NDN Technical Report NDN-0036, 2015. http://named-data.net/techreports.html
Revision 1: December 21, 2015

NDN-ACE: Access Control for Constrained Environments over Named Data Networking

Wentao Shang*, Yingdi Yu*, Teng Liang†, Beichuan Zhang† Lixia Zhang*

*University of California Los Angeles
{wentao, yingdi, lixia}@cs.ucla.edu
†University of Arizona
{philoliang, bzhang}@cs.arizona.edu

***Abstract*—The access control problem, including authentication and authorization, is critical to the security and privacy of the IoT networks. In this paper we present NDN-ACE, a lightweight access control protocol for constrained environments over Named Data Networking (NDN). NDN-ACE uses symmetric cryptography to authenticate the actuation commands on the constrained devices but offloads the key distribution and management tasks to a more powerful trusted third party. It utilizes hierarchical NDN names to express fine-grained access control policies that bind the identity of the command senders to the services they are authorized to access. The key management protocol in NDN-ACE allows the senders to update their access keys periodically without requiring tight synchronization among the devices. The evaluation shows that NDN-ACE has fewer message exchange and uses fewer components in the overall network architecture compared to the IP-based alternatives. The "proof-of-concept" prototype also demonstrates the feasibility and efficiency of the NDN-ACE framework.**

I. Introduction

Access control is a critical component in the "Internet-of-Things" (IoT) systems for securing the access to critical and/or sensitive resources. OWASP lists "insufficient authentication & authorization" as one of the top 10 IoT vulnerabilities in 2014 [1]. In this paper, we focus on one specific aspect of access control in IoT systems: *actuation*, where one sends a command to a remote device to invoke certain actions on the device. An actuation system has stringent requirements on *latency* and *authentication*. It must deliver actuation command from a sender to a receiver in a timely manner, and the command receiver must authenticate the command correctly. However, IoT's constrained environment makes it difficult to satisfy these two requirements. Packet losses become common when low-power devices communicate with each other over wireless channels. Devices may not be always online due to energy saving. It is also infeasible to implement complex authentication policy on IoT devices with limited computation power. Moreover, a IoT system may involve a large number of devices with constrained configuration interfaces, increasing the difficulty of configuration updating.

Most existing solutions based on the IP architecture either twist the network layer or build additional layers to address these issues. On the other hand, some ongoing work [2], [3] already suggested that the data-centric communication model could be a better fit for IoT than IP's channel-based communication model. In this paper, we present *NDN-ACE*, an IoT actuation framework that is built directly over a data-centric Internet Architecture: *Named Data Networking* (NDN) [4]–[6]. NDN-ACE seals actuation command into a network-level packet. The packet carries sufficient context for command authentication and is protected through an HMAC chain [7], so that a command receiver can immediately authenticate the command in a packet regardless of where and how the packet is delivered. As a result, NDN-ACE can achieve low latency in command delivery and enforce least privilege in command authentication with low overhead of computation and communication.

This paper is organized as follows. Section II provides the background about NDN and IoT environments. Section III describes the application scenario where NDN-ACE can be applied and design goals. Section IV describes the details of the NDN-ACE protocol design. Section V analyzes the security properties of the protocol. Section VI discusses miscellaneous design and implementation issues. Section VII compares NDN-ACE with two related proposals from the IETF ACE WG. Section VIII presents our prototype implementation. Section IX briefly reviews related works. In Section X we conclude the paper and discuss the future work.

II. Background

A. Internet of Things

"Internet of Things" (IoT) generally refers to the interconnection of different types of computing devices to support various kinds of monitoring and control applications. An IoT network usually consists of a large number of devices communicating through wireless channels. Driven by the low manufacturing cost and large-scale deployment, the IoT devices are typically equipped with limited computing power (i.e., 10s to 100s of MHz CPU, 10s to 100s of KB memory and flash capacity [8]). The energy constraint on IoT devices also leads to the wide adoption of low-energy radio technologies, these low-power and lossy networks (LLN [9]) usually provide low bandwidth (10s to 100s of kbps) and suffer from interference-induced packet loss, which makes network communication an expensive operation.

A major type of IoT applications is actuation systems. In such systems, a device may receive a control command to

perform certain operation. Since many IoT devices have close interaction with the ambient environment, and some of them (e.g., smoke detectors and fire alarms) are even assigned life-critical tasks, it is imperative to secure actuation system, so that only control commands from authorized entities can be executed. However, the resource-constrained nature of IoT devices prohibits security solutions with high computation and communication overhead. Moreover, the massive scale of the IoT deployment also implies that a proper security solution must be highly scalable.

Traditional IP-based IoT solutions often rely on a secure channel (e.g., TLS [10] and its datagram variant DTLS [11]) to protect the communications among the devices. However, such channel-based security model cannot express the application-specific privileges. Consequently the receiver side applications need to implement separate command authentication logic on top of the secured channels. But IoT device's constrained computation power restrict the complexity of logic needed to provide fine-grained access control.

Maintaining a secure channel inevitably introduces energy consumption, while communication pattern of actuation system is usually intermittent and involves only a few packet exchanges. Establishing a secure channel on demand does not alleviate the problem either, due to the overhead of setting up a channel, as well as the key negotiation which introduces additional delays undesirable for certain actuation systems. Moreover, channel-based communication performs poorly in a lossy environment; packet loss detection and recovery may significantly affect the latency and consume more computation resources.

B. Named Data Networking

Named Data Networking (NDN) [4]–[6] is a new Internet architecture that provides data-centric communication semantics at the network layer. Communication in NDN is modeled as retrieving individual piece of data. To enable this communication model, NDN names each individual piece of data under a hierarchical namespace. A data producer puts the data and its name in a network level packet, called a *Data packet*. A data consumer sends a request, called an *Interest packet*, that carries the name of the desired data. Based on the requested data name, the network forwards each Interest packet toward the potential location of the corresponding data.

1) Content-based Security: NDN mandates producer's digital signature on each Data packet, to make each Data packet individually verifiable. As a result, the authenticity of the data is independent from where and how the data is retrieved. To facilitate the data authentication, each data packet contains a *SingatureInfo* field indicating the name of the public key that can verify the signature. A public key is just another type of data with its own name, and since every Data packet has a signature, a Data packet carrying a public key becomes a certificate and can be retrieved and verified in the same way as other Data packets.

