ACE of Spades in the IoT Security Game: A Flexible IPsec Security Profile for Access Control

Santiago Aragon*†, Marco Tiloca†, Max Maass*, Matthias Hollick* and Shahid Raza†
*Technische Universität Darmstadt, Secure Mobile Networking Lab
Mornewegstr. 32, 64293 Darmstadt, Germany
{saragon, mmaass, mhollick}@seemoo.tu-darmstadt.de
†RISE SICS, Security Lab
Isafjordsgatan 22, SE-16440 Kista, Sweden
{marco.tiloca, shahid.raza}@ri.se

***Abstract*—The Authentication and Authorization for Constrained Environments (ACE) framework provides fine-grained access control in the Internet of Things, where devices are resource-constrained and with limited connectivity. The ACE framework defines separate profiles to specify how exactly entities interact and what security and communication protocols to use. This paper presents the novel ACE IPsec profile, which specifies how a client establishes a secure IPsec channel with a resource server, contextually using the ACE framework to enforce authorized access to remote resources. The profile makes it possible to establish IPsec Security Associations, either through their direct provisioning or through the standard IKEv2 protocol. We provide the first Open Source implementation of the ACE IPsec profile for the Contiki OS and test it on the resource-constrained Zolertia Firefly platform. Our experimental performance evaluation confirms that the IPsec profile and its operating modes are affordable and deployable also on constrained IoT platforms.**

I. Introduction

The Internet of Things (IoT) refers to network scenarios where billions of devices communicate over IP networks and are available on the Internet. This includes everyday objects and appliances, and has been constantly fostering a number of use cases and business opportunities, from sensor and actuator networks to smart buildings, from monitoring of critical infrastructures to controlled resource sharing. As more and more applications are being developed, the IoT is expected to have a huge impact on the way we live and work.

At the same time, security plays a fundamental role, even during this transition process. In fact, ensuring security in IoT scenarios is of vital importance to counteract information breaches and service dysfunctions, which may result in severe performance degradation and privacy violations, or even threaten safety of people and infrastructures. Securing the IoT is thus vital to ensure its successful deployment and adoption.

However, unlike in traditional networks, IoT devices are typically resource-constrained, i.e. equipped with limited resources. That is, they are scarce as to processing power, storage and energy availability, often being battery-powered. Besides, most IoT devices are wirelessly connected over low-power and lossy networks, thus exhibiting limited connectivity and availability. Also, they often lack traditional user interfaces, and are likely deployed in unattended environments. As a result, protecting billions of IoT devices with traditional approaches is challenging, which fosters the development of novel security solutions suitable for the IoT. Yet, many of these solutions do not base on established standards and are difficult to scrutinize in terms of their security guarantees.

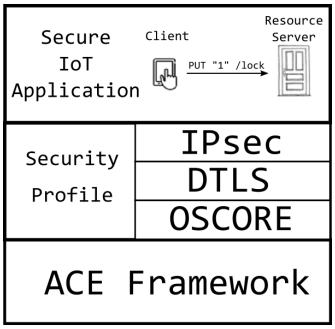


Fig. 1: IoT applications can secure their communications by using the ACE framework and the different security profiles.

The first security challenge consists in efficiently enabling secure communication and message exchange. Due to the resource-constrained nature of typical IoT devices, their great heterogeneity and their large-scale deployment, it is not feasible to rely on solutions for traditional network environments. To this end, a number of secure communication protocols for the IoT are available and have been increasingly adopted in constrained environments. In particular, [1] and [2] show how 6LoWPAN header compression mechanisms optimize security protocols to be deployable in resource-constrained networked scenarios. However, it can be very difficult to provision millions, or even billions, of resource-constrained IoT devices with the cryptographic keys necessary to securely communicate and operate. Even the establishment of secure

sessions based on pre-shared symmetric keys can easily result in hard-to-manage and poorly scalable key distribution.

The second critical security aspect concerns authorization and access control. Typically, a *Client* wants to access a resource hosted on a *Resource Server* (RS), which is often deployed as a resource-constrained device. This requires the Client and the RS to mutually authenticate, and must permit the RS to verify Client requests as previously authorized. In order to enable fine-grained and flexible access control in the IoT, the *Authentication and Authorization for Constrained Environments* (ACE) framework has been proposed [3], building on the authorization framework OAuth 2.0 [4].

The ACE framework relies on an *Authorization Server* (AS), that has a trust relation with the RS and authorizes resource accesses from requesting Clients, based on pre-defined policies. However, the ACE framework admits the definition of separate *profiles* describing how these actors interact with each other and what communication and security protocols they use. A few profiles have been proposed, including [5] for the DTLS protocol [6], as well as [7] for OSCORE [8]. The choice of the particular profile to use has to take into account the specific use case and its security requirements, as well as the related trust and security models. This naturally leads to the most suitable communication and security protocols to adopt, and hence to the related profile describing how to use them in the ACE framework. Figure 1 shows how an IoT application for home automation can leverage on the ACE Framework profiles, e.g. a traditional Internet host like a Smart phone can secure its communications with a smart lock using IPsec, DTLS or OSCORE.

This paper presents the novel ACE IPsec profile, which describes how Client and RS set up and use an IPsec channel [9], contextually with the access control enforced by the AS. The profile displays two key benefits tightly paired with the access control provided by the ACE framework.

First, it enables secure communication between Client and RS at the network layer, by flexibly leveraging the IPsec security protocols AH [10] and ESP [11], and thus counteracting network-layer attacks such as IP spoofing. This is fundamentally achieved by establishing IPsec *Security Associations* between Client and RS. Second, it efficiently addresses the provisioning of key material, by embedding the process in the authorization workflow of the ACE framework and taking advantage of the AS. Specifically, the IPsec Security Associations can be generated by the AS, and then directly provided to Client and RS. As an alternative, the AS provides the Client and RS with the necessary key material to establish the IPsec Security Associations through the standard IKEv2 protocol [12], based either on symmetric or asymmetric cryptography.

