AC-SIF: ACE Access Control for Standardized Secure IoT Firmware Updates

Joel Höglund, Anum Khurshid, Shahid Raza
RISE Research Institutes of Sweden
Isafjordsgatan 22, 16440 Kista, Stockholm
{joel.hoglund, anum.khurshid, shahid.raza}@ri.se

***Abstract*—Globally identifiable, internet-connected embedded systems can be found throughout critical infrastructures in modern societies. Many of these devices operate unattended for several years at a time, which means a remote software update mechanism should be available in order to patch vulnerabilities. However, this is most often *not* the case, largely due to interoperability issues endemic to the Internet of Things (IoT). Significant progress toward global IoT compatibility has been made in recent years. In this paper, we build upon emerging IoT technologies and recommendations from IETF SUIT working group to design a firmware update architecture which (1) provides end-to-end security between authors and devices, (2) is agnostic to the underlying transport protocols, (3) does not require trust anchor provisioning by the manufacturer and (4) uses standard solutions for crypto and message encodings. This work presents the design of a firmware manifest (i.e., metadata) serialization scheme based on CBOR and COSE, and a profile of CBOR Web Token (CWT) to provide access control and authentication for update authors. We demonstrate that this architecture can be realized whether or not the recipient devices support asymmetric cryptography. We then encode these data structures and find that all required metadata and authorization information for a firmware update can be encoded in less than 600 bytes with this architecture.**

***Index Terms*—ACE; SUIT; COSE; IoT; security.**

I. Introduction

The need for secure firmware updates in the Internet of Things (IoT) has been apparent for several years. Seen in a longer perspective, the IoT is still in its infancy, and the current situation regarding software updates for IoT is comparable to personal computers in the 1990s [1]. Most embedded systems do not have a system in place for remote software updates, which means device operators must manually download and install them on each device [2]. As a result, many IoT deployments are simply never updated, even after vulnerabilities are found, because the labor cost outweighs the perceived benefit.

The IoT is traditionally characterized by a lack of standards, which incentivizes companies to develop proprietary solutions [3]. For example, Texas Instruments (TI) and Amazon Web Services introduced an update framework specifically for TI devices running Amazon FreeRTOS [4]. This approach leads to *vendor lock-in*, where each manufacturer offers mutually incompatible software ecosystems. This ultimately hurts the industry and consumers: it prevents end users to freely compose networks of devices from different manufacturers, and it creates prohibitively high costs for smaller companies to enter the market and compete, whose only option might be to become sub-providers to providers of proprietary ecosystems. Embedded systems come with a wide range of hardware, operating systems, capabilities and constraints, which should not be a reason for incompatibility. New standards, such as 6LoWPAN [5], DTLS [6], CoAP [7] and OSCORE [8], enable secure IPv6 networking on devices with only tens of kilobytes of RAM, resulting in constrained devices being globally addressed with internet protocols. Although the *content* of firmware updates varies between devices, an industry-wide standard for the distribution of these updates enables the desired interoperability, where the same update infrastructure can serve multiple, or heterogeneous, deployments, instead of requiring several custom solutions. The need for common standards in the area and its challenges is identified within the Internet Engineering Task Force (IETF) standard [9] leading to the formation of the Software Updates for IoT (SUIT) working group. To have long term impact, a secure update framework must support existing embedded systems and systems which have yet to be conceived. The working group describes a firmware update solution consisting of three components: a mechanism for transporting updates, a *manifest* containing metadata about the update, and the firmware image [10]. SUIT suggests the following design requirements for the update architecture: (i) agnostic to firmware image distribution, (ii) friendly to broadcast delivery, (iii) built on state-of-the-art security mechanisms, (iv) not vulnerable to rollback attacks,

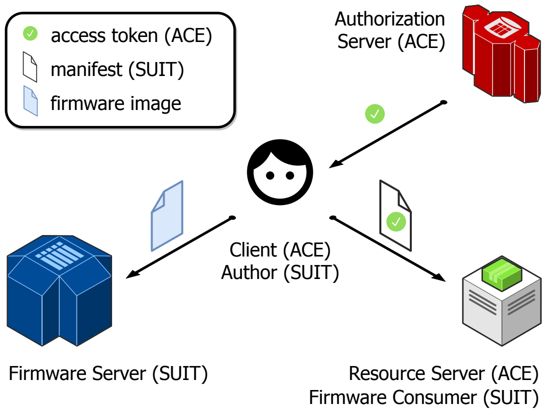
✓ access token (ACE)
manifest (SUIT)
firmware image
Authorization Server (ACE)
Client (ACE) Author (SUIT)
Firmware Server (SUIT)
Resource Server (ACE) Firmware Consumer (SUIT)

Fig. 1. Our proposed firmware update architecture, combining ACE authorization mechanisms with proposed Software Updates for IoT (SUIT) solutions.

(v) minimal impact on existing firmware formats, (vi) enables robust permissions controls and (vii) diverse modes of operation.

Among the challenges of specifying and implementing an architecture to meet these requirements are how to solve access control and credential management. Without adequate security, an update mechanism becomes an attack vector in itself, and can be used to install malware or simply brick devices. Hence, IoT devices must be able to verify the origin and integrity of the firmware specified in the manifests, and the permissions of the update author. In this paper, we present a solution to this problem based on the Authentication and Authorization for Constrained Environments (ACE) framework. A high level illustration is shown in Figure 1. The main contributions of this work are presented through the following sections:

IV A firmware manifest design and update architecture, based on the ACE framework and SUIT recommendations, to provide both authentication and authorisation mechanisms for secure updates.

V Proposals for the use of CBOR Web Tokens (CWT) for Proof-of-Possession (PoP) in the update architecture.

VI An implementation and evaluation of the manifest and access tokens described in Sections IV and V.

