Cyberphysical Security for the Masses: A Survey of the Internet Protocol Suite for Internet of Things Security

Hannes Tschofenig and Emmanuel Baccelli

Abstract—As IoT deployments expand, IoT security lags infamously behind. This paper surveys IoT security protocols standardized by the IETF, maps protocols to regulatory IoT security guidelines (from ENISA), and discusses remaining gaps. We observe that, while IETF protocols alone do not completely secure IoT devices, they go a long way.

Index Terms—IoT, Security, IETF.

1 Introduction

Over the last few years, the number of Internet of Things (IoT) deployments has seen a steady increase. Heterogeneous, innovative IoT hardware is being rolled out at a fast pace, catalyzed by the new availability of open-source, general-purpose, embedded IoT software and open standards for IoT communication.

In parallel, alarming recent reports in both academic work and mainstream media warn about potential cyber-vulnerabilities and actual cyber-attacks involving IoT. Increased awareness of the need for improved security is pushing legislators to pay closer attention to the IoT environment and issue new regulations, such as the first IoT cybersecurity law (SB-327) and, more recently, the European Commission's Cybersecurity Act.

In this paper, we focus on constrained IoT devices (described in RFC 7228 [1]) used in the realm of smart home IoT deployments. Typically, constrained IoT devices use microcontrollers - for instance Arm Cortex-M - on which run real-time operating systems, such as FreeRTOS, RIOT, Micrium's μC/OS or Mbed OS [2]. Compared to machines that run full-blown operating systems, such as Linux, constrained IoT devices use a fraction of the power and are equipped with RAM and Flash sizes (in the kilobyte range) reminiscent of the days of the Commodore VIC-20. Constrained IoT devices, which cannot afford the energy drain of Wi-Fi, connect to the network via low-power, wireless, link-layer technologies, such as Bluetooth Low-Energy, IEEE 802.15.4, LoRa, 3GPP Cellular IoT (NB-IoT), or via wired buses, such as BACnet or Ethernet.

Although EU initiatives, such as IoT security recommendations from ENISA [3], are paving the way to a more secure IoT, reports of completely insecure IoT devices are still commonplace. What causes companies to roll out products that are highly insecure? Is this the result of engineering design flaws, a lack of incentive for product management, or a combination of both?

In this context, we survey the coverage of the Internet protocol specifications for network security standardized by the Internet Engineering Task Force (IETF) and examine the extent to which they address the ENISA recommendations for IoT security. We focus on the IETF because (i) it has been successful in standardizing many of the other Internet protocols that underpin the Internet at large, (ii) the IETF protocol standards aim to work across the various IoT link-layer technologies, as well as to interoperate between different vendors, and (iii) IETF specifications are open and freely available, which enables us to analyze existing work – contrary to the diverse proprietary technologies available on the market.

In the IoT security domain, the IETF positions itself within a larger ensemble of standardization bodies gathering industry groups and organizations with which it collaborates, such as the IEEE, OMA SpecWorks, Open Connectivity Foundation (OCF), Thread Group, oneM2M, Fairhair Alliance, WiSun, Zigbee, and Bluetooth SIG. The work of these organizations is extremely valuable in improving IoT security since many of these develop complete systems, often by re-using building blocks developed by the IETF. While summarizing their work – even at a high level – is beyond the scope of this paper, we relate to these organizations where useful, throughout the paper.

2 Security and Privacy Threats

We consider a simple smart home scenariowhereby several IoT devices are disseminated in a home environment, and interact via the network, primarily with (i) a gateway, and (ii) the cloud or some form of back-end, accessed via the Internet and the gateway. IoT devices may also directly interact with one another. We consider the simplest case whereby all of the required device-to-device and device-to-gateway interactions within the home are achieved with a single link-layer hop; in other words, a scenario in which routing is not necessary within the home.

- *H. Tschofenig is with Arm Limited, United Kingdom.*
- *E. Baccelli is with Inria, France*

2.1 Categories of Attacks

The common approach to designing a security solution is to conduct a threat analysis first. Broadly, attacks can be categorized (as depicted in Fig. 1) into (i) network (or communication) attacks, (ii) software attacks, and (iii) hardware attacks.

Historically, the IETF focused on countering network attacks, and ignored software and hardware attacks. For this reason, the main threat model, as described in RFC 3552 [4], is a classical network attacker who can carry out active and passive attacks against the communication interaction. Only recently, the IETF started tackling some security concerns beyond network attacks, with the work on attestation, trusted execution environments, and firmware updates (see Section 4.6).

Privacy-related threats were added later with RFC 6973 [5] and are now being considered in the design of protocols throughout the IETF, including IoT protocols. In some scenarios there is an overlap of privacy and security threats (for example, in a surveillance scenario), and there are some privacy-specific threats, such as data minimization and user participation. The threat of pervasive monitoring, as revealed by Snowden, certainly had a huge impact on recognizing the importance of privacy in the design of Internet protocols, as documented by the IETF community in RFC 7258 [6].