In order to encourage wider acceptance and interoperability across multiple vendors, we submitted a draft description of our profile to the IETF for possible standardization [13]. The draft focuses on the theoretical contribution and practical considerations, and it does not refer to a particular implementation or experimental evaluation of the IPsec profile.

In this paper, we additionally describe our implementation of the ACE framework and the ACE IPsec profile for the Contiki OS [14]. We test it on real IoT devices using the resource-constrained Zolertia Firefly platform [15]. Our implementation covers all the actors in the ACE framework and is available as open source software at [16]. To the best of our knowledge, this is the first implementation of the ACE framework for the Contiki OS, and the first one ever of its IPsec profile. Additionally, it targets scenarios where even the AS is a resource-constrained device.

We utilize our implementation to experimentally evaluate the performance of the ACE framework when using the novel IPsec profile under different channel establishment and authentication methods. In particular, we consider message size, memory and energy consumption, and time required for the Client to perform an authorized resource access at the RS. Our results confirm that the IPsec profile is affordable on resource-constrained devices, and hence is effectively deployable in IoT scenarios to enforce access control paired with IPsec-based secure communication.

The rest of the paper is organized as follows. Section II discusses the related work. Section III introduces background concepts and technologies. In Section IV, the ACE IPsec profile is introduced. Section V presents our performance evaluation. Finally, Section VI draws conclusive remarks.

II. Related Work

Different profiles have been proposed for the Authentication and Authorization for Constrained Environments (ACE) framework. In [5], Gerdes *et al.* describe the Datagram Transport Layer Security (DTLS) profile, which delegates the authorization and authentication of a Client device to the establishment of a DTLS session [6] between the Client and a Resource Server (RS). Specifically, DTLS can be used in the symmetric Pre-Shared Key (PSK) mode or the asymmetric Raw Public Key (RPK) mode. If the PSK mode is used, the successful establishment of a DTLS session also acts as a proof-of-possession (PoP) for the Client's PSK. In case the RPK mode is used, the Client is authenticated through its asymmetric public key. Finally, this profile uses the Constrained Application Protocol (CoAP) [17] over DTLS between Client and RS. The feasibility of securing CoAP messages with DTLS has been investigated in [1].

The OSCORE profile of ACE proposed by Seitz *et al.* [7] provides communication security between Client and RS by means of the Object Security for Constrained RESTful Environments (OSCORE) protocol [8]. OSCORE ensures request/response binding and selectively protects CoAP messages at the application layer, by using the compact *CBOR Object Signing and Encryption* (COSE) [18] based on the *Concise Binary Object Representation* (CBOR) [19] as data encoding format. This provides true *end-to-end* secure communication between Client and RS, even in the presence of (untrusted) intermediary CoAP proxies, which remain able to perform their intended operations (e.g. message caching). This is not possible when DTLS is used, as it requires transport-layer security to be terminated at the proxy, which is thus able to inspect and possibly alter the entire content of CoAP messages exchanged between Client and RS. A secure

context can be established directly from a symmetric PoP key, or by using external key establishment protocols. Currently, the DTLS and OSCORE profiles have not been implemented or evaluated for resource-constrained IoT devices.

Compared to the OSCORE profile, the IPSec profile presented in this paper preserves and leverages a flexible key establishment based on the IKEv2 protocol [12], tightly paired with ACE authorization process. In addition, it makes it possible to employ policy-based traffic filtering, also during the actual establishment of IPsec channels between Client and RS. In contrast, this feature is not available for the DTLS and OSCORE profiles. Besides, we have implemented the IPsec profile together with the ACE framework on the Contiki OS, and tested it over resource-constrained IoT devices.

Finally, Sciancalepore *et al.* propose a different authorization framework for the IoT [20], also based on OAuth 2.0 and other standard protocols. In particular, it provides access control through an intermediary gateway acting as mediator between IoT networks and non-constrained Internet segments. However, unlike the ACE framework, [20] displays a considerably higher level of complexity and requires the intermediary gateway to be fully trusted.

III. Background

In this section, we review the main concepts and building blocks considered by the IPsec profile presented in the paper.

A. OAuth 2.0

A typical security requirement in the Internet is *authorization*, i.e. the process for granting approval to a *client* that wants to access a resource [21]. The Open Authentication 2.0 (OAuth 2.0) authorization framework has asserted itself among the most adopted standards to enforce authorization [4]. OAuth 2.0 relies on an *Authorization Server* (AS) entity, and addresses all common issues of alternative approaches based on credential sharing, by introducing a proper authorization layer and separating the role of the actual *resource owner* from the role of the *client* accessing a resource.

Specifically, OAuth 2.0 allows a client entity (e.g. a user, a host) to obtain a specific and limited access to a remote resource, hosted at a *Resource Server* (RS), while enforcing the permission from the original resource owner. That is, the resource owner grants authorization through the intermediary AS, which in turn provides the client with an *access token* including the actual authorization information. Access tokens consist of strings that are opaque to the client and encode decisions for authorized resource access in terms of duration and scope. Such decisions are ultimately taken by the AS and enforced by the RS upon processing the access token.

In addition, the AS prevents non-authorized parties from tampering with issued access token or possibly generating bogus ones. To this end, the client presents the access token to the RS upon accessing the intended resource. Then, the RS verifies that the access token is valid, before proceeding with processing and serving the request from the client. This requires that: i) the client is pre-registered at the AS; ii) the AS securely communicates with both the client and the RS; and iii) the AS and RS have pre-established a trust relation. An AS may be associated with multiple RSs at the same time. The involved parties perform RESTful interactions via the HTTP protocol [22], contacting the RESTful *endpoints* associated to specific steps in the OAuth 2.0 flow.

The approach adopted by OAuth 2.0 has become more and more important in IoT scenarios, where heterogeneous and resource constrained devices are deployed on a large scale, often configured as RS. However, these peculiarities make OAuth 2.0 *as is* not suitable for the IoT. This motivated the design of the *Authentication and Authorization for Constrained Environments* (ACE) framework [3], as a standard proposal under the Internet Engineering Task Force (IETF). The ACE framework builds on OAuth 2.0 in order to adapt and extend it for enforcing authorization in constrained IoT environments. To this end, it uses the basic OAuth 2.0 mechanisms where possible, while also providing application developers with extensions, profiles and additional guidance to ensure a privacy-oriented and secure usage.

a) Actors: The ACE framework considers the following four actors, in accordance with the main paradigm inherited from OAuth 2.0.

