IoT Device Identific*A*tion and Reco*G*nition (*IoTAG*)

Lukas Hinterberger*
and Bernhard Weber†
Dept. Electrical Engineering and
Information Technology
Ostbayerische Technische Hochschule
Regensburg, Germany
email:
lukas.hinterberger@st.oth-regensburg.de*
bernhard1.weber@st.oth-regensburg.de†

Sebastian Fischer
Secure Systems Engineering
Fraunhofer AISEC
Berlin, Germany
email:
sebastian.fischer@aisec.fraunhofer.de

Katrin Neubauer‡
and Rudolf Hackenberg§
Dept. Computer Science and Mathematics
Ostbayerische Technische Hochschule
Regensburg, Germany
email:
katrin1.neubauer@oth-regensburg.de‡
rudolf.hackenberg@oth-regensburg.de§

***Abstract*—To ensure the secure operation of IoT devices in the future, they must be continuously monitored. This starts with an inventory of the devices, checking for a current software version and extends to the encryption algorithms and active services used. Based on this information, a security analysis and rating of the whole network is possible. To solve this challenge in the growing network environments, we present a proposal for a standard. With the *IoT* Device Identific*A*tion and Reco*G*nition (*IoTAG*), each IoT device reports its current status to a central location as required and provides information on security. This information includes a unique ID, the exact device name, the current software version, active services, cryptographic methods used, etc. The information is signed to make misuse more difficult and to ensure that the device can always be uniquely identified. In this paper, we introduce IoTAG in detail and describe the necessary requirements.**

Keywords*—Internet of Things; device identification; open standard; IoTAG; security rating.

I. Introduction

It has been shown that the use of Internet of Things (IoT) technologies is always associated with risks. Both, in terms of data protection and the reliability and security of an IoT environment. It is not always possible to completely eliminate all sources of risk. However, the threat potential can be reduced by introducing new technologies to simplify system maintenance. The Federal Office for Information Security in Germany lists measures to protect IoT devices [1]. Based on this, requirements for a compatibility interface will be defined, which can be used to implement a central and manufacturer-independent security management of IoT systems. The data to be provided and the security requirements to be fulfilled by the interface will be based on the draft of the European standard for the security of IoT devices ETSI EN 303 645 v2.0.0 [2]. This standard defines basic requirements for the security of IoT devices.

In order to be able to monitor and evaluate each component individually, even in complex IoT environments, a way to identify each device is required. This means that each device must have a unique identifier, which may only be assigned at least once within a closed system. At best, the identifier is unique worldwide.

After each device has been recorded individually, it must also be possible to identify the product type. This enables the devices to be classified in safety categories. For example, the failure of an individual telephone must be considered less critical than the failure of an alarm system. It is only possible to test a device for existing weak points or for information published by the manufacturer if the product type is known. In the latter case, at least the manufacturer and a clear product designation is required.

In order to be able to check whether the firmware of a device is up-to-date, it is necessary for a device to provide its currently running firmware version. In addition, information about the update behaviour of the device must be provided. This includes information on whether the device can be updated, whether it has an automatic update mechanism and up to which point in time updates are provided by the manufacturer.

IoT devices are by definition in exchange with other network components. This can be done either locally isolated in a separate network or globally over the Internet. To protect sensitive data from unauthorized access, the use of verified algorithms and communication protocols is required. By providing an overview of the encryption and hashing algorithms used by a device, it is possible to check whether outdated or insecure procedures are used. The same applies to network protocols and network technologies in use. If a device provides all network protocols it supports, including the protocol version, it can be checked whether the device is vulnerable to attacks against its communication.

To meet these requirements, we present a proposal for a standard for the detection of IoT devices: *IoT* Device Identific*A*tion and Reco*G*nition (*IoTAG*). It focuses on the security of the devices and will provide necessary information to estimate the security of all devices in the network.