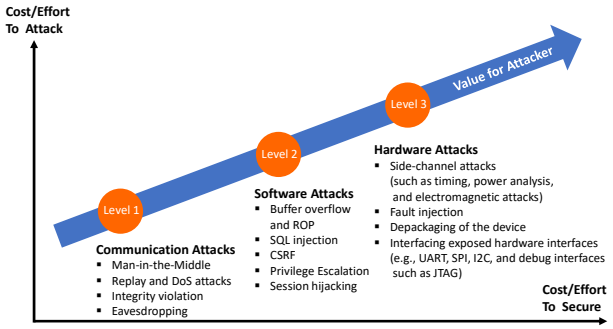

Fig. 1. Threat Categories.

2.2 Threat Modelling

A common technique to start threat modelling is collect a list of assets that need to be protected. Then, the threat analysis considers what attackers can do to these assets, and what the impact of those actions are. In addition to the use of a methodology, like the Microsoft STRIDE Threat Model, it is also useful to consider recommendations (or checklists) of commonly occurring security problems. An example of such a checklist can be found in the ENISA good practices for security of Internet of Things [3].

A typical caveat, however, is that sequences of product design decisions may introduce threats that were initially absent. Furthermore, a wildly successful IoT device will be used in unexpected ways and deployed in environments never imagined by the engineers designing the device. As such, threat modelling is hard, and it is often necessary to examine the threats not only once, at the beginning of the product development process, but also to adjust the list of threats during the product lifecycle.

The following are three examples that illustrate the challenging nature of the threat analysis activity:

- Consider a product that is sold to consumers and deployed in the customer's home. Often these products are considered physically secure because a thief breaking into someone's home is probably less interested in tampering with a smart home product, such as a thermostat or a kettle. However, if the same customer sometimes rents his home to strangers, via services like Airbnb, the threat model has to be re-evaluated. This case illustrates a potential change in the deployment environment. Accurate threat modelling requires a good understanding of how customers use their products, how valuable the target is to the attacker, and the changing landscape of threats.
- Consider another product that uses strong crypto and a rock-solid security protocol to protect the integrity and confidentiality of a firmware update mechanism. Imagine further that the engineers of this product decide to deploy the same symmetric key, a class key, on all devices of the same product family. In this scenario, an attacker only has to buy and analyze a single device - for example, by conducting a (novel) side channel analysis - to obtain the class key and ship a firmware update to all devices in the same product family. In the worst case, the attacker can deliver the firmware updates over the Internet without having to be in close proximity to the devices. This example shows how the changing capabilities of attackers can change the threats posed. The people conducting the threat modelling must be aware of the attackers' techniques and their level of sophistication.
- Consider a final example where the threat analysis addresses all of the assets that the company selling the product cares about. Later, it turns out that the product is vulnerable to a reflection attack, which was not considered in the threat analysis because the computing resources of third parties (the victims of the attack) were not included in the asset list. Those involved in threat modelling have to be aware of the type of attacks encountered in specific industries today.

In this survey we focus on network adversaries, where the attacker has access to the communication – a model also known as the Dolev-Yao threat model. IETF security typically discusses the threats that are considered in the design. Such an approach is useful because the purposes and the details of protocols vary, despite the underlying Dolev-Yao threat model being the same. For example, for firmware updates the threats are described in the SUIT architecture and information model documents while the TLS 1.3 specification itself contains threats relevant for the TLS protocol.

3 Overview of ENISA Guidelines

Once an initial set of threats is collected, it is time to think about which of those threats should be dealt with, and how. Although a company may decide not to offer a technical solution to all security threats, it is useful to consult industry guidelines, such as the ENISA guidelines [3], which can help to make reasonable decisions. While many security requirements relate to the software and hardware implementation of a product, such as the boot process and hardware crypto, there are a number of requirements which relate to communication protocol. Within the latter category, we list below the most important groups of security considerations:

1) Authentication and communication security
2) Object security
3) Authorization and access control
4) Key management
5) State-of-the-art crypto
6) Restrictive communication
7) Firmware and software updates

Many of the requirements that fall into the above seven groups can also be found in other best current practice guides. In fact, there is an astonishing overlap among the guidelines produced by governments, industry groups, and researchers. This gives us confidence that these guidelines indeed represent a consensus of sorts among security experts.

4 IETF IoT Security Standardization

The seven groups listed in the previous section can be mapped loosely to areas of work in the IETF. In the next subsections, we provide a high-level overview of the ongoing standardization work.