The rest of the paper is organized as follows. The IoT security standards providing the basis of our update architecture are discussed in Section II. Related work is presented in Section III. In Section VII we discuss the security consideration of the proposed architecture, and conclude the paper in Section VIII.

II. Background and Threat Model

This section presents IoT security standards and protocols which form the basis of our proposed update architecture, followed by the assumed threat model.

We briefly summarize the Constrained Application Protocol (CoAP), Concise Binary Object Representation (CBOR), CBOR Object Signing and Encryption (COSE), Public Key Infrastructure (PKI), Authentication and Authorization for Constrained Environments (ACE) and CBOR Web Tokens (CWT).

A. The Constrained Application Protocol (CoAP)

Typical constrained devices are sensors, actuators or both. Heavy computations are offloaded to more powerful devices, while the nodes receive commands, transmit sensor readings and perform periodic tasks. These types of networks are well-suited to RESTful services, but traditional web protocols like HTTP incur an unacceptable overhead for small devices. This has been alleviated by CoAP, a lightweight version of HTTP using binary message encodings rather than human-readable formats and running on top of UDP instead of TCP.

B. CBOR encoding and COSE

In web applications, where computing resources are plentiful and human readability is advantageous, data representations such as XML and JSON have widespread use. For the IoT, CBOR has become the preferred encoding scheme as it

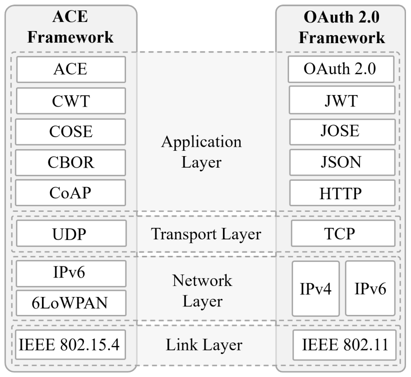

Fig. 2. Network protocols for token-based authentication in the IoT (ACE) along with their web counterparts (OAuth2.0).

is compact, offers lower message overhead and is designed for efficiency [11]. In applications requiring cryptographic operations, COSE is a standard with increasing usage in IoT [12]. COSE provides a standardized format for encryption, signing and Message Authentication Codes (MAC).

C. Public Key Infrastructure (PKI)

PKI provides the basis of authentication and access control in modern networked systems, by managing the distribution and revocation of digital certificates. These certificates rely on asymmetric cryptography, which is computationally demanding for constrained devices. New standards and proposals for lightweight certificate enrollment targeting IoT have provided important PKI building blocks [13][14]. Experimental analyses of these protocols have demonstrated that PKI enrollment is now within the capabilities of constrained devices [15][16]. However, many existing IoT networks still rely on Pre-Shared Keys (PSK), shared with all parties the devices communicate with, or raw public keys (i.e., asymmetric cryptography without attached certificates).

D. The ACE Framework

ACE is an authentication and authorization framework for IoT, built on CBOR, COSE, CoAP and OAuth 2.0 [17]. Clients request access to protected resources from an Authorization Server (AS). If successful, the AS grants the client a token which is bound to a secret key in the client's possession, a specific resource and an expiration date. This token is then used as proof of authorization when accessing the Resource Server (RS). The RS can optionally send an *introspection* request to the AS to confirm the token's validity. A network stack with ACE is shown in Figure 2. In the context of our proposed architecture, the recipient IoT devices act as the RS, as illustrated in Figure 1.

There exists a number of proposals for profiling ACE to be used together with DTLS [18], OSCORE [14] or MQTT [19].

E. CBOR Web Tokens (CWT)

The ACE framework uses CWT instead of their OAuth counterpart, JSON Web Tokens (JWT) [20]. A *token* is essentially a small, serialized object containing *claims* about a *subject*, with some cryptographic guarantees generated by the *issuer* (i.e., the AS). The precise encoding of CWT claims are use-case dependent, but all signatures, MACs and encryption are done following COSE format specifications. *Access tokens* are bound to a key known to the token bearer. These are known as Proof-of-Possession (PoP) keys, and the semantics of binding them to CWTs and requesting them through ACE are described in two separate documents, [21] and [22].

F. Threat Model

Our assumptions on the capabilities of an attacker follow the Dolev-Yao adversarial model [23]. An attacker can eavesdrop and record sent messages, and inject messages into the communication. We assume that the adversary cannot break chryptographic functions, and does not have direct access to tampering with the IoT devices.

III. Related Work

Firmware updates can be grouped into two categories: image-based updates and differential updates. A 2017 survey among embedded software engineers found that almost 60% of respondents had a way of remotely updating their products and all of them used systems developed in-house, with a clear preference for image-based updates [2]. Bootloaders that utilize this approach, such as *MCUboot* [24], partition the device ROM into two sections – one for the old image and one for the new – in a way that a backup exists if the new firmware fails to boot. Differential firmware updates are far more diverse, encompassing module-based approaches [25][26], binary patching [27], binary compression [28], and more. Our work regards the secure distribution of firmware updates, and is agnostic to the firmware content or installation method.

A. Update Distribution Architectures

Software updates on systems with relatively few resource constraints are done via package managers, such as RPM or dpkg, and various commercial app stores. The trust anchors required to verify updates with PKI operations, such as code signing, are pre-installed in the operating system. A 2010 paper argued that because update architectures are an attractive target to attackers, recipients should never rely on a single signature [29]. Instead, the authors advocated for a (t, n) signature threshold scheme, whereby a recipient will not accept an update unless t out of n trusted signers have provided a signature. A profile of this scheme for constrained IoT was later proposed in 2018 [30]. Devices would be provisioned with the Original Equipment Manufacturer (OEM) certificate and trust anchors. The OEM would send signed update metadata to a device owner's *domain controller* server. This server would then sign and forward the message to the end devices; hence the update is $(2, 2)$ in the (t, n) notation.