This paper is based on our previous publication [3] and extends the idea of IoTAG with the necessary descriptions and more details. It is structured as follows: Section II shows some

 ISBN: 978-1-61208-778-8

related work and Section III consists of the IoTAG definition, subdivided into the dataset, the serialization, the integrity and the communication. In Section IV, a brief conclusion is given.

II. Related Work

There are several suggestions to detect IoT devices automatically. However, most of them only consider functionality and not security. Two concepts which also consider security are the Thing Description and the Device Description Language.

A. Thing Description

The concept of Thing Description (TD), presented by the World Wide Web Consortium (W3C), is a uniform representation of metadata of a device, as well as the interfaces provided by the device. It can be either physical or purely virtual properties. These include device properties, such as currently stored settings or sensor data.

In addition, information is provided on available control actions or events that can be used to interact with the devices. Furthermore, an optional "Security" field provides information on the authorization procedures available for accessing device resources. According to the W3C specifications, the TD is exclusively a data exchange format for device metadata that can be provided by the device itself or by an independent resource [4].

For the use of this technology as a security interface, the device-independent provision of information proves to be problematic. As a result, it cannot be guaranteed that the information actually refers to the device and that the data records are up-to-date and correct. The Thing Description specification does not provide any procedures for the transmitted values to be signed by the device or another instance [4].

Also, the small amount of security related information described above excludes TD for use in an automated monitoring and testing scenario. Our requirements for the predefined dataset, data integrity, as well as the availability of predefined communication procedures can be regarded as not fulfilled.

B. Device Description Language

The IoT Device Description Language (IoT-DDL) is a machine- and human-readable XML-based description for IoT devices. The IoT-DDL is used by a device to provide information about its capabilities, resources, entities and services, as well as cloud-based functionalities. This includes information about the hardware installed in a device (e.g., Secure Elements), software functions (e.g., switching the device on or off) or external services (e.g., log server), as well as descriptive metadata, which can include the device manufacturer or the device name. But the scope of this information has not been firmly defined.

The IoT-DDL focuses on both device-to-device and device-to-cloud communication and is intended to simplify the creation of heterogeneous IoT scenarios. Message Queuing Telemetry Transport (MQTT) and the Constrained Application Protocol (CoAP) are supported for communication between devices. The security mechanisms supported by these protocols are not used to secure the communication. Instead, a specially developed AES-based procedure is used to encrypt the transmitted data [5].

Thus, in the case of the IoT-DDL, the requirement for a firmly defined dataset and its integrity are also violated.

III. IoTAG Definition

In contrast to existing interoperability procedures for facilitating the setup and control of IoT infrastructures, as presented in Section II, a new technical proposal for the automated identification and recognition of IoT devices (called IoTAG) will be defined.

The focus of the IoTAG definition lies on the standardized provision of security-critical device data, the integrity preservation of the datasets to be transmitted and the relevance of the information for an individual classification of each device with regard to the implementation of security specifications and recommendations. When designing the necessary guidelines for this purpose, the requirements defined at the beginning of this document for such a communication standard are taken into account.

The implementation of IoTAG on the devices is done by the manufacturers.

A. Dataset

The section "Dataset" consists of subsections which represent some of the attributes. These give an explanation why this information must be provided by an IoTAG compatible device and a description of the attribute's content.

1) Manufacturer: The provision of a manufacturer's designation is useful for several reasons. On the one hand, there is always the possibility that devices with identical or very similar designations are sold by different companies. On the other hand, this information can be used to contact the manufacturer and inform them about software errors or to be able to make use of support services.

The information about the manufacturer thus contains the name of the company that provides the firmware and its updates. This is a string value that contains the official company name according to the respective entry in the commercial register.

2) Name: The name or designation of a device serves to identify the product. This attribute contains the product name under which the device is sold by the manufacturer in the form of a string.

3) Serial number: The serial number of a product is a unique marking of a device assigned by the manufacturer and enables its identification within a product line. In the event of production faults, the affected devices can be identified by their serial numbers.

