud] extend this in a number of ways.
- * [I-D.ietf-rats-architecture] depends upon a key provisioned into the Attesting Environment to sign Evidence.
- * [I-D.ietf-suit-architecture] may depend upon a key provisioned into the device in order to decrypt software updates. Both symmetric and asymmetric keys are possible. In both cases, the decrypt operation depends upon the device having access to a private key provisioned in advance. The IDevID can be used for this if algorithm choices permit. ECDSA keys do not directly support encryption in the same way that RSA does, for instance, but the addition of ECIES can solve this. There may be other legal considerations why the IDevID might not be used, and a second key provisioned.

* TBD

4.1.1. Operational Considerations for Manufacturer IDevID Public Key Infrastructure

The manufacturer has the responsibility to provision a key pair into each device as part of the manufacturing process. There are a variety of mechanisms to accomplish this, which this document will overview.

There are three fundamental ways to generate IDevID certificates for devices:

1. generating a private key on the device, creating a Certificate Signing Request (or equivalent), and then returning a certificate to the device.
2. generating a private key outside the device, signing the certificate, and then installing both into the device.
3. deriving the private key from a previously installed secret seed, that is shared with only the manufacturer.

There is a fourth situation where the IDevID is provided as part of a Trusted Platform Module (TPM), in which case the TPM vendor may be making the same tradeoffs.

The document [I-D.moskowitz-ecdsa-pki] provides some practical instructions on setting up a reference implementation for ECDSA keys using a three-tier mechanism.

4.1.2. Key Generation process

4.1.2.1. On-device private key generation

Generating the key on-device has the advantage that the private key never leaves the device. The disadvantage is that the device may not have a verified random number generator. [factoringrsa] is an example of a successful attack on this scenario.

There are a number of options of how to get the public key securely from the device to the certification authority.

This transmission must be done in an integral manner, and must be securely associated with the assigned serial number. The serial number goes into the certificate, and the resulting certificate needs to be loaded into the manufacturer's asset database.

One way to do the transmission is during a factory Bed of Nails test (see [BedOfNails]) or Boundary Scan. When done via a physical connection like this, then this is referred to as a `_device-generated_` / `_mechanically-transferred_` method.

There are other ways that could be used where a certificate signing request is sent over a special network channel when the device is powered up in the factory. This is referred to as the `_device-generated_` / `_network-transferred_` method.

Regardless of how the certificate signing request is sent from the device to the factory, and how the certificate is returned to the device, a concern from production line managers is that the assembly line may have to wait for the certification authority to respond with the certificate.

After the key generation, the device needs to set a flag such that it no longer will generate a new key / will accept a new IDevID via the factory connection. This may be a software setting, or could be as dramatic as blowing a fuse.

The risk is that if an attacker with physical access is able to put the device back into an unconfigured mode, then the attacker may be able to substitute a new certificate into the device. It is difficult to construct a rationale for doing this, unless the network initialization also permits an attacker to load or replace trust anchors at the same time.

Devices are typically constructed in a fashion such that the device is unable to ever disclose the private key via an external interface. This is usually done using a secure-enclave provided by the CPU architecture in combination with on-chip non-volatile memory.

4.1.2.2. Off-device private key generation

Generating the key off-device has the advantage that the randomness of the private key can be better analyzed. As the private key is available to the manufacturing infrastructure, the authenticity of the public key is well known ahead of time.

If the device does not come with a serial number in silicon, then one should be assigned and placed into a certificate. The private key and certificate could be programmed into the device along with the initial bootloader firmware in a single step.

Aside from the change of origin for the randomness, a major advantage of this mechanism is that it can be done with a single write to the flash. The entire firmware of the device, including configuration of

trust anchors and private keys can be loaded in a single write pass. Given some pipelining of the generation of the keys and the creation of certificates, it may be possible to install unique identities without taking any additional time.

The major downside to generating the private key off-device is that it could be seen by the manufacturing infrastructure. It could be compromised by humans in the factory, or the equipment could be compromised. The use of this method increases the value of attacking the manufacturing infrastructure.

If private keys are generated by the manufacturing plant, and are immediately installed, but never stored, then the window in which an attacker can gain access to the private key is immensely reduced.

As in the previous case, the transfer may be done via physical interfaces such as bed-of-nails, giving the `_infrastructure-generated_` / `_mechanically-transferred_` method.

There is also the possibility of having a `_infrastructure-generated_` / `_network-transferred_` key. There is a support for "server-generated" keys in [RFC7030], [RFC8894], and [RFC4210]. All methods strongly recommend encrypting the private key for transfer. This is difficult to comply with here as there is not yet any private key material in the device, so in many cases it will not be possible to encrypt the private key.