To authenticate a Data packet, one needs a trust model that defines which keys are authorized to sign which piece of data and specifies one or more trusted keys to bootstrap the trust. Naming both the data and the keys allows one to define a trust model in terms of the relationship between the data name and the key name [12]. The hierarchical structure makes NDN names highly expressive and can provide sufficient context for data authentication, and our earlier work has demonstrated that one can use NDN name to easily specify privilege at any desired granularity.

Besides authenticity, NDN also supports content-based confidentiality by encrypting the Data packets directly. Data producers encrypt the data at the time of production, and distribute the corresponding decryption key to the authorized consumers. In this way, the confidentiality of data does not depend on the intermediate devices, achieving true end-to-end confidentiality.

2) Actuation over NDN: The Interest/Data semantics of NDN resembles the request/response semantics of the actuation systems. Burke et al. [13] proposed a framework for supporting actuation commands over NDN's Interest/Data communication in the context of lighting control systems. They use a special *Signed Interest* packet to carry the requested command in the name. Unlike normal Interest packet, a Signed Interest also carries a signature and meta information about the signature, allowing the receiver of the command to authenticate the command. Beside the digital signature, a Signed Interest also includes a timestamp and a nonce to prevent replay attacks. Note that to detect such attacks, either the two communication ends have to synchronize their clocks or the receiving end needs to keep per-sender state about the most recently received timestamp.

III. Application Scenario

We target an application scenario where all devices are interconnected via NDN. A constrained client (C) invokes some privileged service offered by a constrained actuator (A) using the *Signed Interest* mechanism [14] over the network. For example, a smart switch may issue remote commands to turn a smart lamp on and off. The client knows which service it wants to access, while the actuator does not know the identity of the client until it sees the request command. Due to the resource limitation, the actuator delegates the authentication and authorization tasks to a trust third party called *Authorization Server (AS)*, which typically runs on a dedicated machine with much more computing power than the constrained IoT devices. Both the client and the actuator may be offline at any time due to scheduled hibernation, while the AS is assumed to have enough power to stay online all the time except when the network or the host fails.

Assumptions: We assume that the client and the actuator have established trust relationships with the AS using some bootstrapping process. The devices and the AS also exchange their public keys which can be used to authenticate the communications between them. The actuator trusts the AS to follow the access control policies defined by the applications and issue the access keys to the authorized clients on its behalf. The access keys are derived from the pre-shared secret established between the AS and the actuator. The client trusts the AS to provide valid access keys for the services it

wants to access. We also assume that all devices and servers on the network have synchronized clocks (e.g., by sharing the same time source). This allows the actuator to detect replayed commands without having to keep track of per-client timestamp in the Signed Interests.

Attacker model: We consider an attacker who can eavesdrop and tamper with packets transmitted over the network, or inject malicious packets into the network. For example, the attacker may record the actuation commands sent over the network and try to extract the authentication information; the attacker may perform man-in-the-middle attacks by intercepting and modifying the content of the packets; the attacker may also inject invalid actuation commands to trick the actuators into performing unauthorized operations. However, we do not consider DDoS attacks such as jamming the physical communication channel or overwhelming the IoT devices with Interest flooding. Neither do we consider any physical attacks on the devices or the network, such as destroying, removing or shutting down the IoT devices or network routers. Nor do we consider the case where the attacker breaks into an authorized device and takes over the control of the IoT applications running on that device.

IV. Protocol Design

In this section we describe the design of the NDN-ACE protocol. We start with an overview of the protocol operations. Then we describe the details of authorization and command authentication in NDN-ACE. In the end we discuss the issues in key management such as key distribution and update.

Here is a list of design goals that we want to achieve in order to make NDN-ACE applicable to the IoT environments:

- Authentication and authorization for actuation commands between the client and the actuator.
- Fine-grain access control policies.
- Low overhead in the communication messages and the in-memory state on the constrained nodes.
- Low computational cost on cryptographic operations.

A. Overview

The NDN-ACE protocol consists of three major parts of information flow between different devices as is depicted in Figure 1:

A–AS The actuator delegates the access control tasks to the trusted AS by sharing a per-service secret key (seed) with the AS. The AS obtains new seeds periodically from the actuator to replace the old seed and extend the delegation relation. The seed is always encrypted when distributed over the network.

C–AS When the client requests for access to a service, the AS authenticates its identity and generates an access key for the client using the corresponding service seed. The client obtains new access keys periodically from the AS to replace the old key and refresh its access privilege. The access key is always encrypted when distributed over the network.

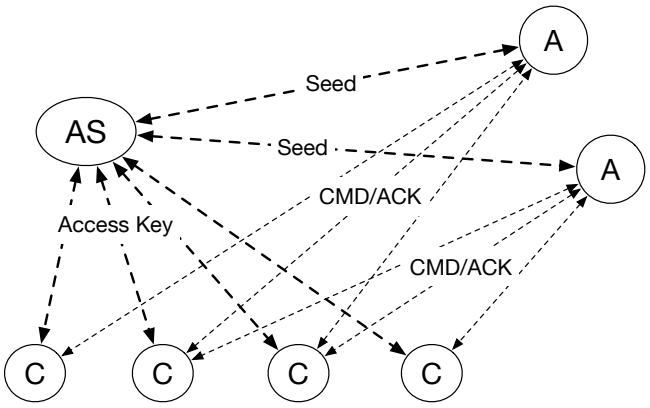

Fig. 1: NDN-ACE architecture and information flow

C–A The client sends authenticated commands to the actuator to access some service. The command Interest is HMAC-signed with the sender's access key for that service. The signature proves the client's ownership of a valid access key and hence indicates that the client has been authorized by the AS.

Every device (C, A and AS) on the network has a device prefix, which is typically constructed by concatenating the network prefix with the device identifier that is unique on the local network. The device identifier can be a name assigned by the network administrator, the serial number created by the device manufacturer, or some random ID generated by the device upon first use. A few examples are shown in Figure 2. In this paper we assume all the devices are connected to the same network and share a common prefix. The framework can also be extended to support cross-domain authentication and authorization as long as there exist trust relationships between different domains.

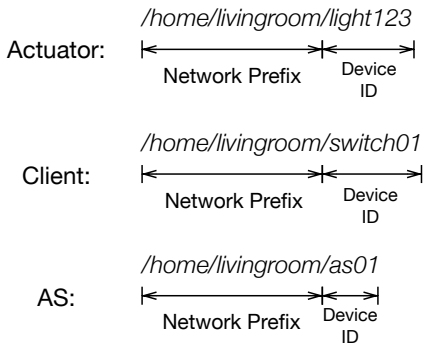

Fig. 2: Naming convention for the devices

The actuator offers some service over the NDN-based IoT network by announcing its device prefix to the routing system and waiting for incoming command Interests with the service name. The service name usually comprises the routing prefix followed by the service ID. Take the examples shown in Figure 3: /home/livingroom/light123/setStatus command can set *light123* to either ON or OFF; /home/livingroom/light123/

readStatus command will return the current status of *light123* to the client.