The representation of the serial number is manufacturer-specific. Basically, it is an arbitrary string of characters that

 ISBN: 978-1-61208-778-8

is unique for each device in a product series and thus, in conjunction with the manufacturer and product name, allows a clear conclusion to be drawn about the object.

4) Type: The device type provides information about the type of product. It indicates the main functionality of the device and can be used to draw conclusions about the complexity of the device. This information is important for the security of an IoT system in that it allows conclusions to be drawn about the effects an attack on a device may have. For example, an attack on a surveillance camera is a greater problem from the perspective of data protection than an attack on a smoke detector, for example. A locking system or an alarm system, must also be classified as more security-critical than a TV.

Possible values for the product type specification are:

- alarm system
- camera
- smart lock
- smart speaker
- smart TV
- smoke detector

This list contains first suggestions and can be extended at any time by further product definitions.

5) ID: In order to be able to identify a device at any time, it must have a unique identifier. For this reason, a combination of existing information is created for device identification: the manufacturer, the product name and the serial number. Although this information can be calculated by software, it is also stored for manual linking of a dataset to a device.

The specifications included in the generation of the device ID are to be regarded as constants and must not be changed subsequently after the device has been taken into operation.

The device ID is a character string that contains a hash value in alphanumeric representation. It is generated by concatenating the information about manufacturer, product name and serial number, then generating the hash value of this string using the SHA-256 algorithm [6] and finally encoding the resulting binary data as base16 string [7]. The use of SHA2 and SHA3 family algorithms is recommended by the National Institute of Standards and Technology (NIST). For performance reasons, the SHA2 algorithms are preferred to the SHA3 algorithms [8] [9] [10].

6) Category: The product category fulfills a purpose comparable to that of the product type. An additional security-related assessment can be carried out by dividing devices into categories that describe their area of application or use scenario. If the two categories “lighting” and “assisted living” are considered as two different areas, the failure of a device can have different effects. If a motion sensor responsible for the lighting fails, the user must activate the light manually. However, if a motion sensor from the assisted living area, which is supposed to report whether the occupant of a house is entering and leaving the bathtub, fails, the help hoped for by using this system can be missed in an emergency like a fall in the bathtub. Thus, devices in the assisted living area are to be classified as more security-critical than pure comfort functions.

The device category attribute can have the following values, among others:

- assisted living
- entertainment
- household
- industry
- infrastructure
- lighting
- personal assistance
- security

These are also initial proposals. With the increasing spread of IoT devices, the fields of application are also expanding, so that further definitions are necessary.

7) Secure boot: Secure boot mechanisms can be used to ensure the integrity of a device’s firmware at system startup. When a device is started securely, signature mechanisms are used to check whether the components involved, such as the boot loader and operating system, are unchanged originals. The information required for this verification is stored in a suitable hardware module, such as a Trusted Platform Module (TPM) [11] [12].

The secure boot attribute uses a boolean value. If no software integrity check is performed, the value is “false”.

8) Firmware: In order to be able to check that the firmware of a device is up-to-date, it must publish the firmware version currently being executed. If a device requires a manual installation of the firmware, there must be a possibility to retrieve it from the manufacturer. To prevent software from being obtained from dubious sources, the IoTAG dataset also provides an internet address for downloading the firmware.

In contrast to the specifications described so far, the firmware is not an atomic value, but two strings to be considered separately: the firmware version (referred to as “version”) as published by the device manufacturer, and a Uniform Resource Locator (URL) [13], which refers to the download resource for this firmware.

For consecutive versions, lexicographically ascending terms are recommended so that the order of release can be determined.

9) Client software: If software for third-party devices is required for the use of an IoT device, IoTAG will provide the latest version supported by the device. In addition, a link to a resource is also provided here from which this software can be obtained. This eliminates the need for the user to search for a source of supply, which in turn reduces the risk of obtaining software from untrusted sources.