Code signing by firmware update authors presents a problem for the IoT. In order for devices to verify the signatures, they must be provisioned with a list of authorized authors and their trust anchors. Moreover, update authors (for instance the OEM) are likely to be from outside the device owner's organization, and the device's lifetime may exceed that of the validity period of the update author's certificate which was available when initially deploying the IoT device. Our work solves these problems by incorporating a token-granting Authorization Server, which is capable of handling all certificate-based authentication on behalf of the IoT devices.

B. Software defined IoT

An approach to software updates for IoT is presented in [31], where more powerful devices act as controllers for more constrained IoT devices, building upon earlier work to define software defined networks for IoT [32]. This approach can offer solutions for heterogeneous networks which include both more powerful devices and devices which are themselves too constrained to act as fully independent endpoints, but does not address questions of standardisation.

C. Ongoing Standardisation Work

Key points of providing well specified mechanisms for secure software updates, are to achieve long time support capabilities and limit the risks of reliance on proprietary systems. Hence proposals for solutions need to relate to the ongoing standardisation efforts in the area. The SUIT working group within IETF has produced three core documents: one RFC describing the SUIT architecture [33], one RFC on a firmware manifest information model [34] and one draft specifying a proposal for a manifest format [35]. The proposal describes one instantiation of firmware manifests with CBOR/COSE encoding. It includes a new scripting language and recommendations that a series of commands should be embedded in SUIT manifests for firmware installation. This approach has its drawbacks, most notably the steep increase in parser complexity, which is likely to deter some vendors from adopting the standard. Including scripts in the manifest would also introduce new security vulnerabilities. The proposed scripting format contains instructions to verify firmware digests and check update compatibilities. This generates new issues about error handling, and how the device should proceed if an update author neglects to include critical security checks in the installation script. Our work defines a set of procedures to be followed by all manifest recipients; the manifest itself contains no instructions. The SUIT documents do not, however, describe how manifest encryption keys are to be distributed, nor how recipient devices are meant to verify author permissions. With the exception of scripting support, our manifest design follows the recommendations stated in these documents, and extends it by including lightweight solutions for authorization.

A 2019 paper by Zandberg et al. was the first to provide an implementation and performance analysis of a SUIT firmware manifest [36]. The work focused primarily on the RAM, ROM and CPU overhead incurred based on the choice of signing algorithm used for the manifest. Our work, in contrast, is focused specifically on how a SUIT manifest must be encoded to support token-based access control and key distribution, and considers both PSK and certificate-based use-cases. A recent survey on IoT update solutions shows that the study of SUIT related solutions is so far in its infancy, with only one other work mentioned besides the Zandberg et al. paper [37]. The short paper by Hernández-Ramos et al. discuss update related challenges. They conclude that the SUIT proposals might benefit from being aided by blockchain based mechanisms, which illustrates their complementary approaches [38].

D. Lightweight Machine-to-Machine

The Lightweight Machine-to-Machine (LwM2M) protocol is a device management protocol targeting IoT. The versions of the protocol since 2018 include a firmware update object [39]. This specification is similar to the SUIT model as it supports a *push* or *pull* architecture for firmware metadata, and firmware images can either be packaged with the metadata or retrieved from another server. However, security considerations are explicitly left outside the scope and no threat model is described. Access control, authentication and confidentiality are left entirely to the transport and application layer security mechanisms. This means that LwM2M is not a competitor to the SUIT proposals, but rather a possible framework in which the update solutions could be used. Early attempts in this directions have been reported in [40].

IV. Proposed Firmware Update Architecture

The communication architecture proposed in SUIT is flexible in a way that updates can be triggered either by the devices or the firmware/update authors (i.e., *push* or *pull*). The manifests can be distributed with or without the corresponding firmware images [33]. Our proposed architecture abides by these principles, but deviates in the way authors are authenticated and firmware is verified. SUIT states that a manifest should be directly signed by its author. This requires the provisioning of trust anchors and legitimate author identities. Moreover, the most constrained devices which still rely on symmetric keys (i.e., PSK) lack the ability to verify digital signatures. We approach this as an access control problem and provisioning devices with a list of trusted authors before deployment is insufficient for a number of reasons, such as:

- Author certificates may expire or are revoked.
- Original trusted Update Authors may fail to issue updates (e.g., when devices outlive their warranty).
- Device owners may not want to accept all updates issued by the manufacturer.

Hence we conclude that authentication is not sufficient for authorization. To address these concerns, we propose integrating the SUIT communication model with access control mechanisms provided by the ACE framework. This solution would allow device operators to centrally manage the list of authorized Update Authors (UA), and could be realized entirely using existing standard-based building blocks. Additionally, our proposed architecture can be realized whether or not the recipient IoT devices can verify digital signatures.

Combining SUIT and ACE results in the architecture illustrated in Figure 1. The recipient IoT devices act as *firmware consumers* from the SUIT perspective. Update Authors (UA) in SUIT play the role of the *client* in ACE (i.e., the entity requesting tokens). The client requests access to the firmware/update from the Authorization Server (AS). Finally the firmware updates are stored at, and can be downloaded from, a SUIT *firmware server*.

A. Authorization Tokens

A simple approach to distributing firmware updates with ACE would be to use one of the mentioned proposed profiles of the framework for secure channel establishment (with DTLS, OSCORE or MQTT). With an encrypted and mutually authenticated channel between the Update Author and recipient, manifests and images would not require further signatures or authentication codes. However, to enable a larger range of use-cases, firmware manifests must be standalone verifiable objects [9]. In our proposed update architecture, tokens are issued to the UA simply to authorize the distribution of manifests. The manifests themselves are authenticated and (partially) encrypted, and can be sent over any channel.