4.1.2.3. Key setup based on 256 bit secret seed

A hybrid of the previous two methods leverages a symmetric key that is often provided by a silicon vendor to OEM manufacturers.

Each CPU (or a Trusted Execution Environment [I-D.ietf-tee-architecture], or a TPM) is provisioned at fabrication time with a unique, secret seed, usually at least 256 bits in size.

This value is revealed to the OEM board manufacturer only via a secure channel. Upon first boot, the system (probably within a TEE, or within a TPM) will generate a key pair using the seed to initialize a Pseudo-Random-Number-Generator (PRNG). The OEM, in a separate system, will initialize the same PRNG and generate the same key pair. The OEM then derives the public key part, signs it and turns it into a certificate. The private part is then destroyed, ideally never stored or seen by anyone. The certificate (being public information) is placed into a database, in some cases it is loaded by the device as its IDevID certificate, in other cases, it is retrieved during the onboarding process based upon a unique serial number asserted by the device.

This method appears to have all of the downsides of the previous two methods: the device must correctly derive its own private key, and the OEM has access to the private key, making it also vulnerable. The secret seed must be created in a secure way and it must also be communicated securely.

There are some advantages to the OEM however: the major one is that the problem of securely communicating with the device is outsourced to the silicon vendor. The private keys and certificates may be calculated by the OEM asynchronously to the manufacturing process, either done in batches in advance of actual manufacturing, or on demand when an IDevID is demanded. Doing the processing in this way permits the key derivation system to be completely disconnected from any network, and requires placing very little trust in the system assembly factory. Operational security such as often incorrectly presented fictionalized stories of a "mainframe" system to which only physical access is permitted begins to become realistic. That trust has been replaced with a heightened trust placed in the silicon (integrated circuit) fabrication facility.

The downsides of this method to the OEM are: they must be supplied by a trusted silicon fabrication system, which must communicate the set of secrets seeds to the OEM in batches, and they OEM must store and care for these keys very carefully. There are some operational advantages to keeping the secret seeds around in some form, as the same secret seed could be used for other things. There are some significant downsides to keeping that secret seed around.

5. Public Key Infrastructures (PKI)

[RFC5280] describes the format for certificates, and numerous mechanisms for doing enrollment have been defined (including: EST [RFC7030], CMP [RFC4210], SCEP [RFC8894]).

[RFC5280] provides mechanisms to deal with multi-level certification authorities, but it is not always clear what operating rules apply.

The certification authority (CA) that is central to [RFC5280]-style public key infrastructures can suffer three kinds of failures:

1. disclosure of a private key,
2. loss of a private key,
3. inappropriate signing of a certificate from an unauthorized source.

A PKI which discloses one or more private certification authority keys is no longer secure.

An attacker can create new identities, and forge certificates connecting existing identities to attacker controlled public/private keypairs. This can permit the attacker to impersonate any specific device.

There is an additional kind of failure when the CA is convinced to sign (or issue) a certificate which it is not authorized to do so. See for instance [ComodoGate]. This is an authorization failure, and while a significant event, it does not result in the CA having to be re-initialized from scratch.

This is distinguished from when a loss as described above renders the CA completely useless and likely requires a recall of all products that have ever had an IDevID issued from this CA.

If the PKI uses Certificate Revocation Lists (CRL)s, then an attacker that has access to the private key can also revoke existing identities.

In the other direction, a PKI which loses access to a private key can no longer function. This does not immediately result in a failure, as existing identities remain valid until their expiry time (notAfter). However, if CRLs or OCSP are in use, then the inability to sign a fresh CRL or OCSP response will result in all identities becoming invalid once the existing CRLs or OCSP statements expire.

This section details some nomenclature about the structure of certification authorities.

5.1. Number of levels of certification authorities

Section 6.1 of [RFC5280] provides a Basic Path Validation. In the formula, the certificates are arranged into a list.

The certification authority (CA) starts with a Trust Anchor (TrAnc). This is counted as the first level of the authority.

