ACCEPTED FROM OPEN CALL

Overview of Device Access Control in the IoT and its Challenges

V. Beltran and A. F. Skarmeta

Access control is a building block for the overall security of any communication system. When it comes to device-to-device communication, decentralized approaches for access control will allow governing a mass of devices in a scalable mode. Common understanding and standardization of application-level access control is also primordial for the incoming era of cooperating devices in the IoT.

Access control is a building block for the overall security of any communication system. When it comes to device-to-device communication, decentralized approaches for access control will allow governing a mass of devices in a scalable mode. Common understanding and standardization of application-level access control is also primordial for the incoming era of cooperating devices in the IoT.

Abstract

Access control is a building block for the overall security of any communication system. When it comes to device-to-device communication, decentralized approaches for access control will allow governing a mass of devices in a scalable mode. Common understanding and standardization of application-level access control is also primordial for the incoming era of cooperating devices in the IoT. This article introduces different architectural models for decentralized device access control, their security requirements and implications.

Introduction

Unauthorized access to IoT devices is a major peril in the IoT. If an attacker succeeds in bypassing a device's access control, sensitive information at the device can be revealed. It is a major concern in smart cities where citizens are surrounded by devices collecting private information (e.g., voice-recognition-enabled TVs that record private conversation, GPS-enabled devices tracking our location or video cameras recording our activities). Considering that data from IoT devices may be linked to other sensitive information (e.g., health/justice records, personal history profiles and bank financial data), it is undeniable that unauthorized access to data in the IoT can be a serious threat to citizens. Not to mention how unauthorized actuation on IoT devices can be potentially harmful on a city scale, controlling critical systems (e.g., health systems, security infrastructures and urban supply networks) autonomously without explicit user consent.

Until now little attention has been given to device access control because of the closed nature of IoT services (i.e., devices typically only interact within their provider's walled garden). This article aims to enlighten readers about the possible architectures for device access control in the IoT and their security implications. This article especially focuses on decentralized models, since scalability will be primordial in large-scale deployments of IoT devices. The rest of this article is organized as follows. The following section first describes possible communication models in the IoT and the role of access control in them. Then we introduce access control enforced by IoT devices and different decentralized architectural models. Following that we analyze the security requirements and implications of these models. Then we describe four proposals in the literature for decentralized access control, and their fulfillment of the presented security taxonomy. The final section gives some conclusions.

IoT Communication Models and Access Control

To foster the discussion on interoperability for IoT devices or so called smart objects, the Internet Architecture Board released guidelines for IoT developers [1]. This recommendation includes different communication models between smart objects. In the device-to-cloud model (Fig. 1a), smart objects only communicate with their cloud service. This service acts as a sink of device data and offers interfaces to access this data for users or other cloud services. No access control needs to be enforced by the device; it is the cloud service who enforces data security.

In the device-to-application-layer gateway GW model (Fig. 1b), smart objects connect to an application-layer GW to communicate with any party. Intermediary GWs are commonly necessary for IoT devices that are not IP-enabled. Thus, the GW performs all the necessary data/protocol translations between the device and the Internet. The GW typically pushes aggregated data from the devices to its cloud service. In cases where other devices can access the device's resources, the cloud service updates the GW about the authorized clients in an access control list (ACL). Typically, the device, the cloud service and the GW are provided by the same vendor. Nevertheless, some commercial aggregating hubs offer automation between IoT devices from different providers (Fig. 1c). The capability of access control at the aggregating hub will depend on partnership agreements with device providers, which makes the system more complex and fragile.

In the device-to-device model, the smart object can directly communicate with other IoT devices without any application-level intermediary. This communication may take place within the device's wireless local area network or through the Internet. Access control therefore needs to be applied by the device itself. The cloud service keeps the device's ACL up to date, directly (Fig. 1d) or through an application-layer GW if the device is not IP-enabled (Fig. 1e).

Decentralized Access Control for IoT Devices

In the typical client-server model shown in Fig. 2a, the server device is a sensor or actuator holding the resource. The server device, thereinafter, the

Digital Object Identifier:
10.1109/MCOM.2017.1700433

The authors are with the University of Murcia.