4.1 Authentication and Communication Security

Security requirements often include authenticity, confidentiality and integrity of a communication interaction. A key question is thereby what the 'endpoints' of this exchange are. For some use cases, it may be sufficient to offer security at the link layer because the two endpoints of the communication are topologically close to each other. For other use-cases, communication between the two endpoints involves application layer gateways. The IETF covers communication security at the IP layer and above.

4.1.1 End-to-End Security over UDP or TCP

Let's consider the usecase whereby an IoT device in a smart home periodically wakes up, and sends sensor readings to a cloud-based backend via an access-point to which it connects over wireless.

Here, "end-to-end" refers to communication between the IoT device and the remote server whereby constrained IoT devices typically use UDP at the transport layer (more often than TCP). To secure communication between two endpoints over TCP, the IETF standardized Transport Layer Security (TLS). Conversely, Datagram Transport Layer Security (DTLS) secures communications over UDP. TLS/DTLS rely on a handshake setting up a security context between two endpoints, which provides authenticity, confidentiality and integrity for subsequent communication between these endpoints over the transport protocol in use (TCP and UDP, respectively). Computationally-heavy tasks are performed infrequently (as part of the handshake) while application data protection is accomplished with symmetric keys derived during the handshake. RFC 7925 [7], which defines profiles for TLS and DTLS 1.2 that are tailored for IoT devices, is frequently used.

The latest development in this area is TLS/DTLS 1.3, a new version of these standards. The final TLS 1.3 [8] specification was published is August 2018. Compared to TLS 1.2, TLS 1.3 aims improves performance with a more efficient handshake, modernizes cipersuites, and offers better privacy protection. Meanwhile, DTLS 1.3 specification is being finalized. Compared to DTLS 1.2, DTLS 1.3 reduces bits-over-the-air, provides better support for sleepy IoT devices, and improves reliability in face of packet loss during the handshake.

4.1.2 Object Security with COSE

Let's now consider another use case whereby an IoT software provider updates (remotely, over the Internet) the software on an IoT device deployed in a smart home.

In this case, "end-to-end" means from the software developer to the IoT device. A typical workflow is (i) the developer builds a firmware image, (ii) uploads it on a server and (iii) the server eventually serves the firmware update to the IoT device, at the first occasion (e.g. the next time the latter turns up). The firmware image must be self-contained and may be stored on server(s) for an extended period of time, which may be untrusted. Here, we are thus aiming to secure infrequent, asynchronous communication, and "one-shot" payloads. To secure such communication end-to-end, TLS/DTLS is not appropriate. The IETF standardized a different security mechanism, COSE [9], which we cover next.

A recent trend in IoT is to use the Concise Binary Object Representation (CBOR) encoding and serialization format. For reference, compared with the JavaScript Object Notation (JSON) format, CBOR was designed with (much) smaller code and message size in mind. In this context, COSE is to CBOR what JSON Object Signing and Encryption (JOSE) is to JSON: the signing and encryption format for CBOR-encoded data. COSE can be seen as a set of building blocks that applications use in the way they find useful (keeping an eye on code size). A device implementing a firmware update solution may, for example, rely on asymmetric crypto and would, therefore, implement the signature verification capability offered by COSE and none of the other security services that use symmetric key crypto. COSE offers several security services, including digital signatures, couter signatures, Message Authentication Code (MAC), encryption and rudimentary key distribution methods.

CBOR is a fairly recent standard, and COSE only became RFC in mid-2017; therefore, Libcose and COSE-C are the only implementations tailored for use in the microcontroller environment, so far.

Since CBOR is a good fit for securing data when the goal is to protect one-shot, self-contained payloads and data at rest.

4.1.3 End-to-End Security over CoAP

Since the publication of the CoAP specification (RFC 7252) in 2014, a variety of embedded IoT stacks support it, typically in conjunction with DTLS. Compared to HTTP, CoAP aims at smaller messages, as well as smaller code size, while providing a RESTful interface which matches purposely well with HTTP commands.

When CoAP is deployed with CoAP proxies and application layer gateways TLS/DTLS cannot provide end-to-end security because they a classical TLS/DTLS exchange terminates at the gateway. To secure CoAP messages the IETF defines another communication security solution called Object Security for Constrained RESTful Environments (OSCORE [10]). OSCORE defines a CoAP option and a mechanism reusing COSE to protect CoAP messages. Note that OSCORE does not protect the entire CoAP message because CoAP proxies must be able to inspect part of the message.

Since OSCORE does not offer key management itself, it has to rely on a separate key management protocol. The ACE-OAuth framework [11] can be reused to facilitate such key management. Other key management protocols currently investigated in the IETF are Ephemeral Diffie-Hellman over COSE (EDHOC) and Application Layer TLS (ATLS). EDHOC is a competitor to TLS/DTLS and ATLS. Both, EDHOC and ATLS, allow the establishment of an OSCORE security context.