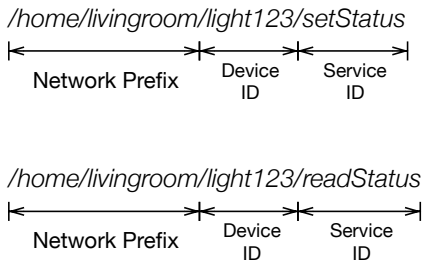

Fig. 3: Naming convention for the service

B. Authorization

The actuator generates a symmetric key as the per-service seed and share that key with the AS in order to delegate the access control tasks. Each seed has a sequence number that gets incremented by the actuator every time the seed is updated. The value of the seed can be computed by some *psuedo-random function* (PRF) that takes the service name, the seed sequence number and a master secret (or some other source of entropy) as the input.

Before accessing the service, the client needs to obtain the access key as the proof of authorization from the AS. The access key is derived from the service seed and cryptographically bound to the service name and the client identity. The access key also has a sequence number that is incremented by the AS every time the key is updated.

To compute the access key, the AS first constructs an NDN name that expresses the access control policy for the client using the following naming convention (the words *SEED* and *KEY* are special name components that are fixed for all names):

Fig. 4: Naming convention for the access key

The access key name in Figure 4 expresses the following authorization information: the client *switch01* is authorized to access the *setStatus* service on the actuator *light123* during the validity period of the corresponding service seed whose sequence number is *456*; the sequence number of this access key is *789*.

The value of the access key is computed as the *HMAC* over the access key name (binary encoded using the NDN name wire format [15]), using the seed as the HMAC key. The access key and the corresponding sequence numbers (of both the access key and the seed) are returned to the client using the key distribution mechanism that we will describe in Section IV-D.

C. Command Authentication

The client accesses the services offered by the actuator using Signed Interests [14]. The name of the Signed Interest specifies the targeted service name (e.g., /home/livingroom/light123/setStatus) and may contain additional components to encode command parameters if necessary (e.g., /home/livingroom/light123/setStatus/on). The Interest also carries an HMAC signature signed with the client's access key. Since the actuator does not maintain per-client key material, the Interest itself must carry enough information so that the actuator can recompute the access key from the service seed. To achieve that, we put the seed sequence number S_S, the client identifier and the access key sequence number S_K in the *SignatureInfo* component of the Signed Interest. By convention, we encode these three pieces of information as an NDN name with three components so that the actuator can easily extract each of them.

When the actuator receives the Signed Interest, it extracts the sequence numbers and client ID from the *SignatureInfo*. The actuator checks that S_S matches the current sequence number of the local seed it is maintaining. Then it constructs the access key name following the naming convention in Figure 4 and computes the HMAC of the resulting name using the local seed as the key, which returns the access key associated with this command. Finally the actuator verifies the HMAC signature of the Signed Interest using the computed access key.

If the command authentication succeeds, the actuator will execute the command and may reply to the client with an acknowledgement (ACK). This ACK is HMAC-signed by the same access key that is used to sign the command, which allows the client to verify the authenticity of the ACK. If the command authentication fails, the actuator may reply with a negative acknowledgement (NACK). The NACK may contain an error code and other relevant information that can help the client recover from the failure. An example of the authenticated command operation is illustrated in Figure 5.

D. Key Management

A common challenge in authentication and authorization protocols is the key management, e.g., how to securely distribute the keys to all parties and how to refresh the keys periodically. In this subsection, we describe the key management mechanism in NDN-ACE. We first talk about how to refresh the seed and the access keys. Then we discuss the protocol for distributing those keys over NDN.

1) Key update: Updating the seeds and the access keys periodically effectively protects the system from key compromises that are hard to detect. In NDN-ACE, each access key is associated with a timer and a usage counter that are maintained by the recipient of the key. Whenever an access key is used to compute a signature, the counter of that key is incremented. The client may choose an upper bound on the number of times that a key can be used and request a new key from the AS when that upper bound is reached. At the same time, the application may choose a upper limit on the lifetime of the key. If the key has not been updated for that amount of time, the client may