0163-6804/18/$25.00 © 2018 IEEE

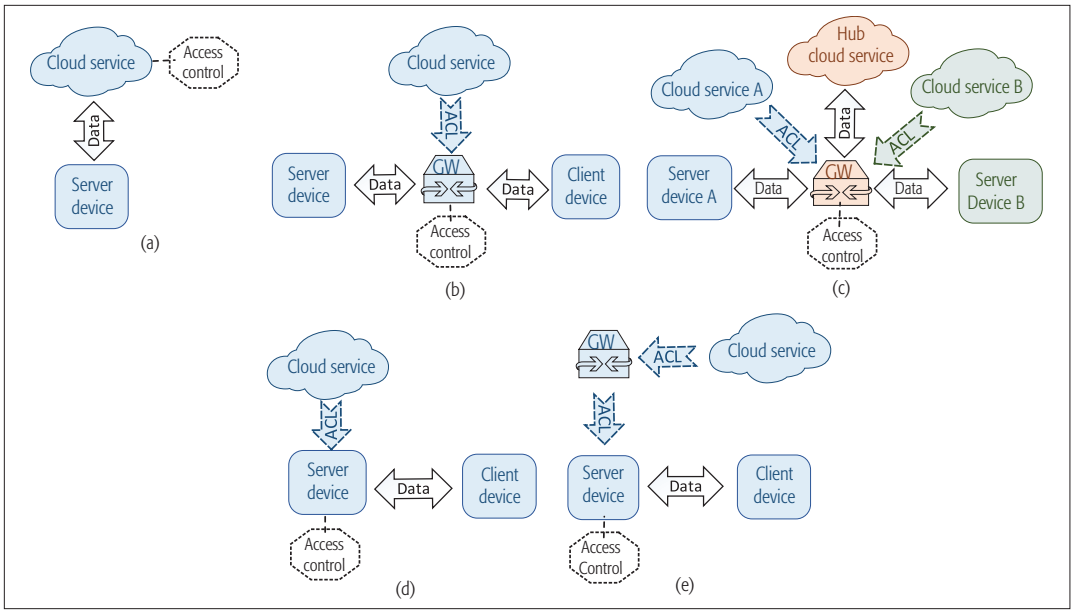

Figure 1. IoT communication models: a) device-to-cloud; b) device-to-application-layer gateway (GW); c) device-to-GW with aggregating hub; d) device-to-device with direct device-cloud communication; e) device-to-device with GW-based cloud communication.

resource server (RS), is responsible for applying the authorization policies in its ACL. Thus, when a client requests a resource (i.e., some data or command execution), the RS will verify the client's identity. Secure identity verification involves the client authenticating by proving possession of a symmetric or asymmetric cryptographic key. If this verification is successful, the RS will check out whether the client is authorized to access the requested resource.

This client-server approach requires having every IoT device managing client identities and possibly applying authorization policies. In medium or large scale deployments, this model will be unfeasible since it involves maintaining authorization information in thousands or hundreds of devices. The expected explosion of the IoT will demand high scalability to govern a mass of devices. Thus, access control in the IoT requires decentralized models that allow applying a system's authorization policies across a multitude of devices. In these models, the RS delegates authorization to an external entity, an authority in its security domain that is frequently referred to as the authorization server (AS). The RS is no longer pre-configured with the identities of legitimate clients in an ACL. Instead, a client wishing to access the RS needs to obtain authorization from the AS on-demand. When the RS receives a request, it needs to verify that this request has been previously authorized from the AS (see Fig. 2(b)).

Decentralized authentication is not new. Three-party authentication, protocols such as Kerberos [2] and OAuth [3] have been extensively deployed in communication networks and the Web, respectively. However, for the IoT to embrace decentralized authorization, new models adapted to the nature of IoT devices are necessary.

We identify three architectural models that well embody the design space for decentralized access control: key derivation, authorization pass and RS-to-AS verification. These architectures differ on how the RS verifies the client's authorization, as introduced below, and their security aspects are

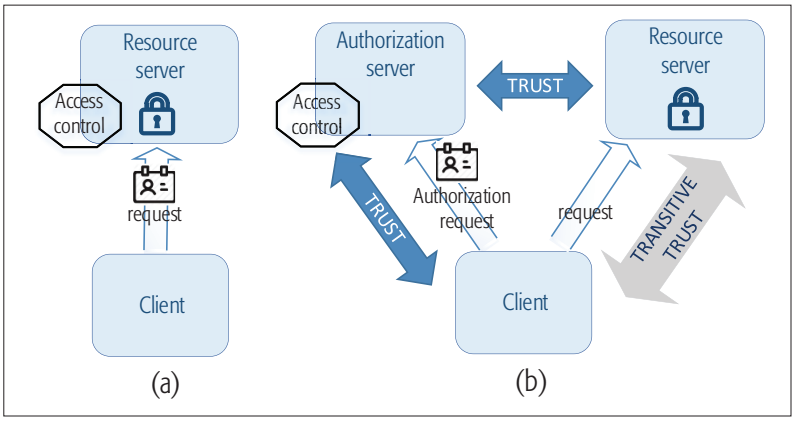

Figure 2. Centralized a) and decentralized b) access control.

discussed later. In the key derivation model in Fig. 3a, the RS shares a secret with the AS and authorized clients are provided with symmetric keys that are generated from this shared secret. When the RS receives a client's request, the former is able to build the client's symmetric key from its shared secret and hence authenticate the client.