The specifications of the client software are analogous to those of the device firmware (see 3.2.8). However, if no client software is required, empty strings are specified.

 ISBN: 978-1-61208-778-8

10) Updates: The update behaviour of a device provides information on whether a device updates itself automatically, i.e., whether it checks for the availability of new firmware versions and obtains and installs them, or whether it must be manually updated to the new version.

It should be noted, that even if a device is configured for automatic updating, the provision of new firmware by the device manufacturer is also necessary. In order to be able to take a device out of service when it is no longer supplied with new software, it is necessary to specify the point in time from which this is the case.

The update configuration information is also a multi-part record within the update item. A boolean value is used to indicate whether a device has an automatic update mechanism and also uses it. “Automatic updates” is chosen as the name for this value.

The end of support is a date formatted as a string according to ISO 8601 [14] and integrated into IoTAG under the name “end of life”.

11) Cryptography: In order to be able to make predictions about the cryptographic capabilities of a device, it is necessary that the algorithms used by a device to secure its communication are known and a statement can be made as to whether these are implemented in hardware or software. It must also be specified whether secret keys are stored exclusively in secure hardware or also in memory areas accessible via software.

The private key required for the signature of IoTAG as described in subsection C is treated individually. A separate variable is introduced to show how this key is managed, as it is essential for the reliability of IoTAG.

Two identical structures are subordinated to the superordinate term cryptography. Each contains an attribute “IoTAG key”, which is a boolean value. If the signature key is managed in a secure hardware environment and cannot be read by software, it takes the value “true” in the hardware structure and the value “false” in the software structure. The reverse is true if the key is accessible via software.

Another boolean value is the variable “key store”. This indicates whether cryptographic keys to be kept secret are stored in this area. This specification can be true in both structures. An overview of the cryptographic algorithms used in a device is given by the variable “algorithms”. This contains a collection of character strings. Each element of this collection contains a cryptographic algorithm according to its standardised designation (example: “ecdsa-sha2-nistp256”, as defined in RFC 5656 [15]).

12) Connectivity: The connectivity of a device describes its physical possibilities to connect to communication partners. Different technologies are used for data exchange. These include the standards under IEEE 802.3 and IEEE 802.11 [16] developed by the Institute of Electrical and Electronics Engineers (IEEE) as well as industrial standards such as Bluetooth [17], ZigBee [18] or other.

For compatibility reasons, IoT devices can support older versions in addition to the current standard of a communication method. But if these have security problems, an attacker can use them to gain access to confidential information [19] [20].

For the transmission of the supported communication standards, a multi-part data structure is used. For example, it will have the attributes “IEEE802_11”, “Bluetooth” and “ZigBee”. Each of them forms a collection of strings. While in the case of Bluetooth and ZigBee the alphanumeric version numbers are included, for the IEEE family of standards, the suffixes of the individual standards are entered. If the suffix begins with a hyphen, it is removed. The first standard of the family is specified with an empty character string. Additionally, the collection can contain the values “WEP”, “WPA”, “WPA2” and “WPS”. These inform whether the respective technology is used by a device.

13) Services: Network devices offer various services to interact with them. Analogous to securing the communication against external attacks as described in subsection D, the interception of the connection by devices within the network must also be prevented. This goal can be achieved, among other things, by dispensing with unencrypted transmission protocols. It should be noted, however, that the implementation of these protocols can also contain errors and therefore the version of the software used must be checked and published by IoTAG.

A separate data structure is defined to describe a network service. This contains the name of the service (Name), the network port (Port), the protocol used (Protocol) including any version designations, as well as the name and version of the software (Software) that offers the service in the format <designation>-<version>. Since the information whether the connection is UDP or TCP-based is also required to specify the network port, the port is specified in the format <Port>/<UDP|TCP>.

The actual IoTAG attribute is ultimately a collection that contains such a data structure for each service offered.

B. Serialization

To prevent incompatibilities due to incorrect interpretations, the serialization format Javascript Object Notation (JSON), according to the specification in ECMA-404 [21] and RFC 8259 [22] with UTF-8 encoding, is selected.