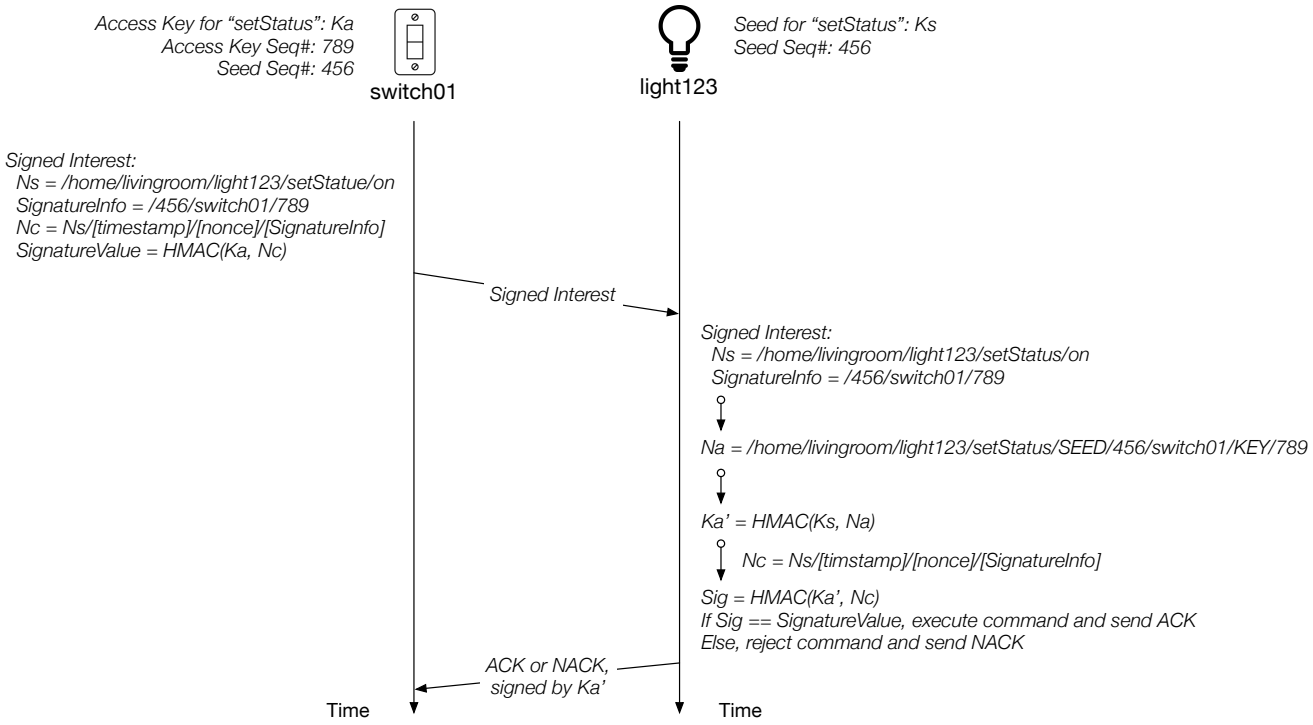

Fig. 5: Command authentication in the light actuation example

request a new key regardless of the usage counter value. The new key has the same name as the previous key, except the sequence number is increased by one.

A similar mechanism is used by the AS to control the update of the seeds generated by the actuator. However, unlike the AS which is always online, the actuator often has to go to hibernation periodically to save power. As a result, the counter-based update is no longer feasible since the actuator may not be available when the update is triggered. Consequently, the AS only needs to update the seed periodically based on a timer that is carefully aligned with the actuator's sleeping schedule.

When the seed is updated, all the access keys derived from the previous seed must also be updated regardless of their current timers or counter values. This creates a "key rollover" problem where the clients may be unaware of the seed update and continue to use the old keys to issue commands.[1] NDN-ACE solves this problem by having the actuator keep the last seed and its sequence number after a new one is generated. If a client issues a Signed Interest that contains the sequence number of the previous seed in the *SignatureInfo*, the actuator will verify the command using the old seed. If the verification succeeds, the actuator will execute the command and then send back an acknowledgement (ACK) notifying the client that its access key is outdated and also telling it the sequence number of the current seed. The ACK serves as a feedback that instructs the client to update its key immediately. As a simple optimization, all sleeping clients should always update their access keys after wake up, since it is very likely that the seed has changed while they are sleeping.

2) Key distribution: When a seed or an access key is updated, it needs to be securely distributed to the key recipient. This is essentially the content distribution problem that has been studied in many prior works [16], [17]. A common solution is to encrypt the content so that only the recipients who have the decryption key can access the content. NDN-ACE uses the same technique for distributing the seed and the access key over an untrusted network. Here we use the access key distribution as an example to illustrate the process. The distribution of the seed is mostly the same.

The key distribution process in NDN-ACE requires only one round of Interest/Data message exchange. We use the classic Diffie-Hellman key exchange algorithm [18] to derive the encryption key. When the client requests a new access key, it generates a Diffie-Hellman public key on the fly and sends a Signed Interest to the AS including its DH public key as the command parameter. The Interest is signed by the client's RSA/ECDSA private key to prevent impersonation attack.

Upon receiving the request, the AS performs the following steps of operations:

1) It verifies the signature of the Signed Interest using the client's RSA/ECDSA public key.
2) If the verification succeeds, it generates its own Diffie-Hellman public key, computes the shared secret from the two DH public keys (the other one is contained in the request), and derives a key encryption key (KEK) using a standard key derivation function.
3) It then generates the new access key and updates the sequence number as is described in Section IV-B.

[1]Having the AS send out immediate notifications to all affected clients does not work in the constrained IoT environment as the clients may be sleeping when the seed is updated.

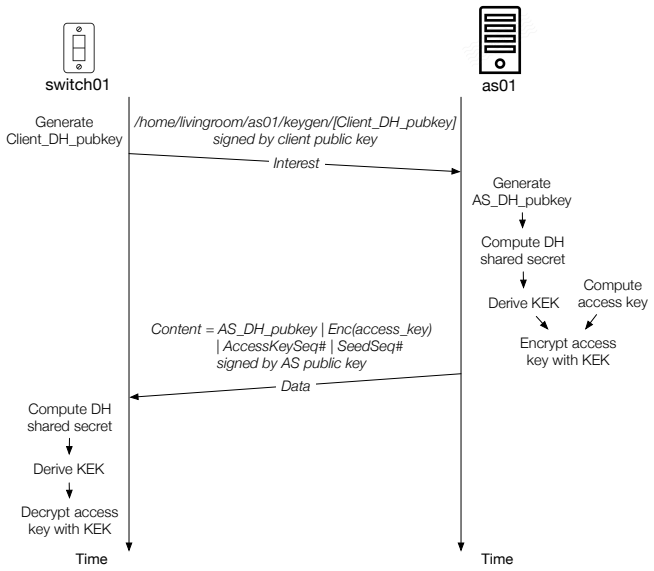

Fig. 6: The distribution of access key from the AS to the client

4) Finally it encrypts the access key using the KEK and returns the encrypted key, together with the sequence numbers (of the new key and the seed) and its own DH public key, to the client. The response is signed by the RSA/ECDSA private key of the AS.

Once getting the response, the client verifies the signature in it using the RSA/ECDSA public key of the AS, then derives the same KEK from the two DH public keys (the other one is contained in the response) and finally decrypts the new access key. The process is illustrated in Figure 6. Note that if the Interest and/or the Data packets get lost during this process, the client must restart the process by generating a fresh ephemeral DH public key and transmitting that key to the AS using a newly-generated Signed Interest (with fresh timestamp and nonce).

V. Security Analysis

In this section we discuss how the NDN-ACE protocol design prevents several types of attacks. Here we assume the natural security properties of the hash function used in the HMAC operations, including:

Non-guessability: without the corresponding signing key, it is computationally infeasible to produce the correct HMAC signature on a given message.

Non-invertibility: given the message and the signature, it is computationally infeasible to reverse the HMAC function and obtain the signing key.

A. Impersonation Attack

An attacker cannot impersonate an authorized client by extracting the victim's access key from the network packets. First, the access key distributed by the AS is encrypted with an encryption key derived from the Diffie-Hellman key exchange between the client and the AS, and the key exchange process is protected from MITM attacks by having the client and the AS sign their packets. Second, the commands do not contain the access key itself, but only the HMAC signatures computed with the access key.

Neither can the attacker impersonate an actuator in order to trick the client or the AS into revealing the access key or the seed to the attacker. First, the seed is generated by the legitimate actuator and distributed to the AS, not vise versa. Second, it is impossible to extract the access key or the seed from the command signature due to the non-invertibility of the HMAC.