In the authorization pass model in Fig. 3b, the client provides the RS with its credentials in the form of an authorization pass. The client obtains this pass along with its cryptographic material from the AS. When the client wishes to access the authorized resource, it first needs to forward the authorization pass to the RS.

In the RS-to-AS verification model, the RS needs to communicate with the AS in order to verify the client's authorization. When the client wishes to access the RS, the former has to transmit some data — let us call it a request token —that uniquely identifies its request. The client may not need to first obtain authorization from the AS. In this case, the client generates the request token and attaches it to its request toward the RS, as shown in Fig. 3c. This token is opaque for the RS that forwards it to the AS to authenticate the client. If authorization is suc-

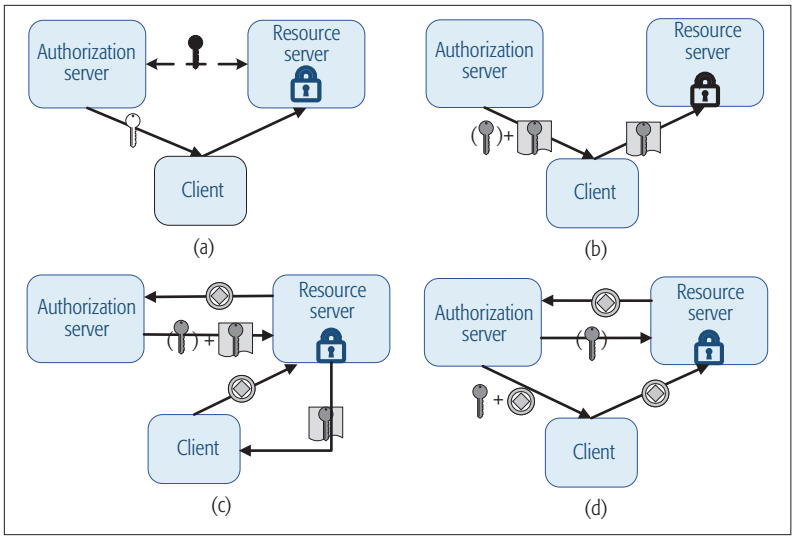

Figure 3. Architectural models for client authentication with a) key derivation, b) authorization pass, c) stateless RS-to-AS verification, and d) stateful RS-to-AS verification. Black- and white-colored keys denote shared secrets and access keys derived from shared secrets, respectively. Grey-colored keys denote keys that can be either symmetric or asymmetric. Keys in parentheses are optional.

cessful, the AS will generate another cryptographic token — from hereinafter the response token — to be consumed for the client for server authentication purposes. We refer to this model as stateless RS-to-AS verification, since the AS does not need to keep state regarding the client's authorization request. Alternatively, the client may be required to obtain authorization from the AS, as shown in Fig. 3d. In this stateful mode, the AS attaches a request token to its response toward the client. This token is cryptographically bound to the AS and opaque to both the client and the RS. Moreover, the AS provides the client with the necessary cryptographic material to authenticate the server.

Security Requirements of IoT Decentralized Access Control Models

In any decentralized model for access control, three main security requirements are necessary to prevent impersonation attacks: client-AS mutual authentication, confidentiality and integrity of the client's credentials for the RS, and client-RS mutual authentication. The following subsections describe how the decentralized access control models introduced earlier can support these requirements. Table 1 summarizes the security mechanisms for the client's communications with both the AS and the RS. This table includes the basic cryptographic operations that can guarantee the security requirements above; particular implementations may involve others. The communication security between the RS and AS is not considered, but they should always authenticate mutually. Table 1 assumes that trust relationships have been established during the bootstrapping phase. As a result, the client and the RS have been preconfigured with their credentials toward the AS.

Key Derivation Model

This model always involves the AS sending symmetric keys to clients. Table 1 assumes that the client only keeps one shared secret per RS. Nevertheless, in cases where clients have scope-specific credentials, this model would involve the RS and client keeping more than one shared secret, as outlined in the following section.

Credentials confidentiality is required in the communication channel between the client and the AS. This confidentiality can be achieved by transport-level security through the use of Datagram Transport Layer Security (DTLS), which can also provide mutual authentication. Client credentials can also be kept confidential at the application level by having the AS encrypting them. This application-level encryption also guarantees implicit mutual authentication when the credentials are encrypted with the client's shared secret with the AS. The client will be able to decrypt the credentials only if it holds the correct shared secret. The client can also confirm the legitimacy of the credentials, since they were encrypted with the correct shared secret. When the AS encrypts the client's credentials with the latter's public key, server authentication is not provided. In this case, the AS should attach a public-key signature to its response.