An ACE exchange always begins with establishing a security context between the client (i.e., UA) and the Authorization Server (AS). At this time, the AS authenticates the client and verifies their permissions to distribute updates before issuing an *access token*. If a symmetric PoP key is requested, it will be sent to the client over this secure channel. *Access tokens* are not required for the distribution of firmware images. Instead, the manifests contain a secure message digest of the corresponding image. This ensures integrity, and allows devices to retrieve firmware images from another server. The firmware retrieval could take place over an encrypted channel, or a combination of untrusted channels and encryped firmware images, depending on the confidentiality needs. We leave the details of this outside the scope of our architecture.

Our update architecture leverages the ACE framework for the provisioning of CBOR Web Tokens for PoP. There is some flexibility in how these tokens are protected and authenticated with COSE, which is discussed in Section V. The CWT standard defines a set of common claims to include in each token, but leaves the precise meaning of the fields up to the particular use-case. We use four of these and define them as:

`iss`	: issuer i.e., the URI of the AS server
`aud`	: audience i.e., the recipient device class's UUID
`iat`	: issued at i.e, the start of the *access token's* validity
`exp`	: expiration i.e., the end of the *access token's* validity

In addition, all tokens contain the confirmation field (`cnf`) which contains the PoP key, following the specification in [21].

B. Manifest Distribution

Our proposed architecture is designed to support both image-based and differential updates with dependencies. In the latter case, recipient devices must parse the dependency list, retrieve corresponding manifests, and parse their dependency lists (illustrated in Figure 3). Installing updates often requires devices to reboot, and potentially lose track of the state in the update process. We propose that devices query a known

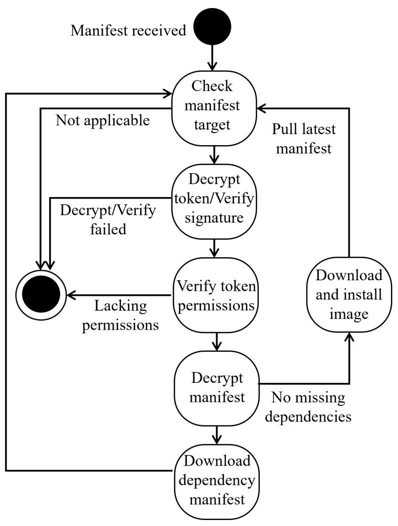

Fig. 3. Procedure followed by recipient devices for manifest dependency tree traversal and firmware update installation.

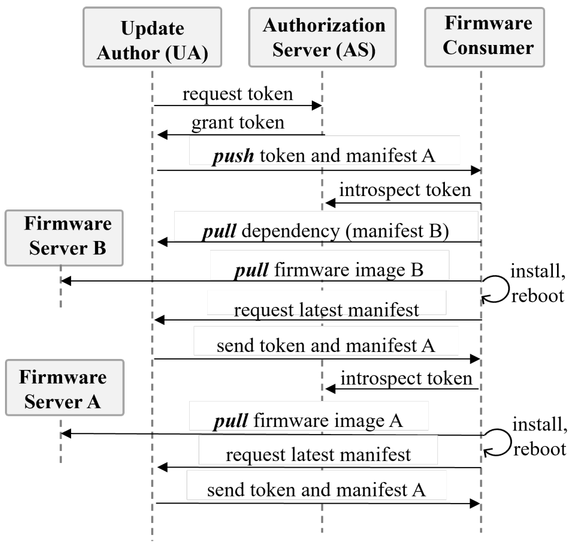

Fig. 4. Update sequence diagram for one possible use-case. The Update Author (UA) establishes a secure channel with the AS before pushing firmware manifest A to a recipient device. Since firmware update A is dependent on firmware update B, the device pulls the dependency list and parses it. Note that token introspection is an optional step.

manifest distributor at startup and request the latest manifest and corresponding access token. The device will know the update is complete when it receives a manifest matching its current firmware.

SUIT describes three categories of update architectures: *server-initiated*, *client-initiated* and *hybrid* updates. The recursive process for dependency installation used in our architecture is categorized as a client-initiated update. Figure 4 depicts interactions between actors for an update with a single dependency. The flow is server-initiated, for the cases where the update author has a known access path to the IoT device, but could easily be turned into client initiated through adding a polling step by the IoT device.

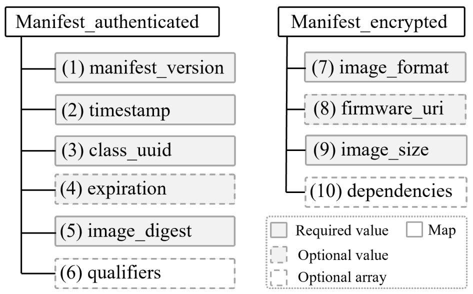

Fig. 5. Our proposed firmware manifest structure. The manifest is encoded as two separate CBOR maps, with the integer key values indicated in parentheses.

C. Manifest Design

We propose encoding firmware manifests as two separate CBOR maps: one containing information about the intended recipient of the update and another containing information about dependencies and image contents. The latter is encrypted, and both are authenticated in a single operation using an Authenticated Encryption with Associated Data (AEAD) algorithm. With this design, it is possible for a UA to broadcast an access token and manifest to a fleet of IoT devices, and any devices to which the update does not apply can quickly ascertain this without performing any cryptographic operations. Hence it is in line with SUIT recommendations to keep the update mechanism broadcast friendly. It is sensible to encrypt information about the image contents, in order to conceal information that is useful to an adversary attempting to gain insights into the software running on devices and its potential vulnerabilities. This includes the dependencies and the exact firmware URI.