NDN-ACE also prevents the attacker from impersonating the AS and obtaining the seed from the actuator because the key distribution process requires the AS to sign its request for the seed. Assuming the actuator has already established trust relationship with the real AS during the bootstrapping phase, this attack can be easily detected by verifying the request signature using the public key of the trusted AS.

B. Privilege Delegation

In NDN-ACE, the client who gets the access keys from the AS are prohibited from further delegating its own access privilege to other parties. This is guaranteed by deriving the access key from the seed using a non-invertible HMAC function. In this way, the client who obtains the access key cannot extract the seed and therefore cannot produce new access keys for other parties.

Technically one could modify NDN-ACE to let the client extend the HMAC chain and produce third-level access keys from its own keys. Doing so would allow the client to further delegate its privilege independently without knowing the seed, while retaining the stateless verifiability at the root of the delegation chain (i.e., the actuator). (There is indeed a framework that implements this style of distributed authorization for cloud applications [19].) However, we feel such chain delegation complicates the simple trust model of NDN-ACE and therefore choose to keep our design simple.

VI. Discussion

In this section, we discuss the design and implementation issues that are not covered in previous sections.

A. Applicability to CoAP

NDN-ACE is designed on top of Named Data Networking and immediately benefits from many features of NDN such as Interest-Data exchange, name-based forwarding and data object security. However, since NDN-ACE can be seen essentially an application-layer security solution, it is not difficult to port NDN-ACE to an IP network by implementing the NDN communication semantics on top of an application layer protocol such as CoAP. CoAP is suitable for supporting NDN-ACE because: first, CoAP is designed for constrained environments; second, both CoAP and NDN supports the same communication semantics that is intrinsic in IoT; third, it is easy to extend CoAP to support object security [20].

In CoAP, the resource names are represented as URIs. The *host* and *port* components are used to route the requests to the

resource server, while the *path* and *query* components identify the resource under the scope of the connected resource server. To distribute the access keys over CoAP, the client can send CoAP requests to the AS with the *path* pointing to the key distribution service (KDS). The *query* components contain the parameter necessary for generating the access key:

coap://as1.livingroom.home:55555/kds?rid=light123&svc=turnOn&sid=switch01&aseq=56789

The responses contain the keys encrypted and signed as in NDN-ACE.

When sending commands, the client can encode the authentication-related name components either in the *path* or in the *query*. The command signature can be base64-encoded to be compatible with the URI encoding:

coap://light123.livingroom.home:55555/turnOn?sseq=1234&sid=switch01&aseq=56789&sig=QUJDMTIz

The actuator will simply parse the CoAP request and authenticate the signature as before. Note that in this design we do not use DTLS to protect the communication channel because NDN-ACE is inherently based on object security.

B. Name Engineering

Naming is the center piece in the design of NDN-ACE. The devices use names to express their identities and the services they provide. The AS uses names to express the access control policies from which the access keys are derived. Furthermore, the versions of the keys are also expressed explicitly in the names as sequence numbers. Given the importance of the names, it is necessary that the naming scheme in NDN-ACE be carefully designed in order to accommodate the requirements from both the applications and the underlying networks.

There are two primary concerns with the NDN-ACE naming scheme as described in this paper. The first one is the use of verbose names in the constrained network. ASCII encoding of human-readable components can lead to excessively long names, which increase the processing cost on the network nodes and result in large packet size. There are multiple ways to address this "long name" issue. First of all, applications can define efficient encodings for the name components that do not have to be human-readable. Alternatively, if the devices can afford some extra computational cost, they can compress the name suffixes that are not related to routing and intended to be interpreted only by the applications.

The second major concern with the NDN-ACE naming scheme is privacy. In some application scenario, it may be desirable to avoid exposing the interface or the semantics of some services in the network packets. In NDN-ACE, the service commands are encoded in cleartext in the names by default. This may allow the attacker to sniff packets and extract sensitive information about the services on the network. The general solution is to apply encryption on the name components that need to be protected.For example, the client may encrypt the command name, the timestamp, the nonce and the signature info into a single opaque component and attach the signature after that.

C. AS Fault Tolerance

The authorization server (AS) in NDN-ACE is a key component in the authentication and authorization process. It maintains the service seeds for the actuators it serves. It also has to stay online to answer the key update requests from the client. The failure of the AS would paralyze the access control framework and the application operations. To avoid this single-point-of-failure issue, we can deploy backup servers to replace the master AS when it fails. Note that the AS does not keep any soft state except the seed information. Therefore when the backup server takes over, it can simply request new seeds from the actuators and use them to derive new access keys for the clients.

To allow automatic hand-over during failure recovery, there need to be pre-established trust relationships among all IoT devices and authorization servers. This can be achieved by adding a common trust anchor as one level of indirection. When a new node (device or server) joins the network, the bootstrap process will install the trust anchor on the new node and also use the trust anchor to sign the identity certificate for that node. This allows the new node to be trusted by all the other nodes on the network who have the trust anchor installed already.

VII. Evaluation

In this section, we compare NDN-ACE with two access control solutions proposed by the IETF ACE WG: DCAF [21] and OAuth 2.0 IoT profile [3]. DCAF is a simple authentication and authorization framework that allows two constrained nodes to establish DTLS connections by delegating the generation and distribution of DTLS pre-shared keys (PSK) to trusted third parties. DCAF further divides the role of authorization server (AS) into two parts: a Client Authorization Manager (CAM) and a Server Authorization Manager (SAM) acting on behalf of the client and server nodes, respectively. DCAF assumes the existence of pre-established DTLS connections between the client and its CAM, and between the server and its SAM. To gain access to the resources on the server, the client requests a DTLS PSK from the SAM via its CAM. The PSK is wrapped by the SAM into an access ticket which contains either the encrypted PSK or the information to help the server to derive the PSK. Both the key and the ticket are transferred back to the client through its CAM over secured DTLS channels. During DTLS PSK exchange, the client presents the ticket to the server who will derive the PSK from the ticket and then set up the DTLS channel.

OAuth 2.0 IoT profile adapts the HTTP-based OAuth 2.0 protocol to the CoAP-based IoT environments. It uses OAuth 2.0 Proof-of-Possession (PoP) tokens [22] to carry authentication and authorization information. When the client requests access privilege to the Resource Server (RS), the Authorization Server (AS) creates a PoP token that is encrypted for the RS and signed by the AS. The token contains authorization information (privilege, validity period, etc.) and the access key information (encrypted symmetric key or public key fingerprint). When issuing the resource request, the client includes the PoP token in the request and uses the corresponding access

TABLE I: Protocol operations for establishing command communication

Protocol	NDN-ACE	DCAF	OAuth IoT Profile
Messages	CMD/ACK	DNS lookup for server + DTLS PSK exchange + CMD/ACK	DNS lookup for RS + CMD/ACK
Total RTT if no packet loss	1	≥ 4	≥ 2

key to sign the request. Upon receiving the request, the RS can verify the authenticity of the token, extract the access key from the token, and then verify the signature of the request. If all verification succeed, the RS will return the protected resource to the client or carry out the requested actuation operation.