Mutual authentication between the client and RS is always based on mutual proof of possession (PoP) of their shared secret. The client and RS can attach a message authentication code (MAC) to its request and response, respectively. Otherwise, if data confidentiality is necessary, DTLS or data-level encryption can be used.

Note that some implementations may rely on the RS to generate cryptographic keys. In this case, the RS needs to have enough entropy for shared secrets to avoid brute force attacks [4]. When the RS keeps only one symmetric key for each client, this model does not require per-request specific processing other than MAC verification. In this case, the key derivation model is more resilient to denial of service (DoS) attacks than other models, since it puts less load on the RS.

Authorization Pass Model

When the authorization pass contains a shared secret for the client and the RS, the AS sends this shared secret to the client. Thus, these credentials should be kept confidential based on a shared secret between the AS and the client or asymmetric cryptography. As described in the key derivation model above, mutual authentication should be done in both cases. The authorization pass needs to be kept confidential all the way through its delivery to the RS. The AS is generally a device with more computing power than the client and the RS. Thus, the AS should encrypt the pass with the RS's cryptographic key in order to avoid an extra DTLS handshake for the client to transmit the pass to the RS.

In cases where the authorization pass contains a public key or an identifier of a public key, the AS should sign it to provide integrity-protection and authenticity. In this case, the client will authenticate to the AS by signing its request. The AS will attach the RS's public key for the client to authenticate server responses.

RS-to-AS Verification Model

In the stateless RS-to-AS verification model, the client generates the request token, which should be linked to the client's identifier and the request's

<table>
<tr><th colspan="2">Decentralized model</th><th colspan="2">Key derivation</th><th colspan="2">Authorization pass</th><th colspan="2">Stateful RS-to-AS verification</th><th colspan="2">Stateless RS-to-AS verification</th></tr>
<tr><td colspan="2">Client's identity</td><td>Kc-as</td><td>PuKc</td><td>Kc-as</td><td>PuKc</td><td>Kc-as</td><td>PuKc</td><td>Kc-as</td><td>PuKc</td></tr>
<tr><td colspan="2">Client's server key</td><td>Kc-rs</td><td>Kc-rs</td><td>a) PuKrs
b) Kc-rs</td><td>a) PuKrs
b) Kc-rs</td><td>a) PuKrs
b) Kc-rs</td><td>a) PuKrs
b) Kc-rs</td><td>a) PuKrs
b) Kc-rs</td><td>a) PuKrs
b) Kc-rs</td></tr>
<tr><td rowspan="3">Client-AS channel</td><td>AS credentials protection</td><td rowspan="2">DTLS(Kc-as)
|| ENC(Kc-as)</td><td>DTLS(PKPas)
|| ENC(PuKc)</td><td rowspan="2">a) MAC(Kc-as)
b) DTLS(Kc-as)
|| ENC(Kc-as)</td><td>a) SIG(PrKas)
b) DTLS(PKPas)
|| ENC(PuKc)</td><td rowspan="2">a) MAC(Kc-as)
b) DTLS(Kc-as)
|| ENC(Kc-as)</td><td>a) SIG(PrKas)
b) DTLS(PKPas)
|| ENC(PuKc)</td><td colspan="2" rowspan="3">N/A</td></tr>
<tr><td>AS authentication</td><td>DTLS(PKPas)
|| SIG(PrKas)</td><td>a) SIG(PrKas)
b) DTLS(PKPas)
|| SIG(PrKas)</td><td>a) SIG(PrKas)
b) DTLS(PKPas)
|| SIG(PrKas)</td></tr>
<tr><td>Client authentication</td><td>DTLS(Kc-as)
|| DEC(Kc-as)</td><td>DTLS(PKPc)
|| DEC(PrKc)</td><td>a) MAC(Kc-as)
b) DTLS(Kc-as)
|| DEC(Kc-as)</td><td>a) SIG(PrKc)
b) DTLS (PKPc)
|| DEC(PrKc)</td><td>a) MAC(Kc-as)
b) DTLS(Kc-as)
|| DEC(Kc-as)</td><td>a) SIG(PrKc)
b) DTLS(PKPc)
|| DEC(PrKc)</td></tr>
<tr><td rowspan="3">Client-RS channel</td><td>Credentials protection</td><td colspan="2">N/A</td><td colspan="2">a) Pass integrity
b) Pass confidentiality</td><td colspan="4">N/A</td></tr>
<tr><td>Client authentication</td><td colspan="2">C-PoP(Kc-rs)</td><td colspan="2">a) C-PoP(PrKc)
b) C-PoP(Kc-rs)</td><td>a) MAC(Kc-as)
b) C-PoP(Kc-rs)</td><td>a) SIG(PrKc)
b) C-PoP(Kc-rs)</td><td>a) MAC(Kc-as)</td><td>a) SIG(PrKc)</td></tr>
<tr><td>RS authentication</td><td colspan="2">S-PoP(Kc-rs)</td><td colspan="6">a) S-PoP(PrKrs)
b) S-PoP(Kc-rs)</td></tr>
</table>