JSON is preferred over the Extensible Markup Language (XML) because it has a higher performance in terms of memory resource consumption and computing power [23].

Listing 1 shows a serialized IoTAG data set. The attribute names to be used can be taken from this example. For space reasons, the value of the “ID” attribute has been wrapped into two lines.

```
{
 "Manufacturer": "Beispiel GmbH",
 "Name": "Example-Device",
 "SerialNumber": "D1.0",
 "Type": "example device",
 "ID": "2071c7736acd16f6cea3727d3b7ecde5
```

 ISBN: 978-1-61208-778-8

```
        3f4c2e97b421f3550248e19d7309c636",
 "Category": "infrastructure",
 "SecureBoot": false,
 "Firmware": {
    "Version": "1.0",
    "URL": "https://192.168.102.94:10000/FirmwareInfo"
 },
 "ClientSoftware": {
    "Version": "",
    "URL": ""
 },
 "Updates": {
    "AutomaticUpdates": false,
    "EndOfLife": "2021-01-01T00:00:00"
 },
 "Cryptography": {
    "Software": {
        "IoTAGKey": true,
        "KeyStore": true,
        "Algorithms": [
            "RSASSA-PSS",
            "SHA-256",
            "TLS_AES_128_GCM_SHA256",
            "TLS_CHACHA20_POLY1305_SHA256",
            "aes256-ctr",
            "ecdsa-sha2-nistp521",
            "diffie-hellman-group-exchange-sha256",
            "hmac-sha2-256,hmac-sha2-512"
        ]
    },
    "Hardware": {
        "IoTAGKey": false,
        "KeyStore": false,
        "Algorithms": []
    }
 },
 "Connectivity": {
    "IEEE802_3": [
        "WPA2",
        "b",
        "g",
        "n"
    ],
    "Bluetooth": [
        "4.2"
    ],
    "ZigBee": []
 },
 "Services": [
    {
        "Name": "IoTAG",
        "Port": "27795/TCP",
        "Protocol": "HTTP/2",
        "Software": "IoTAG-Server"
    },
    {
        "Name": "SSH",
        "Port": "22/TCP",
        "Protocol": "SSH-2",
        "Software": "OpenSSH-8.1"
    }
 ]
}
```


Listing 1. IoTAG example

C. Integrity

1) Signature algorithm and authentication: The RSA procedure serves as the basis for the signature mechanism of IoTAG. A minimum length of 2048 bits is recommended for the keys required by this procedure [24]. Since the RSA algorithm would always generate the same encryption text for identical messages, methods have been developed that combine the plaintext with a random value, the padding, before each encryption process. The Public-Key Cryptography Standards (PKCS) define in PKCS#1 with RSASSA-PKCS1-v1_5 and RSASSA-PSS two signing procedures for RSA that take such padding into account. The latter is preferable for new developments, which is why it is used for IoTAG signatures using the standard options defined in PKCS#1 [25].

To verify the signature, the message recipient must know the sender's public key. However, this must also ensure that an attacker has not mistakenly published his key to the recipient and is therefore able to generate misleading messages whose signature is considered valid by the recipient. To counteract this, the signer's public key is published in conjunction with a certificate, which in turn is signed by a trusted third party [12]. In IoTAG certificates are used according to the specification in ITU-T X.509 [26] and RFC 2459 [27]. Such a certificate can be issued directly by the manufacturer of a device and stored on the device, or it can be created when the device is set up and then signed by a local or external certification authority.

2) Signed dataset: Basically, the target of the signature is always the IoTAG dataset in serialized form and thus a UTF-8 encoded character string (see subsection B). However, not this entire string is used for the signature, but instead a hash sum is calculated from it, which is then signed. As recommended by NIST, the SHA-256 algorithm is used to generate this sum [28].