In the rest of this subsection, we compare the three authentication and authorization protocols on two different aspects: communication overhead and architecture complexity.

A. Communication Overhead

Table I shows the comparison of message exchanges required to start the actuation command communication (after the sender obtains the authorization from the AS). Both DCAF and OAuth IoT profile need to perform DNS lookup if they use the URL instead of IP address to identify the command receiver. This will take at least 1 RTT between the sender and the DNS server. After that, DCAF requires 2 extra RTTs of DTLS PSK exchange. Finally the command communication (CMD/ACK) takes 1 more RTT. Assuming there is no packet loss during network transmission, NDN-ACE is able to start application-layer communication immediately, while DCAF and OAuth profile have to take additional RTTs before sending a single-RTT actuation command.

One might argue that the DNS resolution results can be cached by the client for future use, and that the DTLS connection in DCAF can be preserved and reused by future actuation commands. However, maintaining these soft states can be a challenging task on the resource-constrained IoT devices. The device has to either limit the total number of communicating peers (and hence sacrificing scalability) in order to keep the size of those states within its storage capacity, or drop those states according to some replacement policy and reestablish them in the future if needed. NDN-ACE, on the contrary, does not incur any of these overhead.

B. Architecture Complexity

The architectural complexity has significant impact on the feasibility of the IoT solutions. Figure 7 compares the overall protocol stacks that an IoT device needs to implement in order to support NDN-ACE and the IP-based solutions (DCAF and OAuth IoT profile). The NDN-based IoT stack is much simpler because NDN inherently offers the communication semantics that is intrinsic to the IoT environments. Therefore it is possible to build IoT frameworks and applications directly on top of the NDN layer. In contrast, the IP-based IoT protocol stack has to integrate multiple dependent protocols on top

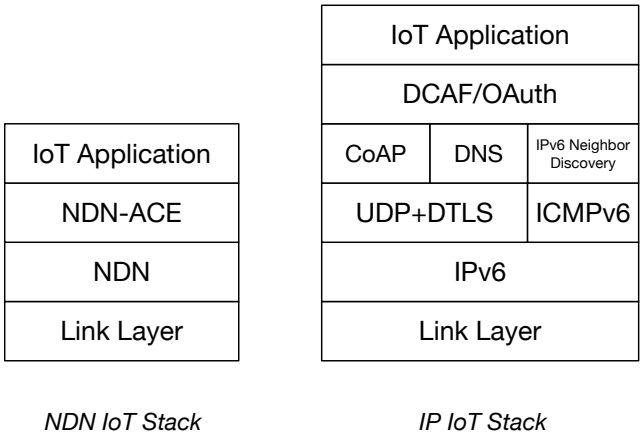

Fig. 7: Comparison of NDN-based and IP-based IoT stack

of IP, such as UDP, DTLS, IPv6 Neighbor Discovery, DNS and CoAP, before it is possible to express the application layer logics such as authentication, authorization and resource access. This extra stack complexity not only leads to inflated program size for each IoT node, but also introduces multiple dependencies on the network infrastructure, such as the DNS service. Both issues will create a significant burden on the implementation and deployment of the IoT applications.

VIII. Prototype

We have implemented NDN-ACE in Python and deployed the prototype on the Raspberry Pi 2 platform as a proof of concept. The reason we choose Raspberry Pi instead of more constrained devices is to benefit from fast prototyping, easy testing and hardware extensibility. The device we are using is equipped with a quad-core ARM Cortex-A7 CPU and 1GB RAM. The CPU normally operates at 600MHz but can be boosted to over 900MHz under high workload (thanks to the automatic overclocking feature of Raspberry Pi). Each Raspberry Pi has a WiFi adaptor connected through a USB port, which is used to interconnect with other devices on the network. All the peripheral sensors are connected through the GPIO pins and can be accessed directly by the applications on the Raspberry Pi.

As a demonstration of the functionality of NDN-ACE, we implement a simple lighting control system that allows remote devices to the LED lights attached to the Raspberry Pi. The test environment comprises of 3 Raspberry Pi devices functioning as the client, the actuator and the AS, respectively. To simplify the network configuration, we connect all devices to the same WiFi AP and use multicast to forward Interest packets among those devices. We use the PyNDN2 [23] library to implement the NDN-related operations. In our demo application, each command Interest received by the actuator is passed to a callback function that processes the command in 6 steps:

- **Interest packet parsing**: parsing the received command Interest and reading various fields.
- **Authentication and authorization**: verifying the command signature using the access key.

TABLE II: Per-command CPU time break down for command processing

Step	CPU Time (ms)	Percentage (%)
Interest parsing	11.876	58.79
Authentication	0.766	3.79
GPIO	0.139	0.69
ACK generation	6.352	31.44
ACK signing	0.355	1.76
ACK sending	0.714	3.53
Total	20.202	100

- **GPIO access**: executing the requested commands through the GPIO interface.
- **ACK packet generation**: creating the ACK packet and filling out various fields.
- **ACK signing**: signing the ACK packet using the access key.
- **ACK sending**: pushing the ACK packet to the sending queue.

Although our prototype is not optimized for performance, we are still interested in how much CPU time is spent at each step of the command processing pipeline. To evaluate that, we program the client to send out 1000 command Interests to the actuator and use a Python profiler to measure the fine-grained CPU time usage on each statement of the command processing function. The client sends its commands at a low sending rate so that the actuator's CPU is not overwhelmed and keeps operating at 600MHz throughout the experiment.

Table II summarizes the measurement result. The total CPU time of per-command processing at the actuator is 20.2 ms. Surprisingly, about 90% of the CPU time is spent on Interest parsing and ACK packet creation. The NDN-ACE command verification and ACK signing only take about 5% of the total CPU time, which shows the efficiency of the HMAC-based authentication procedure. Using a different library or programming language (C or C++) may improve the performance of NDN-related operations (and the overall performance of command processing). We leave that as future work.