In accordance with SUIT's recommendations, device classes representing the target IoT devices are given a 128-bit Universally Unique Identifier (UUID) [41], which is present through the manifest's `class_uuid` field. In our proposed architecture, devices ascertain whether the source of the manifest is authorized to issue updates by comparing this field to the `aud` value in the accompanying *access token*. A timestamp is mandatory in order to prevent rollback attacks, in which an attacker replays an earlier, legitimate firmware manifest with known vulnerabilities. IoT devices must verify that a manifest is issued more recently than their current firmware version. By storing the included timestamp of the current firmware version, a simple ordering check is sufficient to determine the temporal relation between manifests, and does not require access to a well synchronized clock.

The hash of the corresponding firmware image is included in the `image_digest` field. The URI of the firmware server can be specified in the encrypted `firmware_uri` field unless the location is already known to the devices. To handle use-cases where only devices with certain old firmware versions require a patch, the manifests optionally include a `qualifiers` list. This contains a list of firmware digests that a device must already have installed for the update to apply; otherwise it is discarded. The encrypted

TABLE I
CRYPTOGRAPHIC ALGORITHMS EXECUTED BY THE RECIPIENT FOR EACH APPROACH DESCRIBED IN SECTION V.

	AEAD	ECDSA	ECDH	KDF
A	manifest, token			
B	manifest, token	token	manifest	
C	manifest	token	manifest	manifest
D	manifest	token	manifest	

`dependencies` field indicates a list of firmware images which must be installed before installing the present one. This enables differential updates and is handled as per Figure 3.

D. COSE Wrappers

Our proposed manifest is designed for AEAD algorithms, several of which are supported natively by the COSE standard. These algorithms take a Content Encryption Key (CEK), a plaintext and some Additionally Authenticated Data (AAD) as inputs, and produce a ciphertext as output. The unencrypted portion of our manifest design is used as the AAD, and the encrypted portion forms the plaintext. The resulting ciphertext is encapsulated in a `COSE_Encrypt` object. In total, a recipient IoT device will receive three separate CBOR-encoded objects, all of which must be valid in order to accept the update: the token, the AAD, and the COSE-wrapped encrypted manifest data. The `recipients` field in a COSE wrapper is used to encipher the CEK with Key Encryption Keys (KEK) known only to the intended recipients. There are several ways to derive this KEK, which is discussed in further detail in the upcoming sections.

V. AUTHENTICATION OPTIONS

Access control and cryptography in the IoT must be discussed in the context of device capabilities; this ultimately determines the available options. To this end, we group devices into two broad categories: (i) devices that rely entirely on PSK, (ii) devices that possess unique asymmetric key pairs (e.g., digital certificates) and can verify digital signatures. In this section we describe four distinct applications of COSE for protecting firmware manifests and the corresponding access tokens. Only the first option is applicable to devices restricted to only using PSK; the others are applicable wherever asymmetric cryptography is available, where devices are provisioned with certificates via a PKI. The message overhead of each option is analyzed in Section VI.

A. Symmetric PoP Key with PSK

Reliance on PSK for security precludes the use of digital signatures and Diffie-Hellman key exchange algorithms. In addition, since the network's security is based entirely on the secrecy of the PSK, these keys should never be sent to a third party (i.e., an Update Author). We address these constraints by issuing a unique symmetric PoP key with each access token. The key is sent to the author over its secure channel with the AS, and is also included in the `cnf` field of the access token. The token is encapsulated in a `COSE_Encrypt0` object using the network PSK for encryption by the AS, and the manifest is encapsulated in another using the PoP key. It should be noted that this approach is subject to attack vectors not present in the other authentication methods (see Section VII).

B. Symmetric PoP Key

Symmetric PoP keys are an option also where asymmetric cryptography is available. We suggest the following approach, which is not conventional, but well-suited to this particular application. The AS generates the PoP key and encrypts it *with itself* in a `COSE_Encrypt0` object. This is then included in the `cnf` field of the access token, and the token is encapsulated as the payload of a `COSE_Sign1` object signed by the AS. (The following later verification of this signature is what requires asymmetric cryptography capabilities by the receiving IoT device.) The UA distributes the CEK to recipient devices via the `recipients` field in the manifest's COSE wrapper. Recipients can then verify that this CEK is the one contained in the signed token by decrypting the `cnf` field. The motivation for this approach is to avoid including any recipients in the token itself, as this would require the AS to have knowledge of the intended recipients' public keys. The UA must know the recipients' public keys in order to encipher the CEK.

C. Asymmetric PoP Key, Direct Key Agreement

In the case of asymmetric PoP keys, the `cnf` field of the CWT contains the COSE encoding of a public key belonging to the UA. The token is then encapsulated in a `COSE_Sign1` wrapper. The UA now has two options for deriving a CEK for the manifest. The first is through direct key agreement. This type of algorithm applies a key exchange protocol – in this case Elliptic Curve Diffie-Hellman (ECDH) – and a Key Derivation Function (KDF) to generate the CEK directly. The author must use the key pair bound to the token to prove their authorization.

D. Asymmetric PoP Key, Key Wrap

The second asymmetric PoP key approach is to use the key derived through ECDH as a Key Encryption Key (KEK) to encipher a randomly-generated ephemeral CEK. These two approaches have implementation nuances and security considerations which are discussed in Sections VI and VII. Table I summarizes the cryptographic operations that recipient devices much perform in order to process manifests and tokens with each of the four described authentication options.

VI. IMPLEMENTATION

The encoding scheme for each authentication options discussed in the previous section is shown in Table II. In this section, we generate firmware manifests and access tokens for each of the four cases. The purpose of this exercise is both to demonstrate the viability of the proposed architecture, and to evaluate the differences in storage and transmission overhead.

TABLE II
COSE WRAPPERS FOR EACH MANIFEST-TOKEN COMBINATION DESCRIBED IN SECTION V.

