## An introduction to identity and access management for the IoT

Security administrators have traditionally been concerned with managing the identities of and controlling access for the people that are part of or interact with their technology infrastructure. Relatively recently, the concept of **bring your own device** (**BYOD**) was introduced, which allowed authorized individuals to associate mobile phones or laptops with their corporate account to receive network services on their personal devices. The allowed network services were typically provided once certain minimal security assurances were deemed to have been satisfied on the device. This could include using strong passwords for account access, application of virus scanners, or even mandating partial or full disk encryption to help with data loss prevention.

The IoT introduces a much richer connectivity environment than BYOD. Many more IoT devices are expected to be deployed throughout an organization than the usual one or two mobile phones or laptops for each employee. IAM infrastructures must be designed to scale to the number of devices that an organization will eventually support, potentially orders of magnitude higher than today. New IoT subsystems will continually be added to an organization as new capabilities arise to enable and streamline business processes.

The IoT's matrixed nature also introduces new challenges for security administrators in industrial and corporate deployments. Today, many IoT solutions are already being designed to be leased rather than owned. Consider the example of a leased radiology machine that records the number of scans and permits operations up to a certain number of entitlements. Scans are reported online, that is, the machine opens up a communications channel from the organization to the manufacturer. This channel/interface must be restricted to only allow authorized users (that is, the lessor or its agents), and only allow the specific machine(s) associated with the lessor to connect. Access control decisions can potentially become very complex, even restricted to specific device versions, time of day, and other constraints.

The matrixed nature of the IoT is taken further by the need to share information. This is true not only of sharing data collected by IoT sensors with third-party organizations, but also with sharing access to IoT sensors in the first place. Any IAM system for the IoT must be able to support this dynamic access control environment where sharing may need to be allowed/disallowed quickly and at a very granular level for both devices and information.

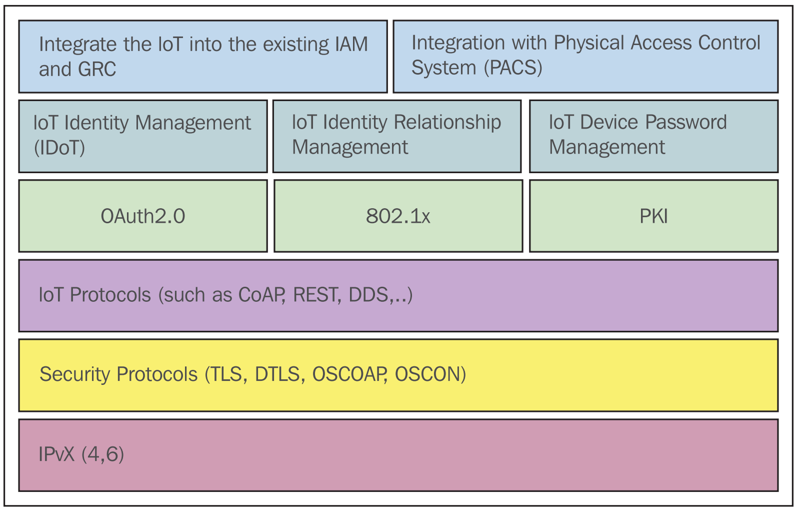
Finally, security administrators must take into account personal IoT devices that attach to their networks. This brings about not only security concerns as new attack vectors are introduced, but also significant privacy concerns related to safeguarding personal information. We have, for example, begun to see organizations support the use of personal fitness devices such as Fitbit for corporate health and wellness programs. In 2016, Oral Roberts University introduced a program that required all freshmen to wear a Fitbit and allow the device to report daily steps and heart rate information to the University's computer systems: `http://www.nydailynews.com/life-style/health/fitbits-required-freshmen-oklahoma-university-article-1.2518842`.

At the other end of the spectrum, a valuable OpenDNS report (reference `https://www.opendns.com/enterprise-security/resources/research-reports/2015-internet-of-things-in-the-enterprise-report/`) showed that in some companies, personnel were beginning to bring unauthorized IoT devices including Smart TVs into the enterprise. These devices were often reaching out to Internet services to share information. Smart devices are frequently designed by manufacturers to connect with the vendor's device-specific web services and other information infrastructure to support the device and the customer's use of it. This typically requires an 802.1x type of connectivity. Providing 802.1x-style network access control to IoT devices requires some thought, since there are so many of these devices that may attach to the network. Vendors are currently working on solutions that can fingerprint IP-based IoT devices and determine whether certain types should be granted access through DHCP provisioning of IP addresses. One may do this, for example, by fingerprinting the operating system or some other characteristic of the device.