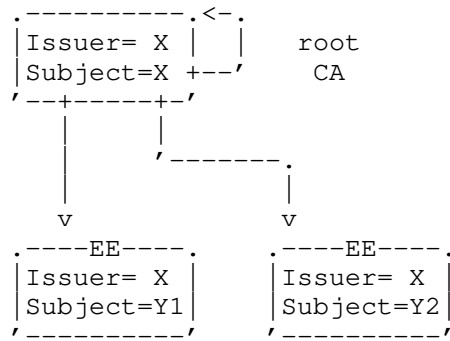
In the degenerate case of a self-signed certificate, then this a one level PKI.

```

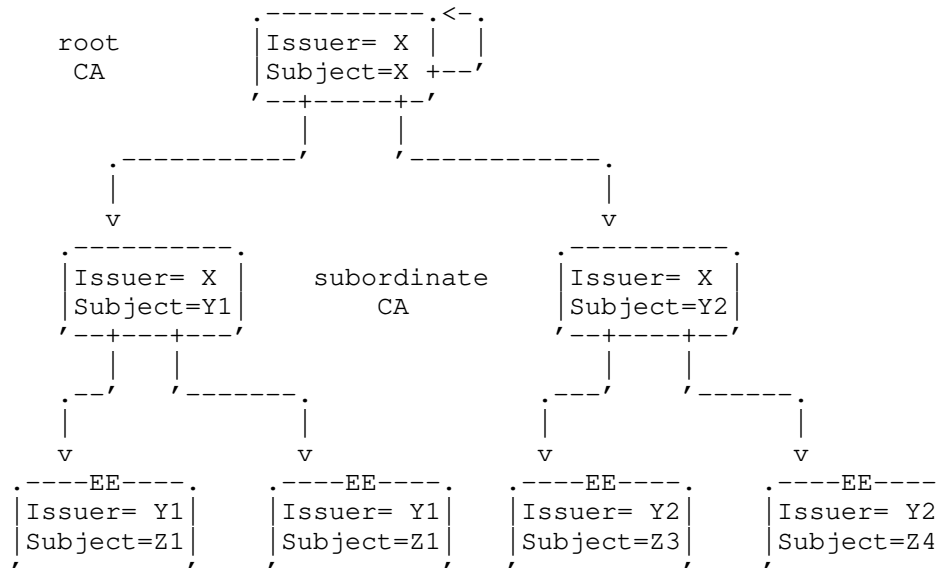
-----<-----
| Issuer= X |
| Subject=X |-----'
-----

```

The private key associated with the Trust Anchor signs one or more certificates. When this first level authority trusts only End-Entity (EE) certificates, then this is a two level PKI.



When this first level authority signs subordinate certification authorities, and those certification authorities sign End-Entity certificates, then this is a three level PKI.



In general, when arranged as a tree, with the End-Entity certificates at the bottom, and the Trust Anchor at the top, then the level is where the deepest EE certificates are, counting from one.

It is quite common to have a three-level PKI, where the root of the CA is stored in a Hardware Security Module, while the level one subordinate CA is available in an online form.

5.2. Protection of CA private keys

The private key for the certification authorities must be protected from disclosure. The strongest protection is afforded by keeping them in a offline device, passing Certificate Signing Requests (CSRs) to the offline device by human process.

For examples of extreme measures, see [kskceremony]. There is however a wide spectrum of needs, as exemplified in [rootkeyceremony]. The SAS70 audit standard is usually used as a basis for the Ceremony, see [keyceremony2].

This is inconvenient, and may involve latencies of days, possibly even weeks to months if the offline device is kept in a locked environment that requires multiple keys to be present.

There is therefore a tension between protection and convenience. This is often mitigated by having some levels of the PKI be offline, and some levels of the PKI be online.

There is usually a need to maintain backup copies of the critical keys. It is often appropriate to use secret splitting technology such as Shamir Secret Sharing among a number of parties [shamir79]. This mechanism can be setup such that some threshold k (less than the total n) of shares are needed in order to recover the secret.

5.3. Supporting provisioned anchors in devices

IDevID-type Identity (or Birth) Certificates which are provisioned into devices need to be signed by a certification authority maintained by the manufacturer. During the period of manufacture of new product, the manufacturer needs to be able to sign new Identity Certificates.

During the anticipated lifespan of the devices the manufacturer needs to maintain the ability for third parties to validate the Identity Certificates. If there are Certificate Revocation Lists (CRLs) involved, then they will need to re-signed during this period. Even for devices with a short active lifetime, the lifespan of the device could very long if devices are kept in a warehouse for many decades before being activated.

Trust anchors which are provisioned in the devices will have corresponding private keys maintained by the manufacturer. The trust anchors will often anchor a PKI which is going to be used for a particular purpose. There will be End-Entity (EE) certificates of this PKI which will be used to sign particular artifacts (such as software updates), or messages in communications protocols (such as TLS connections). The private keys associated with these EE certificates are not stored in the device, but are maintained by the manufacturer. These need even more care than the private keys stored in the devices, as compromise of the software update key compromises all of the devices, not just a single device.

6. Evaluation Questions

This section recaps the set of questions that may need to be answered. This document does not assign valuation to the answers.