	Authentication	Manifest	Token
A	PSK	`COSE_Encrypt0`	`COSE_Encrypt0`
B	Symmetric PoP key	`COSE_Encrypt`	`COSE_Sign1`
C	Asymmetric PoP key	`COSE_Encrypt`	`COSE_Sign1`
D	Asymmetric PoP key	`COSE_Encrypt`	`COSE_Sign1`

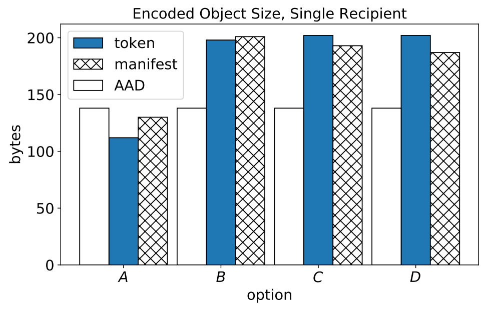

Fig. 6. Encoded sizes of the **manifest**, **token** and **AAD** for each approach in Section V.

A. Profile and Assumptions

For our implementation and analysis, we populate the manifest fields illustrated in Figure 5 with example data. In order to do so we make the following assumptions:

- Images are identified with 32-byte digests.
- Timestamps are represented in relative time.
- The manifest has two qualifiers and two dependencies.
- The firmware server URI is *coaps://example.com*.
- The Authorization Server URI is *coaps://example.com*.

The authenticated and encrypted example manifest components are 138 and 116 bytes, respectively, after CBOR serialization. COSE offers a variety of algorithms with a range of key sizes for each cryptographic operation. For our implementation, we have chosen the following:

- ECDSA signatures with 256-bit keys.
- AES-CCM with 128-bit keys, 64-bit tag and a 13-byte nonce for content encryption.
- AES 128-bit key wrap.
- ECDH Ephemeral-Static (ES).
- HMAC-Based Extract-and-Expand Key Derivation Function (HKDF) with SHA-256.

B. Results and comparison with other SUIT proposals

The update and authentication information is separated into three separate CBOR-encoded objects: the **token**, the encrypted **manifest** data, and the plaintext authenticated manifest data (a.k.a. the additionally authenticated data, or **AAD**). The results are shown in Figure 6. Option A has the smallest total size, with all three CBOR objects totalling 380 bytes. Option C has the largest footprint, totalling 537 bytes. Since the differences are relatively minor, the choice of method should be guided by the offered security properties, as discussed below in VII.

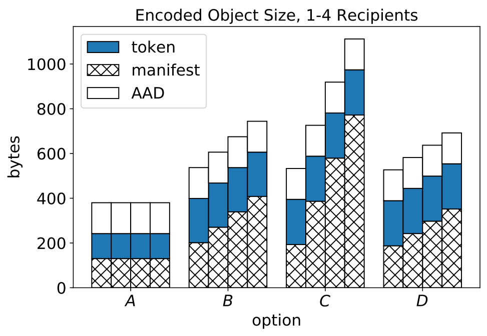

Fig. 7. Encoded sizes of the **manifest**, **AAD** and **token** when the update has multiple recipients.

In the most recent SUIT manifest proposal there are example manifest samples, which allow us to compare our proposals with the draft. In [35] the minimal manifest is only 237 bytes, but for example manifests with content similar to the sample used in our evaluation, the size is between 270 and 400 bytes. The main difference is the addition of our relatively large access tokens, since they are designed to be independent authorization tokens, compliant with ACE requirements. Given this added security functionality we find the added overhead to be clearly acceptable.

In some deployments, it may be preferable for UAs to upload both manifests and firmware images to a dedicated firmware server to be retrieved by devices at a later time. This is feasible within our framework as long as the corresponding access token is stored alongside the manifest. The storage overhead for manifests encoded with multiple recipients is shown in Figure 7. The plot shows the encoded size of the required objects for 1-4 recipients for each authentication method. In Option A, all recipients receive an identical manifest since they possess the same PSK. In Option B and D, the CEK is wrapped, each additional recipient only requires an additional entry in the recipients field of the COSE object. Option C, however, derives a unique CEK for each recipient, which means the manifest must be re-encrypted for every target device, making C the least efficient option for broadcast scenarios.

VII. Security Considerations

The proposals in this paper are founded on well-vetted standards and encryption algorithms. However, there are protocol details that must be fully understood in order to avoid security lapses. A malicious firmware image could permanently disable expensive hardware and compromise an entire network, therefore, great care must be taken to ensure an update distribution mechanism does not become an attack vector in itself.

A. Non-Repudiation

The PSK use-case described in Section V-A precludes any guarantees for the access token. Since the AS uses a symmetric key known to all recipients, an adversary with control over any device would be capable of generating fraudulent tokens and PoP keys. This is problematic, although PSK networks are already subjected to similar risks. Any adversary in possession of a PSK could cause significant damage and disruption, even without the ability to issue firmware updates. If a symmetric PoP key is used and the token is signed by the AS, as in Section V-B, non-repudiation is only guaranteed for the token, but not necessarily the manifest. The UA must encipher the PoP key for each recipient, so if any of the recipients are controlled by an adversary, that adversary would then be in possession of a valid token and the associated PoP key. The use of symmetric PoP keys also breaks end-to-end security between the author and recipients, because the key is known to the AS. The analysis presented in Section VI demonstrated that asymmetric PoP keys with Key Wrap has a similar overhead but without the risks, making that approach clearly preferable.