Client: the entity accessing a remote protected resource.
Resource Server (RS): the entity hosting protected resources and serving requests from authorized clients. Authorization is enforced through access tokens that requesting clients provide to the endpoint */authz-info* at the RS via a POST request.
Authorization Server (AS): the entity authorizing Clients to access protected resources at the RS. The AS is typically equipped with plenty of resources and hosts two endpoints: i) the */token* endpoint, for receiving Access Token Requests from Clients; and ii) the */introspect* endpoint, that the RS can use to query for extra information on received access tokens.
Resource Owner (RO): the entity owning a protected resource hosted at the RS, and entitled to grant access to it. The RO can dynamically provide its consent for giving a Client access to a protected resource, according to the traditional OAuth flows. However, the ACE framework is especially tailored to resource-constrained settings, where such consent is typically pre-configured as authorization policies at the AS. Such policies are then evaluated by the AS upon receiving a token request from a Client. In particular, the policies from the RO influence what claims the AS ultimately includes into the access token released to a requesting client.

b) Building Blocks: From an operational point of view, the ACE framework consists of the following building blocks.

OAuth [4] defines the overall authentication paradigm resulting in the protocol flows and actors' interaction.

CoAP [17] is a RESTful application-layer protocol for the IoT, typically running over UDP and able to greatly limit overhead and message exchanges. As CoAP is lightweight and tailored to resource-constrained IoT devices, it is the preferred choice in the ACE framework. Also, CoAP has been designed to explicitly support operations of intermediary Proxy nodes.

Concise Binary Object Representation (CBOR) [19] is a compact version of JavaScript Object Notation (JSON) [23], i.e. a light-weight format for data interchange which is easy to create and process. In particular, CBOR enables binary encoding of small messages conveying self-contained access tokens, CoAP POST parameters, and CoAP responses.

CBOR Object Signing and Encryption (COSE) [18] enables application-layer security in the ACE framework, especially in order to secure access tokens.

c) Authorization credentials: In order to access protected resources hosted at a RS, a Client must get the right authorization credentials in the form of an access token. Specifically, an access token is a data structure including authorization permissions issued by the AS, provided to the Client, and delivered to the RS for authorized resource access. Access tokens are opaque to the Client, i.e. their semantics are unknown to the Client, and are cryptographically protected, e.g. by means of COSE [18]. That is, access tokens are intelligible only to the RS and the AS.

A proof-of-possession (PoP) token is an access token bound to a cryptographic key, which is used by the RS to authenticate a Client request. PoP tokens rely on the AS to act as Trusted Third Party (TTP), in order to bind a PoP key (PoPK) to an access token. PoP keys can be based on symmetric or asymmetric cryptography. In case of a symmetric PoP key, the AS generates it and provides it to the Client and the RS. To this end, the AS can: i) make it available at the */introspect* endpoint; or ii) provide it to the RS (Client) in the access token (Access Token Response). For asymmetric PoP keys, the Client generates a key pair, and provides the public key to the AS in the Access Token Request. Also, the AS provides the RS' public key to the Client in the Access Token Response. The Client's public key is made available to the RS through the */introspect* endpoint, or conveyed in the access token.

The ACE framework delegates to separate *security profiles* the description of how enforcing secure communication and mutual authentication among the involved parties, as well as the details about their specific interactions. In particular, a security profile must specify: i) the communication and security protocols between the RS and the Client, as well as the methods to achieve mutual authentication; ii) the communication and security protocols for interactions between the Client and the AS; iii) the PoP protocols to use and how to select one; and iv) the mechanisms to protect the */authz-info* endpoint at the RS. The AS informs the Client of the specific profile to use by means of the *profile* parameter in the Access Token Response. Also, the AS is expected to know what profiles are supported by the Client and RS.

B. The ACE framework

The protocol flow in the ACE framework consists of the following steps, also shown in Figure 2. Communications between Client and AS as well as between RS and AS should be secured, in accordance with the used security profile.
(A) Access Token Request. The Client sends an *Access Token Request* to the */token* endpoint at the AS. This includes

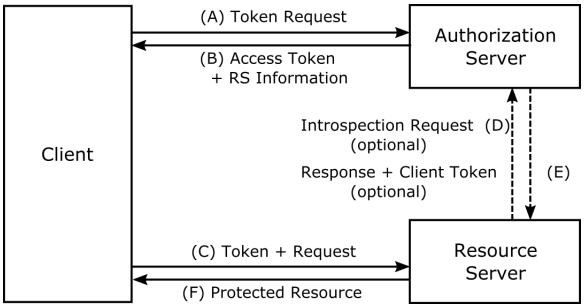


Fig. 2: ACE framework's protocol flow (adapted from [3]).

information about the Client's credentials and the requested permissions for accessing the protected resource at the RS.
(B) Access Token Response + RS Information. Once it has successfully processed the Access Token Request received at step (A), the AS generates an *access token* and provides it to the Client in the Access Token Response. The access token and the Access Token Response include also RS information and relevant parameters such as token type, expiration time, scope, state, security profile to be used and PoP key.
(C) Token + Request. First, the Client sends the access token to the */authz-info* endpoint at the RS, followed by the actual Resource Request at the specific resource endpoint. The RS and the Client authenticate each other and set up a secure communication channel, according to the security profile specified by the AS and the related PoP keys. The RS can validate the access token entirely by itself, or by interacting with the AS via the */introspect* endpoint.
(D) Introspection Request. The RS may send the access token for validation to the */introspect* endpoint at the AS. If the access token is self-contained and the RS can validate it by itself, this step as well as step (E) can be omitted.
(E) Token Introspection Response + Client Token. Upon receiving an Introspection Request from the RS, the AS validates the access token and replies to the RS. The response includes extra information to achieve mutual authentication between the Client and the RS. Additional parameters meant to be forwarded to the Client are sent as a Client Token.
(F) Protected Resource Response. Once the access token has been successfully validated and a secure channel has been established, the RS processes the Resource Request received from the Client at step (C). Then, the RS provides the requested resource to the Client over the established secure channel, in accordance with the used security profile.