Kx-y: symmetric key shared between devices X and Y
PuKx/ PrKx : asymmetric public/private keys of device X
SIG(PrK): public-key signature with private key PrK
MAC(Ks): Message Authentication Code with symmetric key Ks
DTLS(K/PKP): DTLS using symmetric key K or a public-private key pair PKP
DEC(K): Decryption using symmetric or asymmetric key K
ENC(K): Encryption using symmetric or asymmetric key K
C-PoP(K): Proof of Possession of the client's symmetric or asymmetric key (i.e., DTLS, DEC, MAC or SIG)
S-PoP(K): Proof of Possession of the server's symmetric or asymmetric key (i.e., DTLS, ENC, MAC or SIG)

Table 1. Security operations for decentralized access control models, when the client's identity towards the AS is a symmetric key (Kc-as) or a public key (PuKc). Cryptographic keys belonging to the client device, the AS and the RS have subscripts "c," "as" and "rs," respectively. Some models accept two alternatives for server authentication: the client device authenticates the RS with its public key - a) option-, or the AS establishes a shared secret between the client and the RS b) option-. Thus, the AS can protect the client credentials in two different modes (separated by "| |") that determine the methods for mutual authentication.

parameters. The client uses its own credentials to sign or MAC the request token with its private key or shared secret, respectively. The RS will forward this token to the AS to authenticate the client. If the AS authenticates and authorizes the client's token, the AS will provide the RS with a response token to be forwarded to the client. This token will allow the client to authenticate the server. It can contain a fresh shared secret between the RS and the client. In this case, the AS will encrypt the token with the client's credentials and provide the RS with the fresh shared secret with the client. Otherwise, the response token will contain the RS's public key, and in this case the AS will sign or MAC the token with its private key or the client's shared secret, respectively.

In the stateful RS-to-AS verification model, the client will request authorization from the AS in order to obtain an opaque request token. The AS can respond to the client with a fresh shared secret to authenticate toward the RS.

In this case, the client will attach a MAC to its request toward the RS. The AS will provide the RS with this shared secret in order to let the latter authenticate to the client.

Conversely, the AS can provide the client with the RS's public key if the client has to authenticate with its own credentials. The client will use its private key or shared secret with the AS to sign or MAC the request token, respectively. If the AS successfully authenticates the token and authorizes the client, the RS will attach a public-key signature to its response to the client.

In both the stateless and stateful modes of this model, when a shared secret is established between the client and the RS, these devices may cache this secret for subsequent authorization requests without the AS involvement. Likewise, with asymmetric cryptography, the AS can provide the RS with the client's public key to enable subsequent client request without its involvement.

Security Implications for IoT Decentralized Access Control

This section is intended to serve as a guideline to measure the suitability of each access control model described earlier for different scenarios of decentralized authorization in the IoT [5]. To this end, Table 2 summarizes the capability of the

	Scope limitation	Access revocation	Offline AS-RS	Offline AS-client
Key derivation	–	×	✓	✓
Authorization pass	✓	After pass lifetime	✓	For pass lifetime
RS-to-AS verification	✓	✓	×	✓ if stateless

Table 2. Security features of decentralized access control models that are satisfied (✓), not supported (x) or not suitable due to scalability issues (–).

models to provide the relevant security-related features in these use cases: authorization scope, access revocation and resilience to offline AS.

First, authorization scope is the capability of the AS to limit the scope of authorization. It is important to guarantee that clients only access the resources that they need and are granted authorization. An access grant may be scoped per device, per resource or per-service (i.e., read/write operations). An access grant may also be scoped to contextual information such as validity periods or local conditions that will be checked by the RS. Second, revocation prevents unauthorized access of clients that were at some point authorized but are no longer allowed to access the RS. Thus, the AS should be capable of revoking the access granted to a client. Lastly, resilience to offline AS refers to the capability of the client or RS to function even if they are unable to communicate with the AS at the time the client's request takes place. This situation may be due not only to connectivity problems or AS downtime but also to energy-saving restrictions at battery-powered constrained devices. In Table 2, the terms "offline AS-client" and "offline AS-RS" refer to the resilience to offline AS by the client and the RS, respectively. Note that if an access revocation event occurs during the time the AS is offline (to the client or/and the RS), the RS may grant access to no-longer-authorized clients in any of the decentralized access control models. A possible mitigating solution is to implement some lightweight mechanism at the RS for detecting the connectivity status with the AS. In the case that the AS was not reachable, the RS may decide not to grant access to any client. The following subsections outline the suitability of the decentralized access control models for the above-mentioned security-related features.