IoT IAM is one aspect of an overarching security program that must be designed to mitigate this dynamic new environment, where:

- New devices can be securely added to the network at a rapid pace and for diverse functions
- Data and even devices can share not only within the organization but with other organizations
- Privacy is maintained despite consumer data being collected, stored, and frequently shared with others

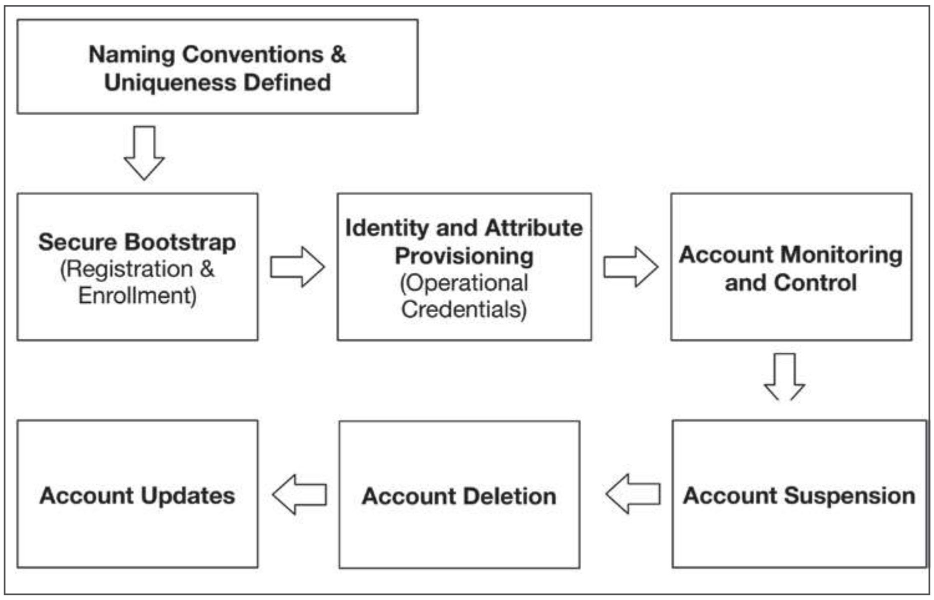
The following figure shows a holistic IAM program for the IoT:


Integrate the IoT into the existing IAM and GRC
Integration with Physical Access Control System (PACS)
IoT Identity Management (IDoT)
IoT Identity Relationship Management
IoT Device Password Management
OAuth2.0
802.1x
PKI
IoT Protocols (such as CoAP, REST, DDS,..)
Security Protocols (TLS, DTLS, OSCOAP, OSCON)
IPvX (4,6)

As noted in in the preceding figure, it is important to line up the new IoT Identity and Access Management strategy with the existing governance models and IT systems in your organization. It may also be worthwhile to consider integration of authentication and authorization capabilities for your IoT devices with your **physical access control systems** (**PACS**). PACS provide electronic means of enabling and enforcing physical access policies throughout your organization's facilities. Frequently, PACS systems are also integrated with **logical access control systems** (**LACS**). LACS systems provide the technology and tools for managing identity, authentication, and authorization access to various computer, data, and network resources. PACS/LACS technologies represent the ideal systems for an organization to begin incorporating new IoT devices in a relatively controlled manner.

## The identity lifecycle

Before we begin to examine the technologies that support IAM for the IoT, it is useful to lay out the lifecycle phases of what we call identity. The identity lifecycle for an IoT device begins with defining the naming conventions for the device; it ends with the removal of the device's identity from the system. The following figure provides a view of the process flow:


Naming Conventions & Uniqueness Defined
Secure Bootstrap (Registration & Enrollment)
Identity and Attribute Provisioning (Operational Credentials)
Account Monitoring and Control
Account Suspension
Account Deletion
Account Updates

This lifecycle procedure should be established and applied to all IoT devices that are procured, configured, and ultimately attached to an organization's network. The first aspect requires a coordinated understanding of the categories of IoT devices and systems that will be introduced within your organization, both now and in the future. Establishing a structured identity namespace will significantly help manage the identities of the thousands or millions of devices that will eventually be added to your organization.