B. Key Agreement

The manifest exchange between the author and recipients is one-way, i.e., there is no nonce exchange or handshake like in DTLS or EDHOC. The manifest's CEK is either wrapped (Options B and D) or derived directly (Option C), as described in Section V from the author and recipients' key pairs. In COSE, ECDH key derivation comes in two types: Static-Static (SS) or Ephemeral-Static (ES). In the former case, the author of the COSE object declares that the CEK is either wrapped or derived from the key pairs bound to the author and recipient. In the latter case, the author of the COSE object provides an ephemeral key pair generated for a single encryption operation. ECDH-ES is generally safer to use, because even if an adversary obtains the author's private key, it is not usable for decryption of other manifests or impersonation of the author. It is therefore preferable for UAs to request access tokens bound to an ephemeral public key, not the public key found in their certificate.

C. Firmware Image Digests

Firmware manifests are only linked to firmware images via the inclusion of a secure message digest. If a weak algorithm with the possibility of a hash collision is used for this purpose, such as SHA-1, devices may be exposed to fraudulent images referenced by authentic manifests.

VIII. Conclusion

In this work we have presented an architecture based on existing standards, which can address the urgent need for secure firmware updates in the IoT. We have described the challenges and limitations of access control in constrained environments, and why a token-based framework, such as ACE, is a promising candidate solution. In addition, we have proposed encoding schemes for firmware manifests using the CBOR and COSE standards, and detailed how these would work in conjunction with CWT to provide authorized updates. Examples of these objects were encoded and the result totaled no more than 600 bytes for the firmware manifest data, including authentication and authorization.

Acknowledgments

This research is partially funded by the Swedish Foundation for Strategic Research (SSF) Institute PhD grant, the SSF aSSIsT project and by the H2020 CONCORDIA (Grant ID: 830927) project.

References

[1] B. Schneier, “The Internet of Things is Wildly Insecure–And Often Unpatchable,” January 2014. [Online]. Available: https://www.wired.com/2014/01/theres-no-good-way-to-patch-the-internet-of-things-and-thats-a-huge-problem/

[2] E. Stenberg. (2017, September) Key Considerations for Software Updates for Embedded Linux and IoT. [Online]. Available: https://www.linuxjournal.com/content/key-considerations-software-updates-embedded-linux-and-iot

[3] J. P. Vasseur and A. Dunkels, *Interconnecting Smart Objects with IP: The Next Internet*. Morgan Kaufmann, 2010.

[4] N. Lethaby, “A more secure and reliable OTA update architecture for IoT devices,” Texas Instruments, Tech. Rep., 2018. [Online]. Available: http://www.ti.com/lit/wp/sway021/sway021.pdf

[5] G. Montenegro, J. Hui, D. Culler, and N. Kushalnagar, “Transmission of IPv6 Packets over IEEE 802.15.4 Networks,” RFC 4944, Sep. 2007.

[6] “Datagram Transport Layer Security Version 1.2,” RFC 6347, Jan. 2012.

[7] Z. Shelby, K. Hartke, and C. Bormann, “The Constrained Application Protocol (CoAP),” RFC 7252, Jun. 2014.

[8] G. Selander, J. Mattsson, F. Palombini, and L. Seitz, “Object Security for Constrained RESTful Environments (OSCORE),” RFC 8613, RFC Editor, Tech. Rep. 8613, Jul. 2019. [Online]. Available: https://rfc-editor.org/rfc/rfc8613.txt

[9] H. Tschofenig and S. Farrell, “Report from the Internet of Things Software Update (IoTSU) Workshop 2016,” RFC 8240, RFC Editor, Tech. Rep. 8240, September 2017. [Online]. Available: https://tools.ietf.org/html/rfc8240

[10] B. Moran, M. Meriac, H. Tschofenig, and D. Brown, “A Firmware Update Architecture for Internet of Things Devices,” Internet Engineering Task Force, Internet-Draft draft-ietf-suit-architecture-05, Apr. 2019, work in Progress.

[11] C. Bormann and P. Hoffman, “Concise Binary Object Representation (CBOR),” Internet Requests for Comments, RFC Editor, RFC 7049, October 2013.

[12] J. Schaad, “CBOR Object Signing and Encryption (COSE),” RFC 8152, RFC Editor, Tech. Rep. 8152, Jul. 2017. [Online]. Available: https://rfc-editor.org/rfc/rfc8152.txt

[13] P. van der Stok, P. Kampanakis, M. Richardson, and S. Raza, “EST-coaps: Enrollment over Secure Transport with the Secure Constrained Application Protocol,” Internet Requests for Comments, RFC Editor, RFC 9148, April 2022.

[14] G. Selander, S. Raza, M. Furuhed, M. Vučinić, and T. Claeys, “Protecting est payloads with oscore,” Working Draft, IETF Secretariat, Internet-Draft draft-selander-ace-coap-est-oscore-05, May 2021. [Online]. Available: https://www.ietf.org/archive/id/draft-selander-ace-coap-est-oscore-05.txt

[15] Z. He, M. Furuhed, and S. Raza, “Indraj: Certificate Enrollment for Battery-powered Wireless Devices,” in *Proceedings of the 12th ACM Conference on Security and Privacy in Wireless and Mobile Networks*. ACM, 2019.

[16] J. Höglund, S. Lindemer, M. Furuhed, and S. Raza, “PKI4IoT: Towards public key infrastructure for the Internet of Things,” *Computers & Security*, vol. 89, 2020.

[17] L. Seitz, G. Selander, E. Wahlstroem, S. Erdtman, and H. Tschofenig, “Authentication and authorization for constrained environments (ace) using the oauth 2.0 framework (ace-oauth),” Working Draft, IETF Secretariat, Internet-Draft draft-ietf-ace-oauth-authz-46, November 2021.