Before the hash algorithm can be applied, the IoTAG string is converted into a byte array. Only from this array the hash sum is calculated, to which the signature algorithm is then applied. If the array contains a terminating null byte, this is ignored in the hash calculation.

D. Communication

The last open point to be defined is the IoTAG related communication behaviour. This includes not only the retrieval of IoTAG data from a device, but also the retrieval of software resources via an URL, provided inside the IoTAG dataset. The same technologies are used for both procedures, which is why a general description of the communication endpoint, the transmission protocol and the data format is given before the two procedures are explained in more detail.

1) General description: The Hypertext Transfer Protocol Version 2 with Transport Layer Security (TLS) [29] is selected as the transmission protocol (HTTPS) [30]. This means that an HTTPS-capable server application must be provided as the communication endpoint for querying information, which has a trustworthy certificate for encrypted communication. This application does not have to support the full scope of operations defined in RFC 2616 [31], it only has to be able to respond to a single GET request by providing the respective data record. The addressed resource is determined by the respective URL.

For formatting the data for transmission within HTTP packets, JSON is used.

2) Retrieving Software Resources: It is defined that the IoTAG data set provided by a device always contains a URL to obtain the latest available device firmware or, if necessary, the software for client systems. It is not possible to download the software directly via this URL. Instead, it is used to execute the HTTP request described in subsection A. The response to this request contains a JSON object, which in turn has the string

 ISBN: 978-1-61208-778-8

attributes “URL” and “Version”. This URL can now be used directly to download the firmware. The second specification informs about the version of the software.

3) Retrieving IoTAG: Every IoTAG compatible device must provide a communication interface to retrieve the IoTAG data set. In order to make this procedure uniform, a unique HTTP URL must be defined via which a corresponding resource can be accessed. This requires a uniform port number and a predefined path for the request to the HTTP server. 27795 is specified as the network port. The path consists of a single segment called “iotag”. This results in the following URL scheme, whereby the “<host>” statement is to be interpreted according to the definition in RFC 3986 paragraph 3.2.2. [13]: https://<host>:27795/iotag

4) Transmitted data record: The specification of the signature process shows that in addition to the actual IoTAG data set, additional information is required to verify its correctness. This is a certificate containing the key needed to verify the signature as well as the signature itself. A separate JSON object is also defined for this, which contains this information in the form of the attributes “IoTAG”, “Certificates” and “Signature”.

Since the signature is present as a byte sequence during its calculation, it is encoded for transmission to base64 and can thus be integrated into the JSON object as a string.

A uniform form for the transfer of the certificate must be ensured. For the transmission of ITU-T X.509 certificates in non-binary form, the coding according to RFC 7468 [32] is suitable. Basically, the certificate is first converted into a binary structure, taking into account the coding rules specified in ITU-T X.690 [33], and then encoded to base64, which also allows it to be embedded as a string in the JSON object. If additional certificates are required for the verification of the certificate, all certificates are first encoded and the resulting character strings are then concatenated. The order of the certificates must be observed according to the specification in RFC 5246 chapter 7.4.2 [29].

The IoTAG dataset could be entered directly as an object, since it is JSON-serialized for transmission anyway. To check the signature, the IoTAG object must be extracted from the parent object. This can be done in two ways: the recipient can still treat the transmitted data as a string and try to extract the IoTAG object by manipulating it. However, this procedure is unusual and involves additional development effort, since the corresponding extraction routine must be implemented. Alternatively, the received JSON object can be deserialized to an object of the respective programming language and then processed further.

Although the latter approach is preferable, it also makes signature verification more problematic. To perform this step, the IoTAG object must be serialized to a string again after extraction to calculate the hash sum. However, this serialization produces different results depending on the software used, and thus ultimately results in different hash values. A signature check based on the respective hash sums would thus fail, although the information remained unchanged.