## Establish naming conventions and uniqueness requirements

Uniqueness is a feature that can be randomized or deterministic (for example, algorithmically sequenced); its only requirement is that there are no others identical to it. The simplest unique identifier is a counter. Each value is assigned and never repeats itself. The other is a static value in concert with a counter, for example a device manufacturer ID plus a product line ID plus a counter. In many cases, a random value is used in concert with static and counter fields. Non-repetition is generally not enough from the manufacturer's perspective. Usually, something needs a name that provides some context. To this end, manufacturer-unique fields may be added in a variety of ways unique to the manufacturer or in conformance with an industry convention. Uniqueness may also be fulfilled by using a globally **unique identifier** (**UUID**) for which the UUID standard specified in RFC 4122 applies.

No matter the mechanism, so long as a device is able to be provisioned with an identifier that is non-repeating, unique to its manufacturer, use, application, or a hybrid of all the above, it should be acceptable for use in identity management. Beyond the mechanisms, the only warning is that the combination of all possible identifiers within a statically specified ID length cannot be exhausted prematurely if at all possible.

Once a method for assigning uniqueness to your IoT devices is established, the next step is to be able to logically identify the assets within their area of operation to support authentication and access control functions.

## Naming a device

Every time you access a restricted computing resource, your identity is checked to ensure that you are authorized to access that specific resource. There are many ways that this can occur, but the end result of a successful implementation is that someone who does not have the right credentials is not allowed access. Although the process sounds simple, there are a number of difficult challenges that must be overcome when discussing identity and access management for the numerous constrained devices that comprise the IoT.

One of the first challenges is related to identity itself. Although identity may seem straightforward to you—your name for example—that identity must be translated into a piece of information that the computing resource (or access management system) understands. That identity must also not be duplicated across the information domain. Many computer systems today rely on a username, where each username within a domain is distinct. The username could be something as simple as `<lastname_firstname_middleiniital>`.

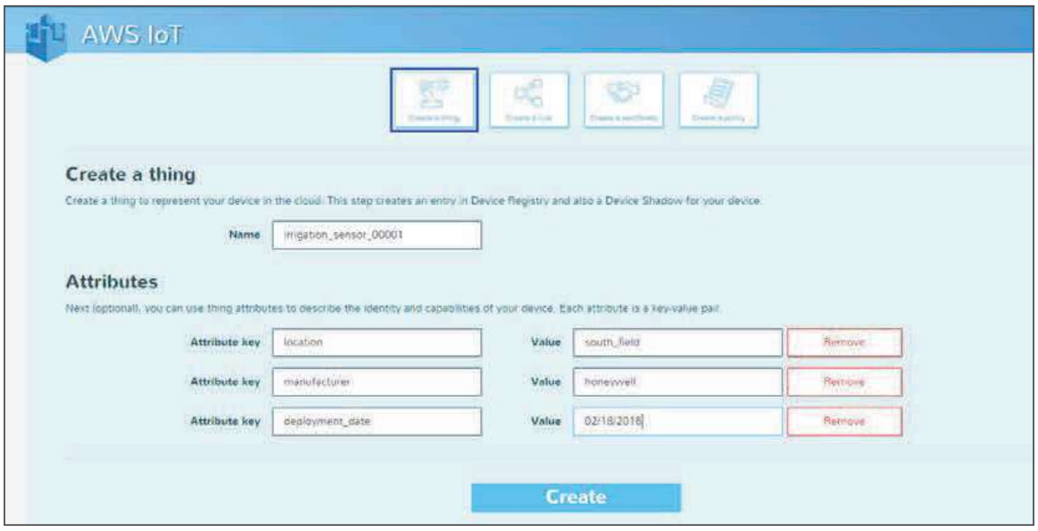
In the case of the IoT, understanding what identities—or names—to provision to a device can cause confusion. As discussed, in some systems devices use unique identifiers such as UUIDs or **electronic serial numbers** (ESNs).

We can see a good illustration by looking at how Amazon's first implementation of its IoT service makes use of IoT device serial numbers to identify devices. Amazon IoT includes a Thing Registry service that allows an administrator to register IoT devices, capturing for each the name of the thing and various attributes of the thing. The attributes can include data items such as:

- Manufacturer
- Type
- Serial number
- Deployment date
- Location

Note that such attributes can be used in what is called **attribute-based access control** (**ABAC**). ABAC access approaches allow access decision policies to be defined not just by the identity of the device, but also its properties (attributes). Rich, potentially complex rules can be defined for the needs at hand.