4.2 Authorization and Access Control

Many IoT devices in the smart home environment, such as door locks, kettles, and thermostats, have to be accessed by users via their everyday devices, such as their smartphones and tablets. This interaction raises a number of authorization questions, including:

- How can users access the IoT device securely, without inadvertently allowing unauthorized access?
- Is it possible to support different users or groups of users?
- Can the access rights of one user be set differently from other users?
- Can access control policies be managed centrally?
- Does the solution scale for a greater number of IoT devices?
- How can strong authentication mechanisms be utilized?

The IETF Authentication and Authorization for Constrained Environments (ACE) working group has developed a solution to answer the above-listed questions, based on the widely used OAuth 2.0 protocol, and brings fine-grained authorization to the IoT world. This body of work is called ACE-OAuth [11] and the specifications are already further along with interoperability tests taking place. While OAuth 2.0 was designed with a wide range of use cases in mind - such as the web, native apps on smart phones, tablets and desktop computers, browser-based apps, and even devices with a limited user interface, such as TVs, picture frames, and game consoles - a few enhancements had to be made to tailor OAuth to the constrained IoT environment.

First, let us briefly summarize how ACE-OAuth is expected to work at a high-level. The OAuth system consists of four main entities, namely the client, the resource server, the authorization server, and the resource owner. The client wants to access a protected resource on a resource server. In a door lock example, the client could be software running on the smart phone, and the resource server is the door lock. The authorization server is responsible for issuing tokens that allow the client to access the door lock such that the authorization policies stored on the authorization server are met. These authorization policies are typically created by the resource owner, or phrased differently, the resource owner decides about who gets access to the protected resource - in our example, the door lock.

The token, which is issued by the authorization server and consumed by the resource server, is called the access token. The OAuth working group standardized a token format, called JSON Web Token (JWT), which encodes claims in JSON, and the token itself is protected using the mechanisms developed in the IETF JOSE working group. Since the OAuth 2.0 specification does not mandate a token format, it is possible to design and use a token format that works best in each environment. In the case of ACE-OAuth, a counterpart to the JWT was developed with the CBOR Web Token (CWT), which uses CBOR instead of JSON, and COSE and instead of JOSE. The result is a smaller token size.

Access tokens used in OAuth 2.0-based deployments are mostly bearer tokens. Proof-of-possession (PoP) tokens were introduced later, and in the IoT environment it is possible to use PoP tokens from the beginning. The key characteristic of a PoP token is that the token itself is associated with a symmetric or asymmetric key, and the entity presenting the token must demonstrate possession of the key. When a public key is associated with the PoP token, this means that the client must use the private key along with a digital signature when demanding access to a protected resource on a resource server. An attacker, therefore, needs to steal the key associated with the token in addition to the token itself. The use of PoP tokens requires an enhancement to the original OAuth 2.0 specification because extra information about the keys to be bound to the tokens must be passed around.

The second enhancement to the OAuth 2.0 specification accommodates for the different protocols being used in the IoT environment. While HTTP is used for IoT communication, the Constrained Application Protocol (CoAP) and the Message Queuing Telemetry Transport (MQTT) protocol are also popular. OAuth 2.0 has not been developed for use with these protocols, and the necessary extensions are defined in the IETF ACE working group.

4.3 Key Management

IoT devices need several keys to enable remote management and services enablement. As explained in a whitepaper by the Internet Protocol for Smart Objects (IPSO) Alliance on credential management for IoT devices [12], a common security assumption is that IoT devices have been provisioned with at least one long-term credential during manufacturing along with trust anchors. This long-term credential is then used to provision further keys to the device in a process called bootstrapping (or alternatively commissioning, onboarding, or enrollment). This new terminology hides the fact that the developed protocols provide two main steps:

1) The IoT device uses the manufacturer-provided credentials and the trust anchor(s) to authenticate to a bootstrap server[1]
2) An exchange to provision or derive new credentials is executed between the IoT device and the bootstrap server. These new keys, often called operational credentials, are used to secure the communication of the device, including data and configuration exchanges.

Many vendors and standards-developing organizations have designed protocols that provide the functionality of bootstrapping. Some of these protocols are specified for use with a particular radio technology (as the Bluetooth Low Energy example shows) and others are radio technology agnostic. Organizations and vendors that have developed these bootstrapping techniques include OMA SpecWorks (with LwM2M),Open Connectivity Foundation (OCF), Thread Group, Intel (with their Secure Device Onboarding), Wi-Fi Alliance (Device Provisioning Protocol), Alljoyn, WiSun, Zigbee, and Bluetooth (Mesh).