C. IPsec and IKEv2

The IPsec suite is a collection of protocols to secure IP-based communications at the network layer [9]. It fundamentally relies on *Security Associations* (SAs), each of which describes how to secure a one-way channel between two parties. Thus, two SAs are required to secure a two-way communication channel. An IPsec SA is identified by a Security Parameters Index (SPI), and it specifies cryptographic material, as well as the parameters and protocols to secure IP packets through the IPsec channel. This includes the security protocol to be used, i.e. Authentication Header Protocol (AH)

[10] or Encapsulating Security Protocol (ESP) [11].

In particular, AH enables connectionless integrity and data origin authentication. Instead, ESP provides confidentiality, data origin authentication, connectionless integrity, replay protection and limited traffic flow confidentiality. Although both protocols provide integrity protection, AH additionally protects the header of IP packets. Both AH and ESP can operate in two modes, namely *transport* and *tunnel*. The former processes IP packets without changing the IP headers, while the latter encapsulates the original IP packet into a new one, thus protecting its payload and header.

Finally, SAs are established manually or dynamically, e.g. by using Internet Key Exchange Protocol version 2 (IKEv2) [12] as key exchange protocol. In particular, IKEv2 enables mutual authentication between two parties through a Diffie-Hellman (DH) key exchange, using the pre-shared key (PSK) or the certificate raw public key (Cert) mode. The usual execution of IKEv2 consists of two pairs of request/response messages, i.e. IKE_SA_INIT and IKE_AUTH. This establishes: i) an IKEv2 SA to protect IKEv2 traffic; and ii) a first IPsec SA to protect the actual IP traffic. Further SAs can be derived through CREATE_CHILD_SA messages.

IV. Protocol Overview

In this section, we describe the ACE IPsec profile. The profile provides an operative instance of the ACE framework, by defining the communication and security protocols used by a Client to perform an authenticated and authorized access to a protected resource hosted at a Resource Server (RS). In particular, it considers the IPsec protocol suite and the IKEv2 key management protocol to enforce secure communications between Client and RS, server authentication and proof-of-possession bound to an ACE access token.

Hereafter, we denote with SA-C the SA used for the unidirectional IPsec channel from the Client to the RS, while with SA-RS the SA used for the unidirectional IPsec channel from the RS to the Client. Also, information to build SAs is encoded as the newly introduced *ipsec* structure in the ACE access token. Such information includes: i) two SPIs, namely *SPI_SA_C* and *SPI_SA_RS*; ii) the IPsec mode, i.e. transport or tunnel; iii) the security protocol, i.e. AH or ESP; iv) cryptographic keys; v) the key establishment method to fully setup the two-way IPsec channel; and vi) the SAs' lifetime. In particular, *SPI_SA_C* (*SPI_SA_RS*) refers to SA-C (SA-RS). In case tunnel mode is chosen, source and destination IP addresses are also specified.

A. Key Establishment Methods

The IPsec profile provides three methods for establishing a pair of SAs, and hence a two-way IPsec channel between Client and RS. The three methods are: i) *Direct Provisioning* (DP); ii) establishment with IKEv2 and symmetric-key authentication; and iii) establishment with IKEv2 and asymmetric-key authentication. For every method, the *ipsec* structure always specifies the protocol mode, the security protocol and the SAs' lifetime. Instead, the SPIs, algorithm and cryptographic keys are specified in different ways, depending on the specific key establishment method. That is, if the Direct Provisioning (DP) method is used, this set of information are explicitly provided. Otherwise, that is IKEv2 is used as Key Management Protocol (KMP), this set of information is not explicitly provided, but rather negotiated and established when the Client and RS performs IKEv2.

The choice of the particular method to use should be driven by the capabilities of the Client and RS, as well as by the policies and infrastructure used in the specific use case for provisioning and managing key material. In particular, the DP method is extremely efficient and hence preferable for very constrained devices, as the Client and RS do not take the explicit burden to establish an SA pair. However, it does not provide strict assurances in terms of perfect forward secrecy. On the other hand, the two methods based on IKEv2 do provide perfect forward secrecy, as a native feature of the IKEv2 protocol. However, this requires the Client and RS to perform a full establishment of their SA pair through IKEv2, with a consequent considerable commitment in terms of resources. The particular choice among IKEv2 symmetric-key and asymmetric-key authentication method really depends on the key infrastructure of the specific use case. While Certificate-based public keys are typically more cumbersome to handle and process, they are often preferable to pre-shared keys that do result in more efficient processing while at the same in more complicated provisioning and management operations. In the following, we provide more details about the three key establishment methods.

1) Direct Provisioning (DP). In this method, the SA pair is pre-defined by the AS. That is, SA-RS and SA-C are specified in the access token and in the RS Information of the Access Token Response that the AS sends to the Client. Note that the AS cannot guarantee the uniqueness of the *SPI_SA_C* identifier at the RS, and of the *SPI_SA_RS* identifier at the Client. In order to address possible collisions with a previously defined SPI, the AS generates *SPI_SA_C* and *SPI_SA_RS* as random values. By doing so, the probability of a collision to occur is at most 2^{-32} for 32-bit long SPIs.

In case a collision occurs at the RS, i.e. the RS receives an access token with a *SPI_SA_C* value already used by another SA, the RS replies to the Client with an error message and aborts the setup of the IPsec channel. In network scenario scenarios where such additional overhead is not affordable, it is possible to reserve in advance a pool of SPI values intended to be used only with the DP method. This pool is exclusively managed by the AS. Then, when an IPsec channel is closed and the related pair of SAs become stale, the RS asks the AS to restore the SPI of that SA-C as available.