Key Derivation Model

In this model, the client's authorization grant is determined by a cryptographic key at the RS. Thus, it is not well suited to the dynamic limitation of authorization grants. Different authorization scopes can considerably overload the RS since it has to keep a different symmetric key per scope. For client-specific scopes, the RS will have to generate several symmetric keys for each client based on the scope and the client's information. Revocation is not supported, since the AS does not transmit any information to be consumed by the RS. A mitigating solution might be to have the RS counting the number of times a cryptographic key is used by a client. When the client reaches a threshold, it will have to renew its key. Note, however, that in this model, the only mode to revoke a client's authorization is to renew the corresponding cryptographic key of the server. This will automatically revoke all the clients that have been authorized through the same key. Regarding resilience to offline AS, if the client cannot connect to the AS, the client will still be able to access the RS if it has a valid cryptographic key that has not yet expired. When the RS is unable to communicate with the AS, since this model does not involve the AS for client authentication, the RS will still allow clients to access.

Authorization Pass Model

The authorization pass model allows limiting the scope of authorization grants by adding access rules to authorization passes. To mitigate the effect of unauthorized clients with valid keys, authorization passes should always have a lifetime. After this lifetime expires, the client would be revoked and would need to renew its authorization from the AS. Regarding resilience to offline AS, the client and the RS can communicate with each other even when they cannot communicate with the AS, as long as the client has some live authorization pass.

RS-to-AS Verification Model

Revocation and authorization scope limitation is straightforward in this model, since the RS communicates with the AS for client authorization. Since this model requires the RS to communicate with the AS for every client request, it is not resilient to offline AS. As long as the AS is offline, the RS will not grant access to any client. Nevertheless, implementations may allow the RS and client to cache their shared secret or public keys for mutual authentication without the AS's involvement. In this case, the AS should include information about the lifetime of these credentials in the response token to the client and its response to the RS. In this case, the client and RS will be able to communicate securely during the time the AS is offline, as long as their credentials are not expired.

Research Area on IoT Decentralized Access Control

The next subsections introduce four research proposals for IoT access control. The first one is being promoted toward standardization by the IEFT. The other three proposals are, to the best of the authors' knowledge, the most consolidated academic efforts that specifically target access control for IoT devices in a decentralized mode. Table 3 provides a summary of the characteristics of these proposals based on the security taxonomy previously discussed.

The IETF Authentication and Authorization for Constrained Devices

In the Authentication and Authorization for Constrained devices (ACE) [6], the AS issues authorization passes, referred to as PoP tokens in ACE terminology, along with their associated cryptographic material to authorized clients. The client first posts the token to the "/authz-info" endpoint of the RS; then the client proves possession of the token's cryptographic key for the subsequent data request. ACE specifies that the AS has to indicate how to prove possession of the pass' credentials in the "profile" parameter of its response to the client. However, the definition of security profiles

is out of the scope of ACE. Although ACE requires data confidentiality, integrity-protection and mutual authentication on any communication, it does not put any restriction on how the communication with the AS should be protected. Nevertheless, for data-level security, ACE recommends the Concise Binary Object Representation (CBOR) Object Signing and Encryption (COSE) format [7].

ACE also specifies other modes based on OAuth introspection [8]. In this case, the RS does not process the authorization pass. Instead, it posts the pass to the AS's "introspect" endpoint. The AS will reply to the RS with the pass's keying material for mutual authentication with the client. Thus, this model belongs to the stateful RS-to-AS verification model. ACE is also tentatively considering a different model for clients with limited connectivity, referred to as "client token." Clients are configured with a long-term client token. A client attaches this token to the request to the RS that forwards it to the AS. If the client is authorized, the AS will provide the RS with the keying material for securing the communication with the client. The RS will transmit this keying material to the client. Thus, the ACE client token model corresponds to the stateless RS-to-AS verification model.

Object-Based Security Architecture

The object-based security architecture (OSCAR) [9] relies on symmetric keys that are shared between the RS and its authorized clients. OSCAR is therefore based on the key derivation model, and it relies on the Constrained Application Protocol (CoAP). The AS updates both clients and the RS about the keying material over secure DTLS channels. When a client sends a request to an RS, the RS will construct an encryption key based on the resource's keying material, the client request's id along with other data such as the client's identifier. This encryption key will be used to encrypt the resource's data and send it to the client directly. The client will be able to decrypt the response's content only if it possesses the right resource's keying material and, using it, can construct the decryption key. Moreover, OSCAR allows clients to double-check the legitimacy of messages. The RS signs the response's content with its private key before encrypting it with the symmetric key.