6.1. Integrity and Privacy of on-device data

initial-enclave-location: Is the location of the initial software trust anchor internal to the CPU package? Some systems have a software verification public key which is built into the CPU package, while other systems store that initial key in a non-volatile device external to the CPU.

initial-enclave-integrity-key: If the first-stage bootloader is external to the CPU, and if it is integrity protected, where is the key used to check the integrity?

initial-enclave-privacy-key: If the first-stage data is external to the CPU, is it kept confidential by use of encryption?

first-stage-initialization: The number of people involved in the first stage initialization. An entirely automated system would have a number zero. A factory with three 8 hour shifts might have a number that is a multiple of three. A system with humans involved may be subject to bribery attacks, while a system with no humans may be subject to attacks on the system which are hard to notice.

first-second-stage-gap: If a board is initialized with a first-stage bootloader in one location (factory), and then shipped to another location, there may situations where the device can not be locked down until the second step.

6.2. Integrity and Privacy of device identify infrastructure

For IDevID provisioning, which includes a private key and matching certificate installed into the device, the associated public key infrastructure that anchors this identity must be maintained by the manufacturer.

identity-pki-level: how deep are the IDevID certificates that are issued?

identity-time-limits-per-subordinate: how long is each subordinate CA maintained before a new subordinate CA key is generated? There may be no time limit, only a device count limit.

identity-number-per-subordinate: how many identities are signed by a particular subordinate CA before it is retired? There may be no numeric limit, only a time limit.

identity-anchor-storage: how is the root CA key stored? How many people are needed to recover the private key?

6.3. Integrity and Privacy of included trust anchors

For each trust anchor (public key) stored in the device, there will be an associated PKI. For each of those PKI the following questions need to be answered.

pki-level: how deep is the EE that will be evaluated (the trust root is at level 1)

pki-algorithms: what kind of algorithms and key sizes will be considered to valid

pki-level-locked: (a Boolean) is the level where the EE cert will be found locked by the device, or can levels be added or deleted by the PKI operator without code changes to the device.

pki-breadth: how many different non-expired EE certificates is the PKI designed to manage?

pki-lock-policy: can any EE certificate be used with this trust anchor to sign? Or, is there some kind of policy OID or Subject restriction? Are specific subordinate CAs needed that lead to the EE?

pki-anchor-storage: how is the private key associated with this trust root stored? How many people are needed to recover it?

7. Privacy Considerations

many yet to be detailed

8. Security Considerations

This entire document is about security considerations.

9. IANA Considerations

This document makes no IANA requests.

10. Acknowledgements

Robert Martin of MITRE provided some guidance about citing the SBOM efforts. Carsten Borman provides many editorial suggestions.

11. Changelog

12. References

12.1. Normative References

[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>>.

[ieee802-1AR] IEEE Standard, "IEEE 802.1AR Secure Device Identifier", 2009, <<http://standards.ieee.org/findstds/standard/802.1AR-2009.html>>.

12.2. Informative References

[RFC8995] Pritikin, M., Richardson, M., Eckert, T., Behringer, M., and K. Watsen, "Bootstrapping Remote Secure Key Infrastructure (BRSKI)", RFC 8995, DOI 10.17487/RFC8995, May 2021, <<https://www.rfc-editor.org/info/rfc8995>>.

[I-D.richardson-anima-voucher-delegation] Richardson, M. and W. Pan, "Delegated Authority for Bootstrap Voucher Artifacts", Work in Progress, Internet-Draft, draft-richardson-anima-voucher-delegation-03, 22 March 2021, <<https://www.ietf.org/archive/id/draft-richardson-anima-voucher-delegation-03.txt>>.