Instead, in case a collision occurs at the Client, i.e. the Client receives a *SPI_SA_RS* value already used by other SA, the Client sends a second Access Token Request to the AS, asking for an updated access token. This token request also includes an *ipsec* structure containing only the field *SPI_SA_RS* specifying an available identifier to use. Then, the AS replies with the corresponding Access Token and RS Information updated only as to the requested *SPI_SA_RS*.

2) IKEv2 with symmetric-key authentication. This method uses the IKEv2 protocol to establish the SA pair between Client and RS, while providing mutual authentication through

symmetric cryptography. The Client and RS run IKEv2 in symmetric mode, using a symmetric PSK provided by AS and bound to the access token as a PoP key. The PSK is made available to the Client in the Access Token Response, and to the RS in the access token. If the Client is interacting with the RS for the first time, the AS includes also a unique key identifier of the PSK in the Access Token Response. Otherwise, the Client includes in the Access Token Request a key identifier pointing at a previously established PSK.

3) IKEv2 with asymmetric-key authentication. This method uses the IKEv2 protocol to establish the SA pair between Client and RS, while providing mutual authentication through asymmetric cryptography. The Client and RS run IKEv2 in asymmetric mode, using their RPK or Certificate-based Public Key (CPK) bound to the access token as PoP keys. The RS's RPK/CPK is made available to the Client in the Access Token Response, while the Client's RPK/CPK is made available to the RS in access token. Similarly to the previous method, if the Client is interacting with the AS for the first time, it includes its RPK or CPK in the Access Token Request. Otherwise, the Client includes a key identifier linked to its own RPK or CPK, which is already available at the AS.

B. Protocol Description

In this section, we describe the message exchanges occurring in the ACE framework, in the presence of our Internet Protocol Security (IPsec) profile. Intuitively, the workflow consists of three phases, as shown in Figure 3.

Phase (I) - Unauthorized Client to Resource Server. During this phase, the Client can retrieve information necessary to contact the AS, unless already available. In particular, the Client sends an *unauthorized request* to the RS, which formally denies the request and replies by indicating the associated AS to contact for obtaining an access token.

Phase (II) - Client to Authorization Server. During this phase, the Client sends an Access Token Request to the */token* endpoint at the AS, indicating the resource of interest at the RS and the access *scope*, i.e. the intended operations on such resource. Then, the AS processes the Access Token Request and verifies that the Client is allowed to access the specified protected resource at the RS. In such a case, the AS replies with an access token and the RS information as part of the Access Token Response. In particular, the access token (RS information) includes parameters and key material intended for the RS (the Client) to set up an IPsec as a pair of SAs.

The exact information to exchange between the Client and the AS depends on the SA establishment method IV-A. Unlike the DP method, the alternative ones require the Client and the RS to establish the SA pair by running IKEv2. To this end, the AS indicates the specific KMP to use in the *kmp* field of the access token and of the RS Information. Specifically, *kmp* is set to *"ikev2"* to signal the use of the IKEv2 protocol. Provided that the involved parties have the necessary support, it is possible to use and specify a different key management protocol. Note that the AS is aware of the Client's and RS's capabilities as well as of RS's preferred and supported communication settings [3]. Therefore, the AS is

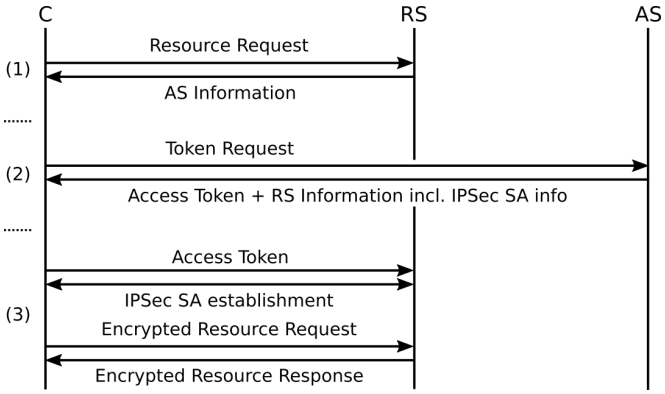


Fig. 3: IPsec profile message exchange (adapted from [13]).

able to set the security and network Parameters for the SA pair consistently with that Client-RS pair.

Phase (III) - Client to Resource Server In this phase, the Client posts the access token to the */authz-info* endpoint at the AS, through a POST CoAP message. Then, the Client and the RS set up the SA pair and the IPsec channel, based on the establishment method signalled by the AS. In particular:

a) The DP method is signalled by the presence of the *ipsec* structure, while the *"COSE_Key"* field is not present.[1].

b) A symmetric-key authenticated establishment is signalled by including a *"COSE_Key"* object with the key type parameter *"kty"* set to *"Symmetric"*, and by indicating the usage of IKEv2 with the *kmp* field set to *"ikev2"*.

c) An asymmetric-key authenticated establishment is indicated by including a *"COSE_Key"* object with the key type parameter *"kty"* indicating the usage of asymmetric cryptography, e.g. *"EC"*, by and indicating the usage of IKEv2 with the *kmp* field set to *"ikev2"*.

In case the DP method is used, the Client and the RS already have all the information to start the IPsec channel, and do not need to explicitly interact with each other. Instead, if any of the authenticated establishment methods is used, the Client and the RS perform an actual SA pair establishment through IKEv2 according to the authentication mode indicated by the *"kty"* field. In the following, we describe how the client and Client and RS finalize/setup the IPsec channel, given the specific establishment method.

a) *Direct Provisioning*. The Client derives all the necessary key material from the *"seed"* field of the *"ipsec"* structure in the RS Information. The Client uses the *seed* to perform the a key derivation algorithm as in IKEv2 [12]. Upon correct submission and successful verification of the access token at the */authz-info* endpoint, the RS performs the same key derivation process. The RS replies to the Client over the IPsec channel, according to what specified in SA-RS. Thereafter, any further communication performed during

[1]The *"COSE_Key"* is a CBOR object protected by using COSE, and containing information about the used key material, such as key type, key identifier, and the actual cryptographic key.