IX. Related Works

The idea of using cryptography for authentication and authorization in computer networks dates back to the 1970s, about the same time when the public key crypto algorithms were first published. Needham et al. [24] proposed a basic framework that uses symmetric or asymmetric encryption algorithms for authentication in distributed networks. The symmetric encryption variant relies on an *authorization server* (AS) that maintains shared secrets with both communicating peers A and B. When A initiates the authentication, AS creates a session key K for A and B, encrypts K with both A's and B's secrets, and passes the encrypted key back to A. A further passes the encrypted key to B (without knowing B's secret) and authenticate B's identity by checking whether B is able to decrypt K. After that the two parties can use K to start secured communication.

Inspired by the Needham authentication protocol, Kerberos [25] is one of the oldest standardized authentication protocols for computer networks. The current version (V5) of Kerberos is defined in [26]. Kerberos implements the same mechanism as the Needham protocol but extends the framework to allow single sign-on and cross-domain authentication. However, both protocols only support authentication: the authorization semantics are left to the upper layers to define.

OAuth is a widely-used authorization protocol that enables third-party access to protected resources in Web applications. It is defined on top of the HTTP semantics: the authorization information is carried in the HTTP packets in the form of tokens (similar to the HTTP Cookies). In OAuth 1.0 [27], each token is coupled with a shared secret and the authenticated HTTP requests contain an HMAC signature that proves the ownership of the token to the resource server. In OAuth 2.0 [28], the requirement for using signatures in the requests is removed from the base framework, which makes OAuth 2.0 completely dependent on the security protection of the underlying channel (e.g., HTTPS [29]). Later the IETF OAuth Working Group started to work on a new architecture that replaces the *bearer* tokens in OAuth 2.0 with *proof-of-possession* (PoP) tokens using a Kerberos-style key distribution mechanism that do not rely on secured channels [22].

ACE [30] is a newly established working group at IETF that focuses on developing authentication and authorization solutions for constrained environments, assuming CoAP (with DTLS protection) as the resource access protocol. Based on the experience from existing standards such as Kerberos and OAuth, the working group has established a general architecture where the client (C) and the resource server (RS), both running on constrained nodes, rely on authorization servers (CAS and AS, respectively) for communicating authentication and authorization information on their behalf [31]. NDN-ACE can be viewed as an instantiation of this general framework that combines the CAS and the AS into a single role. The ACE-WG is currently working on two different proposals, DCAF [21] and OAuth 2.0 profiling for IoT [3], which we have already described in Section VII

OSCAR [32] is an object security architecture for content distribution over constrained IoT devices. Its design philosophy is greatly influenced by CCN/NDN: instead of securing the communication channel, OSCAR secures the data object directly by encrypting and signing the protected resource and hence provides end-to-end security. Like many other related works, OSCAR leverages a three-party authorization architecture where the constrained devices trust the more powerful authorization servers for access token management. However, OSCAR is CoAP-based and still relies on DTLS channels for key distribution. Unlike NDN-ACE, OSCAR does not consider the access control under the semantics of device actuation operations.

Macaroons [19] is a decentralized authorization protocol for cloud environments. Although the design goals and the targeted applications are different, Macaroons and NDN-ACE use the same security structure: HMAC chaining. In Macaroons, the access control policies are expressed using a construct of layered *caveats*. Each layer is HMAC-signed using the signature of the previous layer as the signing key, and the final signature is attached to the macaroon. The first layer is

signed by the root key created by the original issuer. This layered structure with HMAC chaining allows the recipient to further delegate the privilege to other parties by adding more caveats to the macaroon and extending the HMAC chain which is highly useful for cloud applications. The key difference between Macaroons and NDN-ACE is that NDN-ACE solely leverages the NDN naming hierarchy to express the access control policies and the delegation of authorization privilege, instead of inventing an additional data structure for that purpose. NDN-ACE also simplifies the authorization process by allowing only one level of delegation (from AS to command sender), which is sufficient for our targeted applications.

X. Conclusion

In this paper we describe NDN-ACE, an authentication and authorization framework for device actuation in NDN-based constrained environments. NDN-ACE expresses fine-grained access control policies using hierarchical names and cryptographically binds the access key to the policy. It offloads the distribution of access keys to a trusted Authorization Server in the network while still allows the command receiver to verify the command signature without maintaining per-sender access key information or contacting the AS for validation. NDN-ACE achieves command communication with minimum delay using a much simpler protocol architecture compared to the IP-based alternatives. The microbenchmark based on a prototype implementation indicates that the cost of authentication operations in NDN-ACE is negligible compared to that of packet manipulation.

NDN-ACE currently only supports point-to-point command communications. In the future, we plan to extend the NDN-ACE framework to also support multipoint control applications (e.g., a single command to turn on a group of lights). Multipoint command communication requires an efficient command delivery mechanism adapted to the constrained environments and a scalable key management protocol that can distribute the seed to multiple receivers in the same service group. We will also improve the prototype implementation of NDN-ACE and make it available to more IoT applications.

References

[1] OWASP, "OWASP Internet of Things Top 10 Project," https://www.owasp.org/index.php/OWASP_Internet_of_Things_Project, accessed: Oct 2, 2015.

[2] Z. Shelby, K. Hartke, and C. Bormann, "The Constrained Application Protocol (CoAP)," Internet Requests for Comments, RFC Editor, RFC 7252, June 2014. [Online]. Available: http://www.rfc-editor.org/rfc/rfc7252.txt

[3] L. Seitz, G. Selander, E. Wahlstroem, S. Erdtman, and H. Tschofenig, "Authorization for the Internet of Things using OAuth 2.0," Internet Draft, IETF Secretariat, Tech. Rep. draft-seitz-ace-oauth-authz-00, October 2015. [Online]. Available: http://www.ietf.org/internet-drafts/draft-seitz-ace-oauth-authz-00.txt

[4] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard, "Networking Named Content," in *Proceedings of the 5th International Conference on Emerging Networking Experiments and Technologies*, ser. CoNEXT '09. New York, NY, USA: ACM, 2009, pp. 1–12. [Online]. Available: http://doi.acm.org/10.1145/1658939.1658941

[5] NDN Team, "Named Data Networking (NDN) Project," Named Data Networking Project, Technical Report NDN-0001, October 2010.

[6] L. Zhang, A. Afanasyev, J. Burke, V. Jacobson, k. claffy, P. Crowley, C. Papadopoulos, L. Wang, and B. Zhang, "Named Data Networking," *ACM SIGCOMM Computer Communication Review (CCR)*, vol. 44, no. 3, pp. 66–73, Jul 2014.