To counter this problem, a way must be found to transfer the IoTAG data set within a JSON object in such a way that it can be extracted by deserialization without affecting the formatting. This can be achieved by treating the serialized IoTAG data for transmission as a string rather than as an object. In this case, all JSON control characters within this string must be replaced by appropriate escape sequences before transmission to ensure error-free interpretation. However, these must also be removed by the receiver before the hash calculation in order not to falsify the result.

Instead, preference is given to another approach. Here, the transmission of the IoTAG data as a string is retained, but the character string resulting from its serialization is first base64 encoded. The result of this process is then set as the value of the IoTAG attribute. This enables the recipient of the data to parse the received JSON object and decode the information it contains, which ultimately results in the same form as it was processed by the sender.

IV. Conclusion

With IoTAG, a fast and easy solution for the security management of IoT networks is described. The proposed standard includes the necessary information for a risk analysis and the possibility to monitor all devices, regarding to their running software version, protocols and the encryption algorithms.

The implementation can be realized with little effort and the security of the whole network can be improved easily. However, this proposed standard must be implemented and integrated into products by all manufacturers.

This standard can also help attackers to gain information about the devices in the network, but with an improved overview over the devices and their security state, it helps more than it brings new risks.

As a next step, IoTAG can be discussed as a standard or an existing standard can be extended with the features of IoTAG. For this purpose, the signature process must also be adopted to ensure data integrity.

We are currently working on implementation examples to help getting started with IoTAG. With these different implementations, it is also possible to evaluate the best methods and libraries for the signature and the JSON serialization.

References

[1] Federal Office for Information Security (Germany), “SYS.4.4: Allgemeines IoT-Gerät,” IT-Grundschutz-Kompendium 2. Version 2019, Cologne, Bundesanzeiger Verlag GmbH, 2019, p. 3.

[2] European Telecommunications Standards Institute, “Draft ETSI EN 303 645 V2.0.0 (2019-11),” 2019.

[3] S. Fischer, K. Neubauer, L. Hinterberger, B. Weber, and R. Hackenberg, “IoTAG: An Open Standard for IoT Device IdentificAtion and RecoGnition,” The Thirteenth International Conference on Emerging Security Information, Systems and Technologies, IARIA, 2019, pp. 107-113.

[4] World Wide Web Consortium, “Web of Things (WoT) Thing Description,” Apr. 2018. [Online]. Available from: https://www.w3.org/TR/wot-thing-description/ [accessed: 2020-07-20].