Unsurprisingly, the IETF has also developed a number of these protocols for use in IoT environments, as outlined in this survey [13]. Examples include ZeroTouch, ANIMA/BRSKI, Enrollment over Secure Transport (EST), EAP-NOOB, ACE-OAuth, Certificate Management Protocol (CMP), Certificate Management over CMS (CMC), PANA, and EAP/AAA.

4.4 State-of-the-Art Crypto

Cryptography is a conservative business. Developing a new cryptographic algorithm, writing proofs, publishing papers, and standardizing the cryptography is not enough to make it see widespread adoption, or any adoption at all. Years of community-wide review is typically required, which makes cryptography quite costly in terms of development. An engineer given the task of developing software for an IoT product is therefore well served to rely on off-the-shelf crypto rather than going for the cutting edge. Resisting the urge to develop your own cryptographic algorithm is key.

Many of the IETF working groups today rely on the recommendations offered by the Internet Research Task Force (IRTF) Crypto Forum Research Group (CFRG). The recommendations for cryptographic algorithms on IoT devices differs from those used on higher end devices, such as desktops and servers.

For symmetric key cryptography, the industry has moved to authenticated encryption with additional data (AEAD) ciphers. IoT devices often implement the Counter with CBC-MAC (CCM) in hardware, rather than the Galois/Counter Mode (GCM). GCM is preferred by applications that require high data throughput because the mode can be pipelined. AES-CCM uses as its only primitive the AES encrypt operation and this makes it suitable to compact implementations. The CCM mode of operation with AES has been put into most, if not all, IoT specifications as the preferred algorithm. To reduce the overhead caused by the message authentication code (MAC), CCM also supports a variant with a shortened MAC, referred to as CCM-8. CCM-8 is commonly used in IoT deployments.

With TLS 1.3, GCM mode with AES-128-GCM-SHA256 is the 'must'-implement algorithm, and the same AEAD algorithm with a longer key size, namely AES-256-GCM-SHA256, is a 'should'-implement. As a backup cipher, CHACHA20-POLY1305-SHA256 has become popular and it is also a 'should'-implement.

For asymmetric key cryptography, rather than RSA, most IoT deployments use Elliptic Curve Cryptography (ECC) because the latter uses smaller keys for comparable security (a better fit for storage-, network- and CPU-challenged devices). The algorithm of choice for certificates today is ECDSA-SECP256r1-SHA256. For the key exchange, the NIST P-256 curve (secp256r1) is mandatory to implement in TLS 1.3 and the preferred choice in many IoT specifications. As an alternative X25519 is available for key exchange and Ed25519 for digital signatures.

In the future, we may see convergence in the area of symmetric crypto algorithms used on IoT devices and on the general Internet with ChaCha20 and Poly1305.

4.5 Restricting Communication

Most IoT devices are built to fulfil a specific purpose; typically to collect sensor input, process it, and act on the results. In edge or fog computing, even sensor fusion is often outsourced to a gateway, or it is delegated to the server-side infrastructure. Naturally, the communication interaction of such IoT devices is also limited, particularly since these devices have limited RAM, which prevents communication with multiple hosts concurrently.

The earlier work of the IETF Network Endpoint Assessment (NEA) working group aimed at assessing and restricting communication of endpoints, w.r.t an organization's policies. However, it turned out difficult to write and maintain policies for endpoints with versatile communication interaction such as laptops and desktops. Hence, in the end, the NEA standardization work had less security impact than expected.

Recently, with the much simpler communication interaction of IoT devices, the idea of restricting the communication of these devices surfaced again. An organization interested in formulating a policy for what an IoT device is allowed to do faces the challenge that it requires some analysis work to determine which protocols a device is using, and with which hosts it is communicating on the Internet. Only the company developing the final product has complete information available about the communication behavior since it is in full control of developing (or selecting) the software stack.

Although IoT security recommendations highlight the need to remove unused services, network operators still inherently distrust the manufacturers of these devices. The current assumption seems to be that IoT devices will not be developed with security in mind. This lack of confidence in the ability to develop secure IoT devices has led network equipment manufacturers, network operators, and security start-ups to develop various schemes for "learning" whether

1. There is, however, one important exception where devices start without having credentials provisioned, and they use an out-of-band mechanism to obtain security guarantees. Bluetooth Low Energy, for example, uses this approach where in proximity, a PIN entry, is used as a security guarantee to skip the provisioning step.

an IoT device behaves correctly. While products on the market use some form of deep packet inspection and analysis of communication interactions by intermediaries somewhere in the access network, the standardized solution expects the equipment manufacturer to publish Manufacturer Usage Descriptions (MUD [14]). The MUD documents offer information about the communication interaction of the IoT device. These MUD files are communicated from the IoT device to the network using the Dynamic Host Configuration Protocol (DHCP). Alternatively, an URL pointing to a MUD file can be carried as an extension in certificates. Equipment in the network then fetches these MUD files and uses the machine-readable description to create firewall policies automatically. For example, it is expected that a device that misbehaves due to an attack will be firewalled, or at least that alerts will be generated. A manufacturer has to update these MUD files for each product whenever the communication interaction changes; for example, when the device interacts with different servers or uses alternative communication protocols.