- [I-D.friel-anima-brski-cloud]
Friel, O., Shekh-Yusef, R., and M. Richardson, "BRSKI Cloud Registrar", Work in Progress, Internet-Draft, draft-friel-anima-brski-cloud-04, 6 April 2021, <<https://www.ietf.org/archive/id/draft-friel-anima-brski-cloud-04.txt>>.
- [I-D.ietf-anima-constrained-voucher]
Richardson, M., Stok, P. V. D., Kampanakis, P., and E. Dijk, "Constrained Bootstrapping Remote Secure Key Infrastructure (BRSKI)", Work in Progress, Internet-Draft, draft-ietf-anima-constrained-voucher-15, 7 December 2021, <<https://www.ietf.org/archive/id/draft-ietf-anima-constrained-voucher-15.txt>>.
- [I-D.ietf-anima-brski-async-enroll]
Fries, S., Brockhaus, H., Oheimb, D. V., and E. Lear, "Support of Asynchronous Enrollment in BRSKI (BRSKI-AE)", Work in Progress, Internet-Draft, draft-ietf-anima-brski-async-enroll-04, 25 October 2021, <<https://www.ietf.org/archive/id/draft-ietf-anima-brski-async-enroll-04.txt>>.
- [I-D.moskowitz-ecdsa-pki]
Moskowitz, R., Birkholz, H., Xia, L., and M. C. Richardson, "Guide for building an ECC pki", Work in Progress, Internet-Draft, draft-moskowitz-ecdsa-pki-10, 31 January 2021, <<https://www.ietf.org/archive/id/draft-moskowitz-ecdsa-pki-10.txt>>.
- [RFC4949] Shirey, R., "Internet Security Glossary, Version 2", FYI 36, RFC 4949, DOI 10.17487/RFC4949, August 2007, <<https://www.rfc-editor.org/info/rfc4949>>.
- [RFC5011] StJohns, M., "Automated Updates of DNS Security (DNSSEC) Trust Anchors", STD 74, RFC 5011, DOI 10.17487/RFC5011, September 2007, <<https://www.rfc-editor.org/info/rfc5011>>.
- [RFC8366] Watsen, K., Richardson, M., Pritikin, M., and T. Eckert, "A Voucher Artifact for Bootstrapping Protocols", RFC 8366, DOI 10.17487/RFC8366, May 2018, <<https://www.rfc-editor.org/info/rfc8366>>.
- [RFC8572] Watsen, K., Farrer, I., and M. Abrahamsson, "Secure Zero Touch Provisioning (SZTP)", RFC 8572, DOI 10.17487/RFC8572, April 2019, <<https://www.rfc-editor.org/info/rfc8572>>.

- [RFC7030] Pritikin, M., Ed., Yee, P., Ed., and D. Harkins, Ed., "Enrollment over Secure Transport", RFC 7030, DOI 10.17487/RFC7030, October 2013, <<https://www.rfc-editor.org/info/rfc7030>>.
- [RFC8894] Gutmann, P., "Simple Certificate Enrolment Protocol", RFC 8894, DOI 10.17487/RFC8894, September 2020, <<https://www.rfc-editor.org/info/rfc8894>>.
- [RFC4210] Adams, C., Farrell, S., Kause, T., and T. Mononen, "Internet X.509 Public Key Infrastructure Certificate Management Protocol (CMP)", RFC 4210, DOI 10.17487/RFC4210, September 2005, <<https://www.rfc-editor.org/info/rfc4210>>.
- [_3GPP.51.011]
3GPP, "Specification of the Subscriber Identity Module - Mobile Equipment (SIM-ME) interface", 3GPP TS 51.011 4.15.0, 15 June 2005, <<http://www.3gpp.org/ftp/Specs/html-info/51011.htm>>.
- [RFC6024] Reddy, R. and C. Wallace, "Trust Anchor Management Requirements", RFC 6024, DOI 10.17487/RFC6024, October 2010, <<https://www.rfc-editor.org/info/rfc6024>>.
- [BedOfNails]
Wikipedia, "Bed of nails tester", 1 July 2020, <https://en.wikipedia.org/wiki/In-circuit_test#Bed_of_nails_tester>.
- [pelionfcu]
ARM Pelion, "Factory provisioning overview", 28 June 2020, <<https://www.pelion.com/docs/device-management-provision/1.2/introduction/index.html>>.
- [factoringrsa]
"Factoring RSA keys from certified smart cards: Coppersmith in the wild", 16 September 2013, <<https://core.ac.uk/download/pdf/204886987.pdf>>.
- [kskceremony]
Verisign, "DNSSEC Practice Statement for the Root Zone ZSK Operator", 2017, <<https://www.iana.org/dnssec/dps/zsk-operator/dps-zsk-operator-v2.0.pdf>>.

- [rootkeyceremony]
Community, "Root Key Ceremony, Cryptography Wiki", 4 April 2020,
<https://cryptography.fandom.com/wiki/Root_Key_Ceremony>.
- [keyceremony2]
Digi-Sign, "SAS 70 Key Ceremony", 4 April 2020,
<<http://www.digi-sign.com/compliance/key%20ceremony/index>>.
- [shamir79] Shamir, A., "How to share a secret.", 1979,
<<https://www.cs.jhu.edu/~sdoshi/crypto/papers/shamirturing.pdf>>.
- [nistsp800-57]
NIST, "SP 800-57 Part 1 Rev. 4 Recommendation for Key Management, Part 1: General", 1 January 2016,
<<https://csrc.nist.gov/publications/detail/sp/800-57-part-1/rev-4/final>>.
- [fidotechnote]
FIDO Alliance, "FIDO TechNotes: The Truth about Attestation", 19 July 2018, <<https://fidoalliance.org/fido-technotes-the-truth-about-attestation/>>.
- [ntiasbom] NTIA, "NTIA Software Component Transparency", 1 July 2020,
<<https://www.ntia.doc.gov/SoftwareTransparency>>.
- [cisqsbom] CISQ/Object Management Group, "TOOL-TO-TOOL SOFTWARE BILL OF MATERIALS EXCHANGE", 1 July 2020, <<https://www.it-cisq.org/software-bill-of-materials/index.htm>>.
- [ComodoGate]
"Comodo-gate hacker brags about forged certificate exploit", 28 March 2011,
<https://www.theregister.com/2011/03/28/comodo_gate_hacker_breaks_cover/>.
- [openbmc] Linux Foundation/OpenBMC Group, "Defining a Standard Baseboard Management Controller Firmware Stack", 1 July 2020, <<https://www.openbmc.org/>>.
- [JTAG] "Joint Test Action Group", 26 August 2020,
<<https://en.wikipedia.org/wiki/JTAG>>.