OSCAR is designed to authorize read operations on resources and hence write operations (i.e., actuation on devices) are not supported. Although different levels of authorization scope can be supported, each scope's symmetric key is shared among all the authorized clients. Thus, to limit the authorization scope on a per-client basis is not possible. Although OSCAR does not involve the RS keeping client-specific state, it has to generate an encryption key and encrypt the client's response on a per-request basis. This may make server devices more vulnerable to DoS attacks.

Access Control for Constrained Environments over Named Data Networking

Access Control for Constrained Environments over Named Data Networking (NDN-ACE) [10] describes an access control mechanism for actuators based on the Named Data Networking (NDN) paradigm [11]. Nevertheless, this proposal is independent from the underlying communication paradigm (i.e., data-centric or Internet Protocol-based routing).

NDN-ACE, like OSCAR, belongs to the key derivation model. Converse to OSCAR, this proposal is fully based on application-level security. Any request needs to be signed by the sender. Moreover, it is the RS who maintains the symmetric keys for each of its services and lets the AS know about them. Whenever a new symmetric has to be sent to the AS, it is encrypted by the RS (with either a shared secret or the AS's pub-

	ACE	OSCAR	NDN-ACE	NDN-IE
Authorization model				
Authorization pass	✓			✓
RS-to-AS verification	✓			
Key derivation		✓	✓	
Security mechanisms				
Client-to-AS authentication	N/D	DTLS(PrKc)	SIG(PrKc)	SIG(PrKc)
Client-to-RS authentication	N/D	DEC(Kc-rs)	MAC(Kc-rs) + SIG(PrKc)	SIG(PrKc)
RS-to-client authentication	N/D	ENC(Kc-rs)+ SIG(PrKrs)	MAC(Kc-rs)	SIG(PrKrs).
Client cryptographic keys	N/D	PKPc + PuKas +N_{RS}*(PuKrs + Nsc*Kc-rs)	PKPc + PuKas + NRS*(PuKrs + Nsc*Kc-rs)	PKPc + PuKas + N_{RS}*PuKrs
Security features				
Per-client scope limit	✓	X	✓	✓
Access revocation	✓ with RS-to-AS ver. model	X	After key lifetime	After key lifetime
Offline AS-RS	✓ with auth. pass model	✓	✓	X
Offline AS-client	✓ (for key lifetime)	✓	✓	✓ (for key lifetime)
Authorization scope levels				
Device	✓	✓	✓	✓
Resource	✓	✓	✓	✓
Service	✓	READ	✓	✓
Context	✓	×	–	–

N_{RS}: average number of RS with which the client communicates
Nsc: average number of different authorization scopes per RS
Kx-y: symmetric key shared between devices X and Y
PKPx: public/private-key pair of device X
PuKx/ PrKx : asymmetric public/private keys of device X
DTLS(K): DTLS handshake done with key K
DEC(K)/ENC(K)/SIG(K)/MAC(K): Decryption, encryption, public-key signature and MAC cryptographic operations with key K

Table 3. Security analysis for four proposals for decentralized access control. Cryptographic keys belonging to the client device, the AS and the RS have subscripts "c," "as" and "rs," respectively. N/D stands for Not Defined. "—" means that the proposal has a limited support to the feature.

Although IoT GWs are expected to be widely used for non-IP devices or legacy IPv4-only services, these GWs should remain working at the network level. In this regard, the use of common application-level models for access control to IoT devices will facilitate the desired IoT interoperability at the application level.

lic key). When a client wishes to access an RS's service, it gets the proper access key from the AS. The AS generates the client's access key for the service based on the service's symmetric key as well as other information (e.g., the targeted service and the key's sequence number). The access key is encrypted with a fresh encryption key derived from a Diffie-Hellman key exchange between the client and the AS. When a client sends a request to the RS, the former attaches a keyed-hash MAC (HMAC) calculated from the client's access key. The client also includes information about its key, which lets the RS construct the same access key based on its symmetric key and hence verify the client's HMAC.

This approach enables limiting the scope of each client's authorization grant, since the RS builds client-specific access keys based on the scope's cryptographic material and some data attached to the client's request (e.g., targeted resource and sequence number). If the client would transmit a scope different from that granted by the AS, the RS would generate a key different from the one used for the HMAC, thereby rejecting the client's request.

Named Data Networking Based Instrumented Environments

The proposal in [12], which we refer to as NDN-based Instrumented Environments (NDN-IE), describes an authorization mechanism for actuators based on the NDN. Like in NDN-ACE, we only focus on the access control architecture that is agnostic to the networking paradigm.