[18] S. Gerdes, O. Bergmann, C. Bormann, G. Selander, and L. Seitz, “Datagram transport layer security (dtls) profile for authentication and authorization for constrained environments (ace),” Working Draft, IETF Secretariat, Internet-Draft draft-ietf-ace-dtls-authorize-18, June 2021. [Online]. Available: https://www.ietf.org/archive/id/draft-ietf-ace-dtls-authorize-18.txt

[19] C. Sengul and A. Kirby, “Message queuing telemetry transport (mqtt)-tls profile of authentication and authorization for constrained environments (ace) framework,” Working Draft, IETF Secretariat, Internet-Draft draft-ietf-ace-mqtt-tls-profile-17, March 2022. [Online]. Available: https://www.ietf.org/archive/id/draft-ietf-ace-mqtt-tls-profile-17.txt

[20] M. Jones, E. Wahlstroem, S. Erdtman, and H. Tschofenig, “CBOR Web Token (CWT),” RFC 8392, May 2018.

[21] M. Jones, L. Seitz, G. Selander, S. Erdtman, and H. Tschofenig, “Proof-of-possession key semantics for cbor web tokens (cwts),” Internet Requests for Comments, RFC Editor, RFC 8747, March 2020.

[22] L. Seitz, “Additional oauth parameters for authorization in constrained environments (ace),” Working Draft, IETF Secretariat, Internet-Draft draft-ietf-ace-oauth-params-16, September 2021. [Online]. Available: https://www.ietf.org/archive/id/draft-ietf-ace-oauth-params-16.txt

[23] D. Dolev and A. Yao, “On the security of public key protocols,” *IEEE Transactions on Information Theory*, vol. 29, no. 2, pp. 198–208, 1983.

[24] MCUboot contributors, “MCUboot,” https://github.com/mcu-tools/mcuboot, 2022.

[25] A. Dunkels, N. Finne, J. Eriksson, and T. Voigt, “Run-time dynamic linking for reprogramming wireless sensor networks,” in *Proceedings of the 4th International Conference on Embedded Networked Sensor Systems*, ser. SenSys ’06. New York, NY, USA: ACM, 2006, pp. 15–28. [Online]. Available: http://doi.acm.org/10.1145/1182807.1182810

[26] P. Ruckebusch, E. De Poorter, C. Fortuna, and I. Moerman, “Gitar,” *Ad Hoc Netw.*, vol. 36, no. P1, pp. 127–151, Jan. 2016. [Online]. Available: https://doi.org/10.1016/j.adhoc.2015.05.017

[27] Jaein Jeong and D. Culler, “Incremental network programming for wireless sensors,” in *2004 First Annual IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks, 2004. IEEE SECON 2004.*, Oct 2004, pp. 25–33.

[28] M. Stolikj, P. J. L. Cuijpers, and J. J. Lukkien, “Efficient reprogramming of wireless sensor networks using incremental updates,” in *2013 IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops)*, March 2013, pp. 584–589.

[29] J. Samuel, N. Mathewson, J. Cappos, and R. Dingledine, “Survivable key compromise in software update systems,” in *Proceedings of the 17th ACM Conference on Computer and Communications Security*, ser. CCS ’10. New York, NY, USA: ACM, 2010, pp. 61–72. [Online]. Available: http://doi.acm.org/10.1145/1866307.1866315

[30] N. Asokan, T. Nyman, N. Rattanavipanon, A. Sadeghi, and G. Tsudik, “Assured: Architecture for secure software update of realistic embedded devices,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 11, pp. 2290–2300, Nov 2018.

[31] N. Xue, D. Guo, J. Zhang, J. Xin, Z. Li, and X. Huang, “Openfunction for software defined iot,” in *2021 International Symposium on Networks, Computers and Communications (ISNCC)*, 2021, pp. 1–8.

[32] Y. Jararweh, M. Al-Ayyoub, A. Darabseh, E. Benkhelifa, M. Vouk, and A. Rindos, “Sdiot: A software defined based internet of things framework,” *Journal of Ambient Intelligence and Humanized Computing*, vol. 1, pp. 453–461, 08 2015.

[33] B. Moran, H. Tschofenig, D. Brown, and M. Meriac, “A firmware update architecture for internet of things,” Internet Requests for Comments, RFC Editor, RFC 9019, April 2021.

[34] B. Moran, H. Tschofenig, and H. Birkholz, “A manifest information model for firmware updates in internet of things (iot) devices,” Internet Requests for Comments, RFC Editor, RFC 9124, January 2022.

[35] B. Moran, H. Tschofenig, H. Birkholz, and K. Zandberg, “A concise binary object representation (cbor)-based serialization format for the software updates for internet of things (suit) manifest,” Working Draft, IETF Secretariat, Internet-Draft draft-ietf-suit-manifest-17, April 2022.

[36] K. Zandberg, K. Schleiser, F. Acosta, H. Tschofenig, and E. Baccelli, “Secure firmware updates for constrained IoT devices using open standards: A reality check,” *IEEE Access*, vol. 7, pp. 71 907–71 920, 2019.

[37] S. El Jaouhari and E. Bouvet, “Secure firmware over-the-air updates for iot: Survey, challenges, and discussions,” *Internet of Things*, vol. 18, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2542660522000142

[38] J. L. Hernández-Ramos, G. Baldini, S. N. Matheu, and A. Skarmeta, “Updating iot devices: challenges and potential approaches,” in *2020 Global Internet of Things Summit (GIoTS)*, 2020, pp. 1–5.

[39] “Lightweight Machine to Machine Technical Specification 1.0.2,” Open Mobile Alliance, Tech. Rep. OMA-TS-LightweightM2M-V1_0_2-20180209-A, February 2018.

[40] IETF. Ietf hackathon: Software / firmware updates for iot devices. IETF. [Online]. Available: https://datatracker.ietf.org/meeting/111/materials/slides-111-suit-suit-hackathon-report-00

[41] P. J. Leach, R. Salz, and M. H. Mealling, “A Universally Unique IDentifier (UUID) URN Namespace,” RFC 4122, Jul. 2005. [Online]. Available: https://rfc-editor.org/rfc/rfc4122.txt