- [JTAGieee] IEEE Standard, "1149.7-2009 - IEEE Standard for Reduced-Pin and Enhanced-Functionality Test Access Port and Boundary-Scan Architecture",
DOI 10.1109/IEEESTD.2010.5412866, 2009,
<<https://ieeexplore.ieee.org/document/5412866>>.
- [rootkeyrollover]
ICANN, "Proposal for Future Root Zone KSK Rollovers",
2019, <<https://www.icann.org/en/system/files/files/proposal-future-rz-ksk-rollovers-01nov19-en.pdf>>.
- [CABFORUM] CA/Browser Forum, "CA/Browser Forum Baseline Requirements for the Issuance and Management of Publicly-Trusted Certificates, v.1.7.3", October 2020,
<<https://cabforum.org/wp-content/uploads/CA-Browser-Forum-BR-1.7.3.pdf>>.
- [I-D.richardson-rats-usecases]
Richardson, M., Wallace, C., and W. Pan, "Use cases for Remote Attestation common encodings", Work in Progress, Internet-Draft, draft-richardson-rats-usecases-08, 2 November 2020, <<https://www.ietf.org/archive/id/draft-richardson-rats-usecases-08.txt>>.
- [I-D.ietf-suit-architecture]
Moran, B., Tschofenig, H., Brown, D., and M. Meriac, "A Firmware Update Architecture for Internet of Things", Work in Progress, Internet-Draft, draft-ietf-suit-architecture-16, 27 January 2021, <<https://www.ietf.org/archive/id/draft-ietf-suit-architecture-16.txt>>.
- [I-D.ietf-emu-eap-noob]
Aura, T., Sethi, M., and A. Peltonen, "Nimble Out-of-Band Authentication for EAP (EAP-NOOB)", Work in Progress, Internet-Draft, draft-ietf-emu-eap-noob-06, 3 September 2021, <<https://www.ietf.org/archive/id/draft-ietf-emu-eap-noob-06.txt>>.
- [I-D.ietf-rats-architecture]
Birkholz, H., Thaler, D., Richardson, M., Smith, N., and W. Pan, "Remote Attestation Procedures Architecture", Work in Progress, Internet-Draft, draft-ietf-rats-architecture-14, 9 December 2021, <<https://www.ietf.org/archive/id/draft-ietf-rats-architecture-14.txt>>.
- [I-D.birkholz-suit-coswid-manifest]
Birkholz, H., "A SUIT Manifest Extension for Concise Software Identifiers", Work in Progress, Internet-Draft,

draft-birkholz-suit-coswid-manifest-00, 17 July 2018,
<<https://www.ietf.org/archive/id/draft-birkholz-suit-coswid-manifest-00.txt>>.

[I-D.birkholz-rats-mud]

Birkholz, H., "MUD-Based RATS Resources Discovery", Work in Progress, Internet-Draft, draft-birkholz-rats-mud-00, 9 March 2020, <<https://www.ietf.org/archive/id/draft-birkholz-rats-mud-00.txt>>.

[RFC8520] Lear, E., Droms, R., and D. Romascanu, "Manufacturer Usage Description Specification", RFC 8520, DOI 10.17487/RFC8520, March 2019, <<https://www.rfc-editor.org/info/rfc8520>>.

[I-D.ietf-sacm-coswid]

Birkholz, H., Fitzgerald-McKay, J., Schmidt, C., and D. Waltermire, "Concise Software Identification Tags", Work in Progress, Internet-Draft, draft-ietf-sacm-coswid-20, 26 January 2022, <<https://www.ietf.org/archive/id/draft-ietf-sacm-coswid-20.txt>>.

[RFC7168] Nazar, I., "The Hyper Text Coffee Pot Control Protocol for Tea Efflux Appliances (HTCPCP-TEA)", RFC 7168, DOI 10.17487/RFC7168, April 2014, <<https://www.rfc-editor.org/info/rfc7168>>.