The following figure provides a view of the AWS IoT service:



Create a thing

Create a thing to represent your device in the cloud. This step creates an entry in Device Registry and also a Device Shadow for your device.

Name: irrigation_sensor_00001

Attributes

Next (optional), you can use thing attributes to describe the identity and capabilities of your device. Each attribute is a key-value pair.

| | | | | |
|---|---|---|---|---|
| Attribute key | location | Value | south_field | Remove |
| Attribute key | manufacturer | Value | honeywell | Remove |
| Attribute key | deployment_date | Value | 02/18/2018 | Remove |

Even when identifiers such as UUIDs or ESNs are available for an IoT device, these identifiers are generally not sufficient for securing authentication and access control decisions; an identifier can easily be spoofed without enhancement through cryptographic controls. In these instances, administrators must bind another type of identifier to a device. This binding can be as simple as associating a password with the identifier or, more appropriately, using credentials such as digital certificates.

IoT messaging protocols frequently include the ability to transmit a unique identifier. For example, MQTT includes a ClientID field that can transmit a broker-unique client identifier. In the case of MQTT, the ClientID is used to maintain state within a unique broker-client communication session.

## Secure bootstrap

Nothing is worse for security than an IoT-enabled system or network replete with false identities used in acts of identity theft, loss of private information, spoofing, and general mayhem. However, a difficult task in the identity lifecycle is to establish the initial trust in the device that allows that device to bootstrap itself into the system. Among the greatest vulnerabilities to secure identity and access management is insecure bootstrapping.

Bootstrapping represents the beginning of the process of provisioning a trusted identity for a device within a given system. Bootstrapping may begin in the manufacturing process (for example, in the foundry manufacturing a chip) and be complete once delivered to an end operator. It may also be completely performed in the hands of the end user or some intermediary (such as a depot or supplier), once delivered. The most secure bootstrapping methods start in the manufacturing processes and implement discrete security associations throughout the supply chain. They uniquely identify a device through:

- Unique serial number(s) imprinted on the device.
- Unique and unalterable identifiers stored and fused in device **read-only memory** (**ROM**).
- Manufacturer-specific cryptographic keys used only through specific lifecycle states to securely hand off the bootstrapping process to follow-on lifecycle states (such as shipping, distribution, hand off to an enrollment center, and so on). Such keys (frequently delivered out-of-band) are used for loading subsequent components by specific entities responsible for preparing the device.

PKIs are often used to aid in the bootstrapping process. Bootstrapping from a PKI perspective should generally involve the following processes:

- Devices are securely shipped from the manufacturer (via a secure, tamper detection capable shipping service) to a trusted facility or depot. The facility should have robust physical security access controls, record keeping, and audit processes, in addition to highly vetted staff.
- Devices counts and batches are matched against the shipping manifest.

Once received, the steps for each device include:

1. Authenticate uniquely to the device using a customer-specific, default manufacturer authenticator (password or key).
2. Install PKI trust anchors and any intermediate public key certificates (such as those of the registration authority, enrollment certificate authority, or other roots, and so on).
3. Install minimal network reachability information so that the device knows where to check certificate revocation lists, perform OCSP lookups, or other security-related functions.
4. Provision the device PKI credentials (public key signed by CA) and private key(s) so that other entities possessing the signing CA keys can trust the new device.

A secure bootstrapping process may not be identical to that described in the preceding list, but should be one that mitigates the following types of threats and vulnerabilities when provisioning devices:

- Insider threats designed to introduce new, rogue, or compromised devices (that should not be trusted)
- Duplication (cloning) of devices, no matter where in the lifecycle
- Introduction of public key trust anchors or other key material into a device that should NOT be trusted (rogue trust anchors and other keys)
- Compromise (including replication) of a new IoT device's private keys during key generation or import into the device
- Gaps in device possession during the supply chain and enrolment processes
- Protection of the device when re-keying and assigning new identification material needed for normal use (re-bootstrapping, as needed)