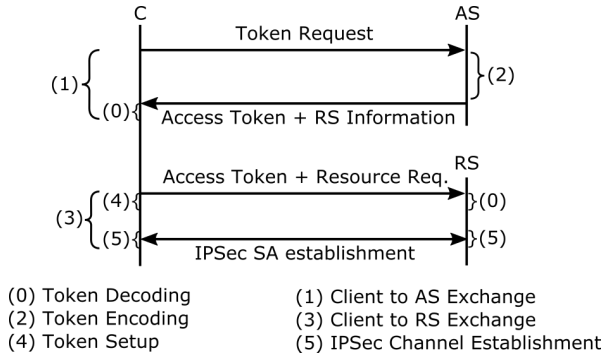


Fig. 4: Evaluation setup

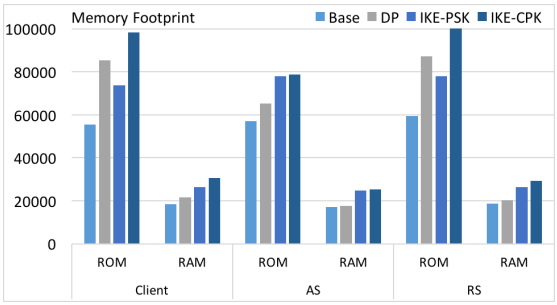


Fig. 5: Memory footprint comparison.

the lifetime of the access token occurs over the IPsec channel defined by the SA pair.

b) *Authenticated SA Establishment using IKEv2*. the Client and the RS run the IKEv2 protocol, and use the key material in the respectively received *"COSE_key"* object in order to achieve mutual authentication. In particular, the Client posts the access token to the */authz-info* endpoint to the RS, which sends back the first IKEv2 message IKE_SA_INIT to acknowledge the correct reception of the access token. Depending on the type of key used as PoP Key (PoPK), i.e. symmetric or asymmetric, the IKEv2 protocol is executed in the corresponding mode [12], i.e. PSK, CPK or RPK, with no modifications. If the IKEv2 execution is successfully completed, the Client and the RS agree on key material, parameters and algorithms used to enforce the IPsec channel.

V. Performance Evaluation and Discussion

In this section we present the performance evaluation of the different key establishment methods for the ACE IPsec Framework, described in Section IV. The implementation is written for Contiki OS [14] based on existing libraries and protocol implementations [24]. In particular, our code was evaluated on the Zolertia Firefly platform, equipped with the CC2538 radio chipset, 32 kB of RAM and 512 kB of flash ROM [15]. A device of this class supports a power supply from two AA (AAA) batteries, each of which typically provides an energy content of 9.36 (5.07) KJ.

Our implementation leverage on hardware-based cryptography and utilizes the following algorithms: AES-CCM* to secure the IEEE 802.15.4 link layer and the COSE objects; AES-128 to provide confidentiality for IPsec and IKEv2 with an 8-bytes long Initialization Vector (IV); ECC-DH with 256-bit Random ECP Group [25], SHA-2 and a Hash-based MAC (HMAC) based on SHA-256 for the authenticated exchange of IKEv2 [26], [27]. IPsec and IKEv2 traffic is encrypted using ESP in transport mode. The scenario to be evaluated is the following: a Client requests access a protected resource stored in a constrained RS. The authentication and authorization of this request are delegated to the AS. In our experimental setup the AS is as well a resource-constrained device and performs routing related activities. Namely, it is set as the root of the Direct Acyclic Graph (DAG) and performs housekeeping operations for Routing Protocol for Low-Power and Lossy Networks (RPL)[28].

TABLE I: ACE Packet size and access token size in Bytes

	Base	**DP**	**IKE-PSK**	**IKE-CPK**
(A)	10	10	10	666
(B)	179	401	445	1697
(C)	188	299	321	947
(F)	23	23	23	23
Access Token	172	283	305	931

We evaluate four setup configurations: a baseline configuration (**Base**), i.e. the ACE Framework w.o. the IPsec profile; and the three key establishment methods: **DP**, Establishment with symmetric key authentication (**IKE-PSK**) and Establishment with asymmetric key authentication (**IKE-CPK**) [12].

Our experimental results include: memory footprint, packet size and time and energy measurements. The latter measurements are evaluated as the shown in Figure 4 which depicts where time and energy measurements are performed. This measurements are labeled as follows: (0) for Access Token decoding; (1) for the Client to AS Exchange; (2) for Access Token and RS Information encoding; (3) for the Client to RS Exchange; (4) for the Access Token setup; and (5) for the IPsec channel establishment.

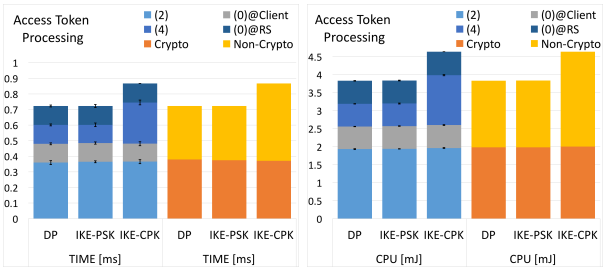
In Table I we provide the size of packets exchanged by our profile. The packet exchanges are labeled as in Figure 2. This measurements reflect the size of the CoAP messages. The last row of Table I provides the size of the *Access token*, which has a big influence on the message size.

In Figure 5 we provide the Memory Footprint evaluation results for the different SA establishment methods of the IPsec profile. We show the absolute value of ROM and RAM footprints for the setups configuration **Base**, **DP**, **IKE-PSK** and **IKE-CPK**.