 ISBN: 978-1-61208-778-8

[5] A. E. Khaled, A. Helal, W. Lindquist, and C. Lee, “IoT-DDL—Device Description Language for the “T” in IoT,” IEEE Access, Nr. 6, pp. 24048-24063, Apr. 2018.
[6] U.S. Department of Commerce und National Institute of Standards and Technology, “Secure Hash Standard (SHS),” 2015.
[7] Internet Engineering Task Force, “RFC 4648 - The Base16, Base32, and Base64 Data Encodings,” Oct. 2006. [Online]. Available from: https://tools.ietf.org/html/rfc4648. [accessed: 2020-07-20].
[8] National Institute of Standards and Technology, “NIST Policy on Hash Functions - Hash Functions — CSRC,” May 2019. [Online]. Available from: https://csrc.nist.gov/Projects/Hash-Functions/NIST-Policy-on-Hash-Functions. [accessed: 2020-07-20].
[9] R. K. Dahal, J. Bhatta, and T. N. Dhamala, “Performance Analysis of SHA-2 and SHA-3 Finalists,” International Journal on Cryptography and Information Security (IJCIS), Sept. 2013, pp.720-730.
[10] U.S. Department of Commerce und National Institute of Standards and Technology, “SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions,” 2015.
[11] J. Vermillard, “Sicherheit für IoT-Geräte,” Linux Magazin, Oct. 2015.
[12] A. S. Tanenbaum, “Moderne Betriebssysteme,” Hallbergmoos: Pearson Deutschland GmbH, 2009, pp. 720-721.
[13] Internet Engineering Task Force, “RFC 3986 - Uniform Resource Identifier (URI): Generic Syntax,” Jan. 2005. [Online]. Available from: https://tools.ietf.org/html/rfc3986. [accessed: 2020-07-20].
[14] International Organization for Standardization, “ISO 8601:2004: Data elements and interchange formats — Information interchange — Representation of dates and times,” 2004.
[15] Internet Engineering Task Force, “RFC 5656 - Elliptic Curve Algorithm Integration in the Secure Shell Transport Layer,” Dec. 2009. [Online]. Available from: https://tools.ietf.org/html/rfc5656. [accessed: 2020-07-20].
[16] A. Healey, “GET 802(R) Standards,” [Online]. Available from: https://ieeexplore.ieee.org/browse/standards/get-program/page/series?id=68. [accessed: 2020-07-20].
[17] Bluetooth SIG, Inc., “Bluetooth Core Specification, Revision 5.2,” 2019.
[18] ZigBee Alliance, “ZigBee Specification,” 2015.
[19] P. Kraft and A. Weyert, “Network Hacking,” Franzis Verlag GmbH, 2015, pp. 345-360.
[20] J. Erickson, “Hacking,” dpunkt.verlag GmbH, 2009, pp. 472-488.
[21] ECMA International, “The JSON Data Interchange Syntax,” 2017.
[22] Internet Engineering Task Force, “RFC 8259 - The JavaScript Object Notation (JSON) Data Interchange Format,” Dec. 2017. [Online]. Available from: https://tools.ietf.org/html/rfc8259. [accessed: 2020-07-20].
[23] N. Nurseitov, M. Paulson, R. Reynolds, and C. Izurieta, “Comparison of JSON and XML data interchange formats: A case study,” International Conference on Computer Applications in Industry and Engineering, CAINE, 2009, pp.157-162.
[24] U.S. Department of Commerce and National Institute of Standards and Technology, “Recommendation for Key Management,” 2015.
[25] Internet Engineering Task Force, “RFC 8017 - PKCS #1: RSA Cryptography Specifications Version 2.2,” Nov. 2016. [Online]. Available from: https://tools.ietf.org/html/rfc8017. [accessed: 2020-07-20].
[26] International Telecommunication Union, “Recommendation ITU-T X.509,” 2016.
[27] Internet Engineering Task Force, “RFC 2459 - Internet X.509 Public Key Infrastructure Certificate and CRL Profile,” Jan. 1999. [Online]. Available from: https://tools.ietf.org/html/rfc2459. [accessed: 2020-07-20].
[28] National Institute of Standards and Technology, “NIST Policy on Hash Functions - Hash Functions — CSRC,” May 2019. [Online]. Available from: https://csrc.nist.gov/Projects/Hash-Functions/NIST-Policy-on-Hash-Functions. [accessed: 2020-07-20].
[29] Internet Engineering Task Force, “RFC 5246 - The Transport Layer Security (TLS) Protocol Version 1.2,” Aug. 2008. [Online]. Available from: https://tools.ietf.org/html/rfc5246. [accessed: 2020-07-20].
[30] Internet Engineering Task Force, “RFC 7540 - Hypertext Transfer Protocol Version 2 (HTTP/2),” May 2015. [Online]. Available from: https://tools.ietf.org/html/rfc7540. [accessed: 2020-07-20].
[31] Internet Engineering Task Force, “RFC 2616 - Hypertext Transfer Protocol – HTTP/1.1,” June 1999. [Online]. Available from: https://tools.ietf.org/html/rfc2616. [accessed: 2020-07-20].
[32] Internet Engineering Task Force, “RFC 7468 - Textual Encodings of PKIX, PKCS, and CMS Structures,” Apr. 2015. [Online]. Available from: https://tools.ietf.org/html/rfc7468. [accessed: 2020-07-20].
[33] International Telecommunication Union, “Recommendation ITU-T X.690,” 2015.

 ISBN: 978-1-61208-778-8