At the time of writing, the standardization work on MUD is largely completed and it remains to be seen whether manufacturers who previously did not care about implementing a firmware update solution and alike can now be convinced to publish MUD files about their products in a timely fashion. At the same time, many companies are trying hard to improve the security of endpoints with new hardware security mechanisms and with readily available open source code that takes most of the difficult security programming out of the hands of developers.

4.6 Firmware and Software Updates

Providing a firmware update solution for IoT devices is essential to dealing with bugs and the changing environment. While this seems obvious, we still see lots of devices in the market that do not enable updating their code. Best current practice guides not only demand the ability to update firmware, but also the ability to do so securely. In 2017, the IETF formed the Software Updates for Internet of Things (SUIT) working group to standardize an IoT firmware update solution. The initial starting point was to outline the architecture and the metadata describing a firmware image along with its security mechanism [15]. The metadata contained in a data structure that is protected, at least against modifications, is called a manifest. IoT device management solutions, such as LwM2M, can then be used to deliver the manifest and the firmware image to IoT devices.

The manifest contains several elements to instruct an IoT device to install only firmware that comes from an authorized source, has not been modified, is (optionally) confidentiality protected, is suitable for the hardware, and meets various other conditions.

The IETF SUIT aims to cover a wide range of use cases, not only because the IoT market is quite diverse in its deployment needs, but also because of the hardware being used. Since many IoT devices today contain multiple microcontrollers to perform different tasks, such as radio communication and application processing, the manifest also has to provide an indication of which microcontroller must be updated. Likewise, a single microcontroller may have software from different vendors, such as a Bluetooth software stack from the chip manufacturer and application code from a developer. Transferring large firmware images can be a challenge for low-power radio technologies, and some vendors, therefore, prefer to use differential updates or to update selected components instead. The manifest also has to cover these use cases.

Dealing with different versions of firmware over the lifetime of the device is another a requirement. These scenarios require some form of dependency mechanism and version management. The use of an encrypted firmware image is also becoming more popular because it reduces the attack surface since knowledge of the code running on an IoT device, even if it must be reverse engineered, gives an attacker a lot of additional insight. Obtaining a firmware image is often the first task in an attack chain.

While the standardization process in the IETF SUIT group is still ongoing, the group is interested in reusing existing building blocks, such as CBOR and COSE, to encode and protect firmware images, respectively.

The IETF SUIT working group is related to the Trusted Execution Environment Provisioning (TEEP) working group since the goal in both cases is to update code on a device. The main difference between SUIT and TEEP is that TEEP focuses on hardware that uses Trusted Execution Environments (TEEs), such as Arm TrustZone or Intel SGX. A TEE is designed to provide a hardware-isolation mechanism to separate a regular operating system from security-sensitive application components, such as key storage. The goal is, therefore, to update code that runs inside these TEEs, the so-called trusted apps. Most of the hardware equipped with a TEE today - for example, gateway devices in IoT deployments - is rather powerful compared to a constrained IoT device; however, with the recent addition of TrustZone technology to the Arm M-profile architecture, constrained microcontrollers are gaining trusted execution capabilities.

The TEEP working group is standardizing an application layer protocol, the Open Trust Protocol (OTrP [16]), which manages the interaction between a TEE and a server-side component, the so-called Trusted Application Manager (TAM), to query the TEE for installed trusted apps and to manage the lifecycle of these trusted apps. Trusted apps are only installed if the TAM has successfully attested the hardware and software functionality of the TEE. The design of OTrP is complicated by the interaction of the trusted app and code running on the regular operating system, the different ways of delivering software to higher-end devices, such as phones and tablets, and because devices may be equipped with multiple TEEs. The group agreed to re-use the SUIT manifest for the actual software update and the attestation mechanism developed in a recently established working group – the Remote ATtestation ProcedureS (RATS) working group.

The RATS working group aims to standardize a container for attestation information. The most promising candidate is the Entity Attestation Token (EAT) format, which introduces attestation-specific claims for use in the CWT structure. The promise of the work is to enable IoT devices to communicate information about the manufacturer, used hardware security features, and hashes computed over the bootloader and firmware code of the device to a com-

munication party, opening up additional decision-making possibilities.