[7] H. Krawczyk, M. Bellare, and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication," Internet Requests for Comments, RFC Editor, RFC 2104, February 1997. [Online]. Available: http://www.rfc-editor.org/rfc/rfc2104.txt

[8] C. Bormann, M. Ersue, and A. Keranen, "Terminology for Constrained-Node Networks," RFC 7228 (Informational), Internet Engineering Task Force, May 2014. [Online]. Available: http://www.ietf.org/rfc/rfc7228.txt

[9] J. Vasseur, "Terms Used in Routing for Low-Power and Lossy Networks," RFC 7102 (Informational), Internet Engineering Task Force, Jan. 2014. [Online]. Available: http://www.ietf.org/rfc/rfc7102.txt

[10] T. Dierks and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2," Internet Requests for Comments, RFC Editor, RFC 5246, August 2008. [Online]. Available: http://www.rfc-editor.org/rfc/rfc5246.txt

[11] E. Rescorla and N. Modadugu, "Datagram Transport Layer Security Version 1.2," Internet Requests for Comments, RFC Editor, RFC 6347, January 2012. [Online]. Available: http://www.rfc-editor.org/rfc/rfc6347.txt

[12] Y. Yu, A. Afanasyev, D. Clark, k. claffy, V. Jacobson, and L. Zhang, "Schematizing Trust in Named Data Networking," in *Proceedings of the 2Nd International Conference on Information-Centric Networking*, ser. ICN '15. New York, NY, USA: ACM, 2015, pp. 177–186. [Online]. Available: http://doi.acm.org/10.1145/2810156.2810170

[13] J. Burke, P. Gasti, N. Nathan, and G. Tsudik, "Securing instrumented environments over content-centric networking: the case of lighting control and NDN," in *Computer Communications Workshops (INFOCOM WKSHPS), 2013 IEEE Conference on*, April 2013, pp. 394–398.

[14] NDN Team, "NDN Packet Format Specification: Signed Interest," http://named-data.net/doc/ndn-cxx/current/tutorials/signed-interest.html, accessed: Oct 2, 2015.

[15] ——, "NDN Packet Format Specification: Name," http://named-data.net/doc/ndn-tlv/name.html, accessed: Oct 2, 2015.

[16] W. Shang, Q. Ding, A. Marianantoni, J. Burke, and L. Zhang, "Securing building management systems using Named Data Networking," *Network, IEEE*, vol. 28, no. 3, pp. 50–56, 2014.

[17] Y. Yu, A. Afanasyev, and L. Zhang, "Name-Based Access Control," Named Data Networking Project, Technical Report NDN-0034, October 2015.

[18] W. Diffie and M. Hellman, "New Directions in Cryptography," *Information Theory, IEEE Transactions on*, vol. 22, no. 6, pp. 644–654, Nov 1976.

[19] A. Birgisson, J. G. Politz, Ú. Erlingsson, A. Taly, M. Vrable, and M. Lentczner, "Macaroons: Cookies with Contextual Caveats for Decentralized Authorization in the Cloud," in *Network and Distributed System Security Symposium*, 2014.

[20] G. Selander, J. Mattsson, F. Palombini, and L. Seitz, "Object Security of CoAP (OSCOAP)," Internet Draft, IETF Secretariat, Tech. Rep. draft-selander-ace-object-security-03, October 2015. [Online]. Available: http://www.ietf.org/internet-drafts/draft-selander-ace-object-security-03.txt

[21] S. Gerdes, O. Bergmann, and C. Bormann, "Delegated CoAP Authentication and Authorization Framework (DCAF)," Internet Draft, IETF Secretariat, Tech. Rep. draft-gerdes-ace-dcaf-authorize-04, October 2015. [Online]. Available: http://www.ietf.org/internet-drafts/draft-gerdes-ace-dcaf-authorize-04.txt

[22] P. Hunt, J. Richer, W. Mills, P. Mishra, and H. Tschofenig, "OAuth 2.0 Proof-of-Possession (PoP) Security Architecture," Internet Draft, IETF Secretariat, Tech. Rep. draft-ietf-oauth-pop-architecture-05, October 2015. [Online]. Available: http://www.ietf.org/internet-drafts/draft-ietf-oauth-pop-architecture-05.txt

[23] NDN Team, "NDN client library with TLV wire format support in native Python," https://github.com/named-data/PyNDN2.

[24] R. M. Needham and M. D. Schroeder, "Using Encryption for Authentication in Large Networks of Computers," *Commun. ACM*, vol. 21, no. 12, pp. 993–999, Dec. 1978. [Online]. Available: http://doi.acm.org/10.1145/359657.359659

[25] B. Neuman and T. Ts'o, "Kerberos: an authentication service for computer networks," *Communications Magazine, IEEE*, vol. 32, no. 9, pp. 33–38, Sept 1994.

[26] C. Neuman, T. Yu, S. Hartman, and K. Raeburn, "The Kerberos Network Authentication Service (V5)," Internet Requests for Comments, RFC Editor, RFC 4120, July 2005. [Online]. Available: http://www.rfc-editor.org/rfc/rfc4120.txt

[27] E. Hammer-Lahav, "The OAuth 1.0 Protocol," Internet Requests for Comments, RFC Editor, RFC 5849, April 2010. [Online]. Available: http://www.rfc-editor.org/rfc/rfc5849.txt

[28] D. Hardt, "The OAuth 2.0 Authorization Framework," Internet Requests for Comments, RFC Editor, RFC 6749, October 2012. [Online]. Available: http://www.rfc-editor.org/rfc/rfc6749.txt

[29] E. Rescorla, "HTTP Over TLS," Internet Requests for Comments, RFC Editor, RFC 2818, May 2000. [Online]. Available: http://www.rfc-editor.org/rfc/rfc2818.txt

[30] IETF, "IETF ace Working Group Charter," https://datatracker.ietf.org/wg/ace/charter/. [Online]. Available: https://datatracker.ietf.org/wg/ace/charter/

[31] S. Gerdes, L. Seitz, G. Selander, and C. Bormann, "An architecture for authorization in constrained environments," Internet Draft, IETF Secretariat, Tech. Rep. draft-ietf-ace-actors-02, October 2015. [Online]. Available: http://www.ietf.org/internet-drafts/draft-ietf-ace-actors-02.txt

[32] M. Vucinic, B. Tourancheau, F. Rousseau, A. Duda, L. Damon, and R. Guizzetti, "OSCAR: Object Security Architecture for the Internet of Things," *CoRR*, vol. abs/1404.7799, 2014. [Online]. Available: http://arxiv.org/abs/1404.7799