Time measurements are collected using the system clock measured in system ticks. To convert our measurements to seconds the following formula is applied:

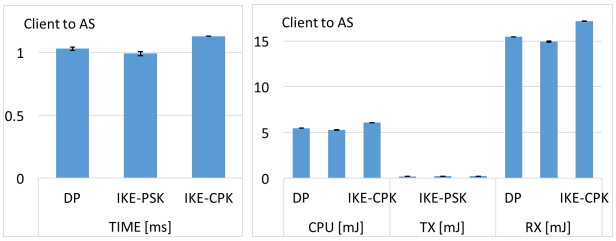
$$time = \frac{sys\,clock}{ticks/second}$$

Note that measurements (1) and (3) include network latency as round-trip time. Energy consumption measurements are divided into three contributions: energy spent at the CPU,



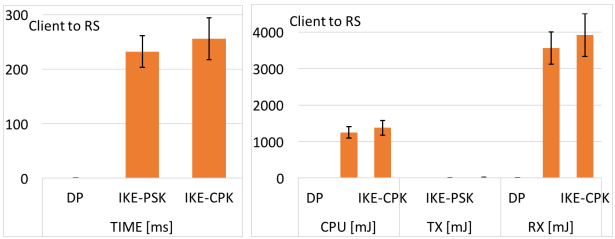
(a) Time measurements (b) Energy measurements

Fig. 6: Token processing



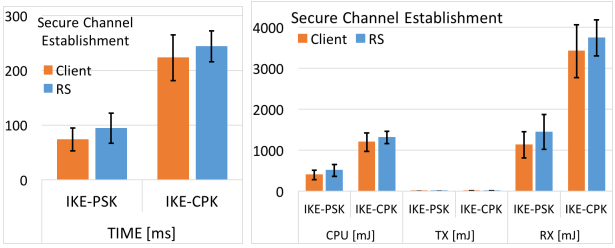
(a) Time measurements (b) Energy measurements

Fig. 7: (1) Client to AS evaluation results



(a) Time measurements (b) Energy measurements

Fig. 8: (3) Client to RS evaluation results



(a) Time measurements (b) Energy measurements

Fig. 9: (5) Secure Channel Establishment

energy spent in transmission state (TX) and the energy spent in reception state (RX). To measure the energy consumed by the devices we use powertrace, a run-time power profiling mechanism which is part of Contiki. This tool has an accuracy of 94% with an 0.6% overhead [29]. The energy consumption out of the powertrace measurements is computed as follows:

$$energy = \frac{powertrace\,value * current * voltage}{ticks/second}$$

For every setup configuration, 20 runs of the protocol were considered. We give average results for successful handshakes without packet loss. Note that wireless communication can be lossy in constrained environments with a loss rate typically increasing for larger packet sizes. In this case, the handshake duration as well as the energy consumption increase due to the retransmission of the packets.

In Figure 6 we show the time and energy results of the access token processing. On the right side of the figure we depict the contribution of the four steps involving token processing, i.e. (0) Token decoding performed at the Client and the RS, (2) Token encoding and (4) Token Setup, as in Figure 4. On the left side, we categorized these operations in crypto- and non-crypto-related actions.

The Client-to-AS message exchange results are shown in Figure 7. In this figure we can observe Client-to-AS network latency, processing time and energy consumption, i.e. measurement **(1)** in Figure 4. A comparable performance disregarding the SA establishment method can be observed. Namely, the Access Token Request/Response present a consistent behavior across the different setup configurations.

In Figure 8 we show the Client-to-RS message exchange evaluation results, i.e. the measurement tag as (3) in Figure 4. We can observe that the results for the IKEv2-based methods are comparable with the **IKE-CPK**, showing a slightly bigger energy consumption. At the same time, **DP** time and energy results are notably lower than the IKE-based key establishment methods. The total energy spent in a **DP** establishment for (3) is on average 15 mJ, and the exchange is done in less than 1 ms on average.

The evaluation results of the the establishment of a secure channel between RS and the Client are shown in Figure 9. Note that only IKE-based establishments perform an IPsec SA establishment, since in the **DP** method the IPsec SAs are provided by the AS. The Client and the RS perform similarly during (5), as in Figure 4. This result is aligned with the symmetric nature of the IPsec protocol, since both ends of the communication play a similar role, unlike protocols like DTLS where there are a client and a server role with different responsibilities. However, within this symmetry, it is noticeable that for the RS, (5) takes longer than for the Client. The aforementioned difference reflects the fact that the RS is the initiator party of the IPsec channel establishment.

We can see that the energy spent in transmission state, label as TX in Figures 7, 8 and 9 appears negligible when compared with the energy spent at the CPU or in receiving state, labeled as RX in the aforementioned figures. On the other hand, RX measurements in the aforementioned figures represent a significant share of the energy spend during a protocol run. This behavior is due to the fact that the

reception state is always set to *on* in our resource-constrained devices. Energy optimization techniques such as Radio Duty-cycle (RDC), specified and benchmarked in [30], are out of the scope of this work

VI. Conclusion

This paper has presented our novel ACE IPsec profile for authentication and authorization in the IoT. Our profile enables the scalable and flexible establishment of IPsec communication channels between Clients and Resource Servers, while contextually enforcing fine-grained access control from the ACE framework. In particular, IPsec Security Associations can be either directly provided to Client and Resource Server, or established though the standard IKEv2 key management protocol. We have implemented the IPsec profile for the Contiki OS, and carried out an experimental performance evaluation, considering resource-constrained IoT devices of the Zolertia Firefly platform. Results show that, under different configurations and authentication modes, our ACE IPsec profile is affordable also in resource-constrained devices. Therefore, it is effectively deployable in IoT scenarios for successfully enforcing access control paired with IPsec-based secure communication. Future works will focus on implementing alternative profiles of ACE and comparing their performance in resource-constrained IoT settings.

Acknowledgment

This work has been supported by the EIT-Digital High Impact Initiative ACTIVE; the EIT-Digital Master School; VINNOVA CEBOT; VINNOVA Eurostars SecureIoT; the DFG in the Collaborative Research Center CROSSING (project S1) and project C.1 within the RTG 2050 "Privacy and Trust for Mobile Users"; the EIT-Digital Master School.

References

[1] S. Raza, H. Shafagh, K. Hewage, R. Hummen, and T. Voigt, "Lithe: Lightweight Secure CoAP for the Internet of Things," *Sensors Journal, IEEE*, vol. 13, no. 10, pp. 3711–3720, 2013.

[2] S. Raza, T. Chung, S. Duquennoy, D. Yazar, T. Voigt, and U. Roedig, "Securing internet of things with lightweight ipsec," 2011.