5 Discussion

While the IETF is obsessed with protocol design, the ENISA Baseline Security Recommendations for IoT go beyond and also concern processes, software and hardware-related properties. Considering attacks on IoT systems, we see that the most common types of attacks and vulnerabilities can be addressed by the IETF IoT security standards:

- Inadequate software update mechanism
- Missing key management and default passwords
- Unauthorized access to configuration data
- Missing communication security
- Physical attacks

Obviously, physical attacks are largely outside the scope of the IETF. Furthermore, reference standardization work is still ongoing and must be completed before the results can be applied to products. Beyond these trivial observations, what are the gaps?

5.1 Integration Challenges

To design a secure IoT product, a developer needs to put various building blocks correctly together to produce a usable system. This systems-level approach has historically not been handled by the IETF, which is more focused on developing the building blocks. As such, for a complete IoT device management solution, engineers must look elsewhere; for example, to the OMA SpecWorks or to the OCF. It can be confusing to engineers who are developing products to find out suddenly that the building blocks developed by the IETF do not fit nicely into the systems developed by these other organizations.

5.2 Implementation Gaps

Implementations for constrained IoT devices often lag behind standardization. Existing open source code is often incomplete, difficult to integrate into off-the-shelf hardware, and has not been well-tested and peer reviewed. It is not uncommon to see IoT-centric protocols implemented in Java or Python, while the language of choice for embedded developers today is C or C++.

Integrating security into constrained IoT devices still requires a lot of know-how. Due to the constraints of these devices, many layers of abstraction offered by high-end devices that run full-blown operating systems, like Linux, are not available. Developers, therefore, have to familiarize themselves with the features of their hardware and with how to access the various features, such as the Random Number Generator (RNG), the Memory Protection Unit (MPU), or hardware crypto. Worse: many IoT operating systems even lack memory-safety and isolation features.

To facilitate bridging implementation gaps, new standards in the area of APIs in the style of the Cortex Microcontroller Software Interface Standard (CMSIS) could be useful. In the context of IETF, one might wonder whether it would make sense to shift attention from communication security to endpoint security, and to aim at playing a bigger role in developing such standards.

5.3 Open Standards vs. Proprietary Solutions

Network security technologies can typically be used in all verticals, but the smart home market is unique. Getting standardized security technologies deployed in the home market is challenging because many of the key players are still under the impression that they will dominate the market soon(ish), and they do not consider the need to interoperate with any other product, in the belief that 'the winner takes it all'. Currently, if interoperation exists, it is typically through proprietary mechanisms with application layer gateways.

We do, however, expect this situation to change over time as many of the IoT protocols and security features become commodities rather than differentiators. It is also likely that end-users will expect to have IoT devices that interoperate rather than having each product family be interconnected only through a gateway that requires yet another device in the household.

Note that interoperability and the use of open standards does not necessarily make it easier for attackers to attack a larger number of devices. Typically the number of reviews in a standards developing organization increases the quality of both the technical specification and the (multiple) implementations of that specification.

5.4 Choosing a Communication Security Paradigm

So far, we see the use of link layer security mechanisms and DTLS/TLS in IoT deployments. Object-level security is used very selectively in combination with DTLS/TLS, but often in a proprietary way. The adoption of IPsec/IKE and HIP has failed due to the lack of product-quality embedded stacks. It remains to be seen whether OSCORE will be successful in the market, particularly since the new version of TLS/DTLS 1.3 is now also available. Intuitively, an object-level security solution, like COSE, is easier to deploy than OSCORE because it is independent of the underlying transport and, in many deployments, the end-to-end path experiences a lot of protocol translation, often with considerable changes in API semantics. Any end-to-end security solution must consider what the endpoints are because protecting sensor readings and other application data may not always be possible when the gateway is supposed to perform algorithmic computations, such as machine learning.

5.5 Future-proof Cryptography for IoT

A development worth noting is the work on post-quantum crypto in light of the fact that many studies have focused on creating algorithms that resist large, specialized quantum computers. While many of the algorithms are still being evaluated - for example, as part of the NIST Post-Quantum Cryptography standardization effort - there are concerns that some IoT devices that are expected to have a lifetime of 10-20 years will encounter problems because of the progress made with quantum computers. For this reason, the use of new algorithms (e.g. hash-based signatures, considered post-quantum-safe) has been suggested as a way to secure firmware updates. There are, however, also challenges with these new algorithms. Many of the post-quantum algorithms are characterized by slower performance, larger key

size, or a larger signature size than conventional public key crypto schemes.

5.6 Choosing a Key Management Solution

Key management is an important aspect in IoT security, including for the establishment of communication security, and for bootstrapping. As shown in this survey, there are many choices available, which makes it difficult to deploy technology that interoperates.

While this an area of ongoing work, it is not clear whether these different mechanisms are likely to converge, and how many of them will see widespread deployment, or any deployment at all. Although many of the proposals are similar at a high level, there are some subtle differences in their designs. For example, some allow only certain credential types to be used, work only with specific authentication and key exchange protocols (such as TLS/DTLS), make assumptions regarding connectivity and intermediaries, or work only on specific layers in the protocol stack. Because each proposal introduces new terminology comparing them is often challenging.