[I-D.bormann-lwig-7228bis]

Bormann, C., Ersue, M., Keranen, A., and C. Gomez, "Terminology for Constrained-Node Networks", Work in Progress, Internet-Draft, draft-bormann-lwig-7228bis-07, 25 October 2021, <<https://www.ietf.org/archive/id/draft-bormann-lwig-7228bis-07.txt>>.

[I-D.ietf-teep-architecture]

Pei, M., Tschofenig, H., Thaler, D., and D. Wheeler, "Trusted Execution Environment Provisioning (TEEP) Architecture", Work in Progress, Internet-Draft, draft-ietf-teep-architecture-15, 12 July 2021, <<https://www.ietf.org/archive/id/draft-ietf-teep-architecture-15.txt>>.

Author's Address

Michael Richardson
Sandelman Software Works

Email: mcr+ietf@sandelman.ca

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

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

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

Abstract

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

Status of This Memo

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

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

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

This Internet-Draft will expire on August 22, 2021.

Copyright Notice

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

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

Table of Contents

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

1. Introduction

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

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

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

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

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

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

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

2. Terminology

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

3. Usage of bootstrapping terminology in standards

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

3.1. Device Provisioning Protocol (DPP)

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

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

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

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

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

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

3.3. Open Connectivity Foundation (OCF)

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

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

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

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

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

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

3.4. Bluetooth

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

3.5. Fast IDentity Online (FIDO) alliance

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

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

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

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

3.6. Internet Engineering Task Force (IETF)

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

3.6.1. Enrollment over Secure Transport (EST)

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

3.6.2. Bootstrapping Remote Secure Key Infrastructures (BRSKI)

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

3.6.3. Secure Zero Touch Provisioning

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

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

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

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

4. Comparison

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

5. Recommendations

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

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

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

6. Classification of available mechanisms

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

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

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

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

7. IoT Device Bootstrapping Methods

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

7.1. Managed Methods

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

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

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

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

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

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

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

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

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

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

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

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

7.1.1. Bootstrapping in LPWAN

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

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

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

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

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

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

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

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

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

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

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

7.3. Leap-of-faith/Opportunistic Methods

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

7.4. Hybrid Methods

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

8. Security Considerations

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

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

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

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

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

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

9. IANA Considerations

There are no IANA considerations for this document.

10. Acknowledgements

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

11. Informative References

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

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

- [kerberosiot] Hardjono, T., "Kerberos for Internet-of-Things", February 2014, <https://kit.mit.edu/sites/default/files/documents/Kerberos_Internet_of%20Things.pdf>.
- [LoRaWAN] Sornin, N., Luis, M., Eirich, T., and T. Kramp, "LoRa Specification V1.0", January 2015, <<https://www.lora-alliance.org/portals/0/specs/LoRaWAN%20Specification%201R0.pdf>>.
- [ocf] Open Connectivity Foundation, "OCF Security Specification", Open Connectivity Foundation , June 2017, <https://openconnectivity.org/specs/OCF_Security_Specification_v1.0.0.pdf>.
- [oma] Open Mobile Alliance, "Lightweight Machine to Machine Technical Specification: Core", Open Mobile Alliance , November 2020, <www.openmobilealliance.org/release/LightweightM2M/V1_2-20201110-A/OMA-TS-LightweightM2M_Core-V1_2-20201110-A.pdf>.
- [panaiot] Hernandez-Ramos, J., Carrillo, D., Marin-Lopez, R., and A. Skarmeta, "Dynamic Security Credentials PANA-based Provisioning for IoT Smart Objects", 2nd World Forum on Internet of Things (WF-IoT) , IEEE , 2015.
- [proximate] Mathur, S., Miller, R., Varshavsky, A., Trappe, W., and N. Mandayam, "Proximate: proximity-based secure pairing using ambient wireless signals.", Proceedings of MobiSys International Conference , pp. 211-224, June 2011.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC2904] Vollbrecht, J., Calhoun, P., Farrell, S., Gommans, L., Gross, G., de Bruijn, B., de Laat, C., Holdrege, M., and D. Spence, "AAA Authorization Framework", RFC 2904, DOI 10.17487/RFC2904, August 2000, <<https://www.rfc-editor.org/info/rfc2904>>.
- [RFC3748] Aboba, B., Blunk, L., Vollbrecht, J., Carlson, J., and H. Levkowetz, Ed., "Extensible Authentication Protocol (EAP)", RFC 3748, DOI 10.17487/RFC3748, June 2004, <<https://www.rfc-editor.org/info/rfc3748>>.