[3] L. Seitz, G. Selander, E. Wahlstroem, S. Erdtman, and H. Tschofenig, "Authentication and Authorization for Constrained Environments (ACE)," Internet Engineering Task Force, Internet-Draft draft-ietf-ace-oauth-authz-10, Feb. 2018, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-ietf-ace-oauth-authz-10

[4] D. Hardt, "The OAuth 2.0 Authorization Framework," RFC 6749, Oct. 2012. [Online]. Available: https://rfc-editor.org/rfc/rfc6749.txt

[5] 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)," Internet Engineering Task Force, Internet-Draft draft-ietf-ace-dtls-authorize-03, Mar. 2018, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-ietf-ace-dtls-authorize-03

[6] E. Rescorla and N. Modadugu, "Datagram Transport Layer Security Version 1.2," RFC 6347, Jan. 2012. [Online]. Available: https://rfc-editor.org/rfc/rfc6347.txt

[7] L. Seitz, F. Palombini, and M. Gunnarsson, "OSCORE profile of the Authentication and Authorization for Constrained Environments Framework," Internet Engineering Task Force, Internet-Draft draft-ietf-ace-oscore-profile-01, Mar. 2018, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-ietf-ace-oscore-profile-01

[8] G. Selander, J. Mattsson, F. Palombini, and L. Seitz, "Object Security for Constrained RESTful Environments (OSCORE)," Internet Engineering Task Force, Internet-Draft draft-ietf-core-object-security-09, Mar. 2018, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-ietf-core-object-security-09

[9] K. Seo and S. Kent, "Security Architecture for the Internet Protocol," RFC 4301, Dec. 2005. [Online]. Available: https://rfc-editor.org/rfc/rfc4301.txt

[10] S. Kent, "IP Authentication Header," RFC 4302, Dec. 2005. [Online]. Available: https://rfc-editor.org/rfc/rfc4302.txt

[11] ——, "IP Encapsulating Security Payload (ESP)," RFC 4303, Dec. 2005. [Online]. Available: https://rfc-editor.org/rfc/rfc4303.txt

[12] C. Kaufman, P. E. Hoffman, Y. Nir, P. Eronen, and T. Kivinen, "Internet Key Exchange Protocol Version 2 (IKEv2)," RFC 7296, Oct. 2014. [Online]. Available: https://rfc-editor.org/rfc/rfc7296.txt

[13] S. Aragon, M. Tiloca, and S. Raza, "IPsec profile of ACE," Internet Engineering Task Force, Internet-Draft draft-aragon-ace-ipsec-profile-01, Oct. 2017, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-aragon-ace-ipsec-profile-01

[14] "Contiki: The Open Source OS for the Internet of Things," http://www.contiki-os.org/, accessed: 30-09-2017.

[15] "Zolertia: Firefly," http://zolertia.io/product/hardware/firefly, accessed: 30-09-2017.

[16] "Ipsec profile implementation for contiki," https://gitlab.com/ace-ipsec-profile/internet-draft, accessed: 27-12-2017.

[17] Z. Shelby, K. Hartke, and C. Bormann, "The Constrained Application Protocol (CoAP)," RFC 7252, Jun. 2014. [Online]. Available: https://rfc-editor.org/rfc/rfc7252.txt

[18] J. Schaad, "CBOR Object Signing and Encryption (COSE)," RFC 8152, Jul. 2017. [Online]. Available: https://rfc-editor.org/rfc/rfc8152.txt

[19] C. Bormann and P. E. Hoffman, "Concise Binary Object Representation (CBOR)," RFC 7049, Oct. 2013. [Online]. Available: https://rfc-editor.org/rfc/rfc7049.txt

[20] S. Sciancalepore, G. Piro, D. Caldarola, G. Boggia, and G. Bianchi, "OAuth-IoT: An access control framework for the Internet of Things based on open standards," in *2017 IEEE Symposium on Computers and Communications (ISCC)*, July 2017, pp. 676–681.

[21] R. W. Shirey, "Internet Security Glossary, Version 2," RFC 4949, Aug. 2007. [Online]. Available: https://rfc-editor.org/rfc/rfc4949.txt

[22] R. T. Fielding and J. F. Reschke, "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content," RFC 7231, Jun. 2014. [Online]. Available: https://rfc-editor.org/rfc/rfc7231.txt

[23] T. Bray, "The JavaScript Object Notation (JSON) Data Interchange Format," RFC 7159, Mar. 2014. [Online]. Available: https://rfc-editor.org/rfc/rfc7159.txt

[24] M. Kovatsch, S. Duquennoy, and A. Dunkels, "A Low-Power CoAP for Contiki," in *Proceedings of the 8th IEEE International Conference on Mobile Ad-hoc and Sensor Systems (MASS 2011)*, Valencia, Spain, Oct. 2011.

[25] M. Lepinski and S. Kent, "Additional Diffie-Hellman Groups for Use with IETF Standards," RFC 5114, Jan. 2008. [Online]. Available: https://rfc-editor.org/rfc/rfc5114.txt

[26] P. E. Hoffman, "Use of Hash Algorithms in Internet Key Exchange (IKE) and IPsec," RFC 4894, May 2007. [Online]. Available: https://rfc-editor.org/rfc/rfc4894.txt

[27] S. Frankel and S. G. Kelly, "Using HMAC-SHA-256, HMAC-SHA-384, and HMAC-SHA-512 with IPsec," RFC 4868, May 2007. [Online]. Available: https://rfc-editor.org/rfc/rfc4868.txt

[28] R. Alexander, A. Brandt, J. Vasseur, J. Hui, K. Pister, P. Thubert, P. Levis, R. Struik, R. Kelsey, and T. Winter, "RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks," RFC 6550, Mar. 2012. [Online]. Available: https://rfc-editor.org/rfc/rfc6550.txt

[29] A. Dunkels, J. Eriksson, N. Finne, and N. Tsiftes, "Powertrace: Network-level power profiling for low-power wireless networks," 04 2011.

[30] A. Dunkels, J. Eriksson, and N. Tsiftes, “Low-power interoperability for the ipv6-based internet of things,” 05 2012.