In this proposal, when a client wishes to access an RS, it gets the authorization from the AS in the form of signed application names. Thus, this approach follows the authorization pass model. An application name is a hierarchical name mainly composed of the domain of the authorization, the name of the client, the client's permissions and the client's public key. The AS signs application names that hence serve as authorization passes. When the client sends a request to the RS, the former adds its signed application name and signature to the request. The RS then retrieves the client's public key from the signed application name and verifies the client's signature. The RS will also verify that the client is authorized to access the requested resource based on the signed application name's domain and permissions.

Mutual authentication is done by the sender and the recipient attaching their public-key signatures to their messages. Moreover, this proposal defines that the RS may be configured with a long-term symmetric key that is used to derive per-client symmetric keys. In this case, the RS sends this shared secret encrypted with the client's public key. Although new attributes can be defined to include contextual conditions for access control in signed application names, their expressiveness is more limited than that of other more flexible formats such as CBOR.

Conclusion

Scalable access control methods among IoT devices should be adopted to secure the distributed cooperation between IoT devices. Although IoT GWs are expected to be widely used for non-IP devices or legacy IPv4-only services, these GWs should remain working at the network level. In this regard, the use of common application-level models for access control to IoT devices will facilitate the desired IoT interoperability at the application level. To shed some light on it, this article has presented different architectural models for access control. Each model involves certain load in terms of cryptographic keys and operations to IoT devices and has different security implications. A security taxonomy has been presented to help developers choose the access control architecture that best fits their IoT system's needs. By way of example, four research proposals for decentralized access control have been analyzed based on this security taxonomy.

Acknowledgment

This work has been sponsored by the Spanish National Project PERSEIDES (TIN2017-86885-R) granted by the Ministry of Economy and Competitiveness of Spain (including ERDF support), CHISTERA PCIN-2016-010, and the IoTCrawler project funded from the European Union's Horizon 2020 research and innovation program.

References

[1] H. Tschofenig *et al.*, "Architectural Considerations in Smart Object Networking," IETF RFC 7452, Mar. 2015.
[2] B. C. Neuman and T. Ts'o, "Kerberos: An Authentication Service for Computer Networks," *IEEE Commun. Mag.*, vol. 32, no. 9, Sept. 1994, pp. 33–38.
[3] V. Beltran, "Characterization of Web Single Sign-On Protocols," *IEEE Commun. Mag.*, vol. 54, no. 7, July 2016, pp. 24–30.
[4] P. Eronen and H. Tschofening, "Pre-Shared Key Ciphersuites for Transport Layer Security (TLS)," IEFT RFC 4279, Dec. 2015.
[5] L. Seitz *et al.*, "Use Cases for Authentication and Authorization in Constrained Environments," IETF RFC 7744, Jan. 2016.
[6] L. Seitz *et al.*, "Authentication and Authorization for Constrained Environments," IETF Internet-Draft ietf-ace-oauth-authz-07, Aug. 2017.
[7] J. Schaad, "CBOR Object Signing and Encryption," IETF RFC 8152, July 2017.
[8] J. Richer, "OAuth2.0 Token Introspection," IETF RFC 7662, Oct. 2015.
[9] M. Vuini *et al.*, "OSCAR: Object Security Architecture for the Internet of Things," *Proc. IEEE 15th Int'l. Symp. World of Wireless, Mobile and Multimedia Networks (WoWMoM)*, June 2014, pp. 1–10.
[10] W. Shang *et al.*, "NDN-ACE: Access Control for Constrained Environments over Named Data Networking," Technical Report NDN-0036, NDN Project, Revision 1, Dec. 2015; accessed on 13-Jan-2018: https://named-data.net/wp-content/uploads/2015/12/ndn-0036-1-ndn-ace.pdf
[11] L. Zhang *et al.*, "Named Data Networking," *ACM SIGCOMM Comput. Commun. Review*, vol. 44, no. 3, July 2014, pp. 66–73.
[12] J. Burke *et al.*, "Securing Instrumented Environments over Content-Centric Networking: The Case of Lighting Control and NDN," *Proc. IEEE Conf. Computer Communications (INFOCOM Workshops)*, Apr. 2013, pp. 394–98.

Biographies

Victoria Beltran received her B.S. and M.S. degrees in computer engineering from the University of Murcia. She holds a Ph.D. in telematics engineering from the Universitat Politècnica de Catalunya in Barcelona. She has been associate professor at the University of Murcia and a researcher at Bell-Labs Alcatel-Lucent, Orange Labs, Columbia University, Institut Mines-Telecom, and Singapore University of Technology and Design.

Antonio F. Skarmeta received the M.S. degree in computer science from the University of Granada and B.S. (Hons.) and the Ph.D. degrees in computer science from the University of Murcia, Spain. Since 2009 he has been a full professor at the same University. He has been the head of the research group ANTS since its creation in 1995 and a member of TDL. His main research focus is on networking, security and IoT.