In summary, developments in the area of key management beg the question of whether more choice is always better for developers.

6 Conclusion

Designing a secure IoT product is hard, as demonstrated by press releases about hacked IoT products. Recently, guidelines and recommendations for securing IoT devices have been published. In this paper we have surveyed categories of security protocols standardized by IETF, specifically designed for constrained IoT devices, for authentication and communication security, object security, authorization and access control, key management, cryptography, restricting communication, as well as software updates. The above categories are based on the list of IoT security recommendations by ENISA.

In this paper, we ponder the possibility of developing a reasonably secure product based on the standardization work done by IETF. As far as the scope of the IETF work goes, we believe this is possible: standardization work has advanced to a point that there is typically no need for home-grown solutions (which are often less secure).

There are, however, challenges and gaps as well. At the time of writing some standardization work is not yet completed. In other cases, standardization work is ahead of implementations. Developers often face a lot of pain when integrating security libraries into their hardware of choice. In general, few security libraries are developer-friendly and available for use with the large number of IoT operating systems. More resources are needed to implement high-quality embedded libraries. Even if a protocol specification is sound, it is only secure in practice when implemented without fundamental bugs. In particular, solid implementation of crypto requires highly specialized work since subtleties in specifications are common, and naive protocol implementation (not to mention bugs) can lead to side-channel vulnerabilities. Recommendations thus point towards using well tested, widely used libraries (instead of home-grown implementations). Since even well-tested security protocol implementations have bugs it is important to not underplay the importance of firmware update, which is a crucial security feature of a product.

Last but not least, it is worth noting that there is still configuration and optimization potential within each of the IETF-developed security protocols, assuming an engineer has security know-how and is familiar with embedded development.

References

[1] C. Bormann, M. Ersue, and A. Keranen, *Terminology for constrained-node networks*. Internet Engineering Task Force RFC 7228, 2014.

[2] O. Hahm, E. Baccelli, H. Petersen, and N. Tsiftes, *Operating Systems for Low-end Devices in the Internet of Things: a Survey*. IEEE Internet of Things Journal, 2016.

[3] European Union Agency For Network And Information Security (ENISA), *Baseline Security Recommendations for IoT*, 2017. Online: https://www.enisa.europa.eu/publications/baseline-security-recommendations-for-iot

[4] E. Rescorla, B. Korver, *Guidelines for writing RFC text on security considerations*. Internet Engineering Task Force RFC 3552, 2003.

[5] A. Cooper, et al. *Privacy Considerations for Internet Protocols*. Internet Engineering Task Force RFC 6973, 2013.

[6] S. Farrell, H. Tschofenig, *Pervasive monitoring is an attack*. Internet Engineering Task Force RFC 7258, 2014.

[7] H. Tschofenig, T. Fossati, *Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS) Profiles for the Internet of Things*. Internet Engineering Task Force RFC 7925, 2016.

[8] E. Rescorla, *The Transport Layer Security (TLS) Protocol Version 1.3*, Internet Engineering Task Force RFC 8446, 2018.

[9] J. Schaad, *CBOR Object Signing and Encryption (COSE)*, Internet Engineering Task Force RFC 8152, 2017.

[10] G. Selander, et al. *Object Security for Constrained RESTful Environments (OSCORE)*. IETF Internet Draft (work-in-progress) draft-ietf-core-object-security, 2019.

[11] L. Seitz, et al. *Authentication and Authorization for Constrained Environments using the OAuth 2.0 Framework (ACE-OAuth)*. IETF Internet Draft (work-in-progress) draft-ietf-ace-oauth-authz, 2018.

[12] H. Tschofenig, N. Smith, *Credential Management for Internet of Things Devices*, OMA Specworks, IPSO Alliance, 2017. Online: https://www.omaspecworks.org/wp-content/uploads/2018/03/IPSO-IoT-Credential-Management_Final.pdf

[13] B. Sarikaya, M. Sethi, D. Garcia-Carillo, *Secure IoT Bootstrapping: A Survey*. IETF Internet Draft (work-in-progress) draft-sarikaya-t2trg-sbootstrapping, 2018.

[14] E. Lear, et al. *Manufacturer Usage Description (MUD)*. Internet Engineering Task Force RFC 8520, 2018.

[15] B. Moran, H. Tschofenig, *A CBOR-based Firmware Manifest Serialisation Format*. IETF Internet Draft (work-in-progress) draft-moran-suit-manifest, 2018.

[16] M. Pei, et al. *The Open Trust Protocol (OTrP)*. IETF Internet Draft (work-in-progress) draft-ietf-teep-opentrustprotocol, 2018.