- [RFC3971] Arkko, J., Ed., Kempf, J., Zill, B., and P. Nikander, "SEcure Neighbor Discovery (SEND)", RFC 3971, DOI 10.17487/RFC3971, March 2005, <<https://www.rfc-editor.org/info/rfc3971>>.
- [RFC3972] Aura, T., "Cryptographically Generated Addresses (CGA)", RFC 3972, DOI 10.17487/RFC3972, March 2005, <<https://www.rfc-editor.org/info/rfc3972>>.
- [RFC4120] Neuman, C., Yu, T., Hartman, S., and K. Raeburn, "The Kerberos Network Authentication Service (V5)", RFC 4120, DOI 10.17487/RFC4120, July 2005, <<https://www.rfc-editor.org/info/rfc4120>>.
- [RFC4253] Ylonen, T. and C. Lonvick, Ed., "The Secure Shell (SSH) Transport Layer Protocol", RFC 4253, DOI 10.17487/RFC4253, January 2006, <<https://www.rfc-editor.org/info/rfc4253>>.
- [RFC4291] Hinden, R. and S. Deering, "IP Version 6 Addressing Architecture", RFC 4291, DOI 10.17487/RFC4291, February 2006, <<https://www.rfc-editor.org/info/rfc4291>>.
- [RFC4764] Bersani, F. and H. Tschofenig, "The EAP-PSK Protocol: A Pre-Shared Key Extensible Authentication Protocol (EAP) Method", RFC 4764, DOI 10.17487/RFC4764, January 2007, <<https://www.rfc-editor.org/info/rfc4764>>.
- [RFC5191] Forsberg, D., Ohba, Y., Ed., Patil, B., Tschofenig, H., and A. Yegin, "Protocol for Carrying Authentication for Network Access (PANA)", RFC 5191, DOI 10.17487/RFC5191, May 2008, <<https://www.rfc-editor.org/info/rfc5191>>.
- [RFC5272] Schaad, J. and M. Myers, "Certificate Management over CMS (CMC)", RFC 5272, DOI 10.17487/RFC5272, June 2008, <<https://www.rfc-editor.org/info/rfc5272>>.
- [RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol (NETCONF)", RFC 6241, DOI 10.17487/RFC6241, June 2011, <<https://www.rfc-editor.org/info/rfc6241>>.
- [RFC7030] Pritikin, M., Ed., Yee, P., Ed., and D. Harkins, Ed., "Enrollment over Secure Transport", RFC 7030, DOI 10.17487/RFC7030, October 2013, <<https://www.rfc-editor.org/info/rfc7030>>.

- [RFC7228] Bormann, C., Ersue, M., and A. Keranen, "Terminology for Constrained-Node Networks", RFC 7228, DOI 10.17487/RFC7228, May 2014, <<https://www.rfc-editor.org/info/rfc7228>>.
- [RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <<https://www.rfc-editor.org/info/rfc7230>>.
- [RFC7250] Wouters, P., Ed., Tschofenig, H., Ed., Gilmore, J., Weiler, S., and T. Kivinen, "Using Raw Public Keys in Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)", RFC 7250, DOI 10.17487/RFC7250, June 2014, <<https://www.rfc-editor.org/info/rfc7250>>.
- [RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <<https://www.rfc-editor.org/info/rfc7252>>.
- [RFC7593] Wierenga, K., Winter, S., and T. Wolniewicz, "The eduroam Architecture for Network Roaming", RFC 7593, DOI 10.17487/RFC7593, September 2015, <<https://www.rfc-editor.org/info/rfc7593>>.
- [RFC8040] Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", RFC 8040, DOI 10.17487/RFC8040, January 2017, <<https://www.rfc-editor.org/info/rfc8040>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.
- [RFC8366] Watsen, K., Richardson, M., Pritikin, M., and T. Eckert, "A Voucher Artifact for Bootstrapping Protocols", RFC 8366, DOI 10.17487/RFC8366, May 2018, <<https://www.rfc-editor.org/info/rfc8366>>.
- [RFC8376] Farrell, S., Ed., "Low-Power Wide Area Network (LPWAN) Overview", RFC 8376, DOI 10.17487/RFC8376, May 2018, <<https://www.rfc-editor.org/info/rfc8376>>.
- [RFC8572] Watsen, K., Farrer, I., and M. Abrahamsson, "Secure Zero Touch Provisioning (SZTP)", RFC 8572, DOI 10.17487/RFC8572, April 2019, <<https://www.rfc-editor.org/info/rfc8572>>.

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

[vermillard]

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

[wps]

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

Authors' Addresses

Mohit Sethi
Ericsson
Hirsalantie 11
Jorvas 02420
Finland

Email: mohit@piuha.net

Behcet Sarikaya
Denpel Informatique

Email: sarikaya@ieee.org

Dan Garcia-Carrillo
University of Oviedo
Oviedo 33207
Spain

Email: garciadan@uniovi.es