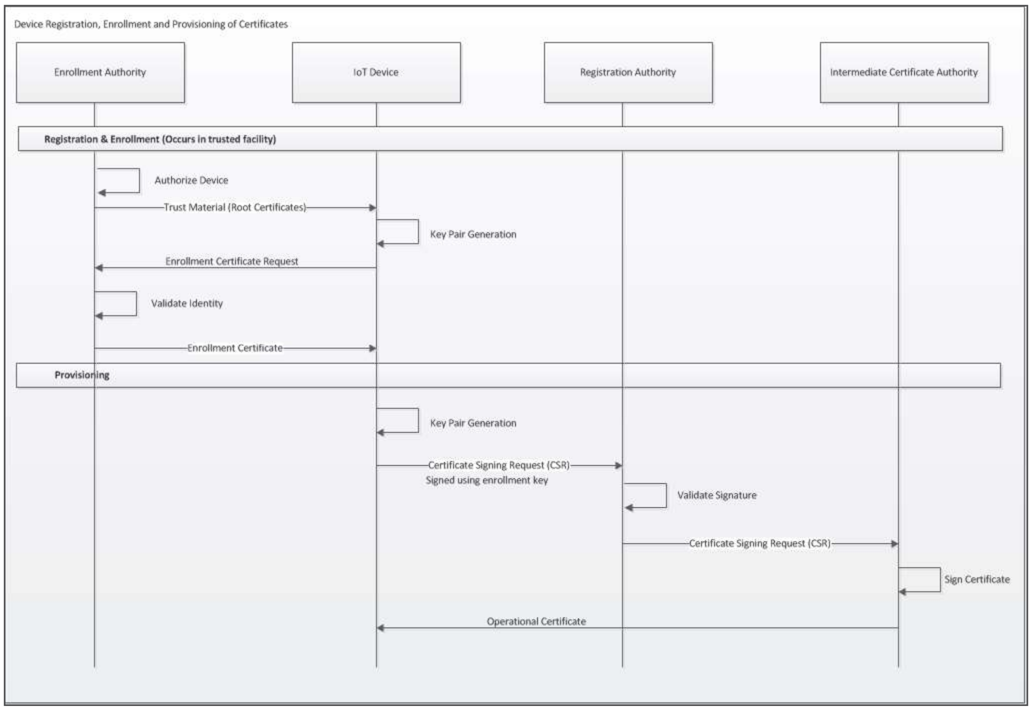
Given the security critical features of smart chip cards and their use in sensitive financial operations, the smart card industry adopted rigid enrollment process controls not unlike those described in the preceding list. Without them, severe attacks would have the potential to cripple the financial industry. Granted, many consumer-level IoT devices are unlikely to have secure bootstrapping processes, but over time we believe that this will change depending on the deployment environment and the stakeholders' appreciation of the threats. The more connected devices become, the greater their potential to do harm.

In practice, secure bootstrapping processes need to be tailored to the threat environment for the particular IoT device, its capabilities, and the network environment in question. The greater the potential risks, the more strict and thorough the bootstrapping process needs to be. The most secure processes will generally implement strong separation of duties and multi-person integrity processes during device bootstrap.

## Credential and attribute provisioning

Once the foundation for identities within the device is established, provisioning of operational credentials and attributes can occur. These are the credentials that will be used within an IoT system for secure communication, authentication, and integrity protection. We strongly recommend using certificates for authentication and authorization whenever possible. If using certificates, an important and security-relevant consideration is whether to generate the key pairs on the device itself, or centrally.

Some IoT services allow for central (such as by a key server) generation of public/private key pairs. While this can be an efficient method of bulk-provisioning thousands of devices with credentials, care should be taken to address potential vulnerabilities the process may expose (such as the sending of sensitive, private key material through intermediary devices/systems). If centralized generation is used, it should make use of a strongly secured key management system operated by vetted personnel in secured facilities. Another means of provisioning certificates is through the local generation of the key pairs (directly on the IoT device) followed by the transmission of the public key certificate through a certificate signing request to the PKI. Absent well-secured bootstrapping procedures, additional policy controls will have to be established for the PKI's **registration authority** (**RA**) in order to verify the identity of the device being provisioned. In general, the more secure the bootstrapping process, the more automated the provisioning can be. The following figure is a sequence diagram that depicts an overall registration, enrollment, and provisioning flow for an IoT device:


Device Registration, Enrollment and Provisioning of Certificates
Enrollment Authority
IoT Device
Registration Authority
Intermediate Certificate Authority
Registration & Enrollment (Occurs in trusted facility)
Authorize Device
Trust Material (Root Certificates)
Key Pair Generation
Enrollment Certificate Request
Validate Identity
Enrollment Certificate
Provisioning
Key Pair Generation
Certificate Signing Request (CSR)
Signed using enrollment key
Validate Signature
Certificate Signing Request (CSR)
Sign Certificate
Operational Certificate

## Local access

There are times when local access to the device is required for administration purposes. This may require the provisioning of SSH keys or administrative passwords. In the past, organizations frequently made the mistake of sharing administrative passwords to allow ease of access to devices. This is not a recommended approach, although implementing a federated access solution for administrators can be daunting. This is especially true when devices are spread across wide geographic distances such various sensors, gateways, and other unattended devices in the transportation industry.

## Account monitoring and control

After accounts and credentials have been provisioned, these accounts must continue to be monitored against defined security policies. It is also important that organizations monitor the strength of the credentials (that is, cryptographic ciphersuites and key lengths) provisioned to IoT devices across their infrastructure. It is highly likely that pockets of teams will provision IoT subsystems on their own, therefore defining, communicating, and monitoring the required security controls to apply to those systems is vital.

Another aspect of monitoring relates to tracking the use of accounts and credentials. Assign someone to audit local IoT device administrative credential use (passwords, and SSH keys) on a routine basis. Also seriously consider whether privileged account management tools can be applied to your IoT deployment. These tools allow for features such as checking out administrative passwords to aid in audit processes.

## Account updates

Credentials must be rotated on a regular basis; this is true for certificates and keys as well as passwords. Logistical impediments have historically hampered IT organizations' willingness to shorten certificate lifetimes and manage increasing numbers of credentials. There is a tradeoff to consider, as short-lived credentials have a reduced attack footprint, yet the process of changing the credentials tends to be expensive and time consuming. Whenever possible, look for automated solutions for these processes. Services such as Let's Encrypt (`https://letsencrypt.org/`) are gaining in popularity to help improve and simplify certificate management practices for organizations. Let's Encrypt provides PKI services along with an extremely easy-to-use plugin-based client that supports various platforms.

## Account suspension

Just as with user accounts, do not automatically delete IoT device accounts. Consider maintaining those accounts in a suspended state in case data tied to the accounts is required for forensic analysis at a later time.

## Account/credential deactivation/deletion

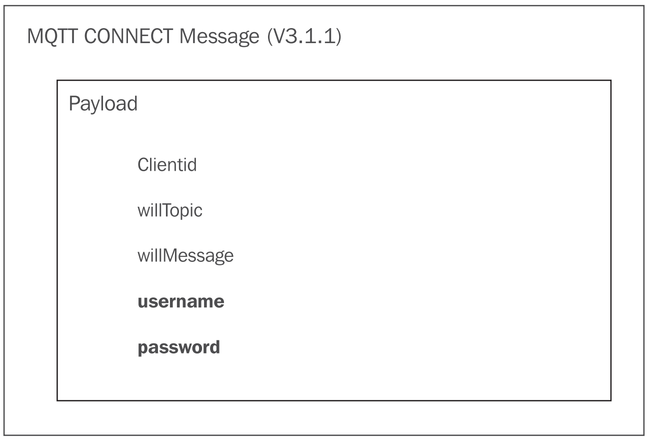
Deleting accounts used by IoT devices and the services they interact with will help combat the ability of an adversary to use those accounts to gain access after the devices have been decommissioned. Keys used for encryption (whether network or application) should also be deleted to keep adversaries from decrypting captured data at a later point in time using those recovered keys.

## Authentication credentials

IoT messaging protocols often support the ability to use different types of credentials for authentication with external services and other IoT devices. This section examines the typical options available for these functions.

### Passwords

Some protocols, such as MQTT, only provide the ability to use a username/password combination for native-protocol authentication purposes. Within MQTT, the CONNECT message includes the fields for passing this information to an MQTT Broker. In the MQTT Version 3.1.1 specification defined by OASIS, you can see these fields within the CONNECT message (reference: `http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html`):



Note that there are no protections applied to support the confidentiality of the username/password in transit by the MQTT protocol. Instead, implementers should consider using the **transport layer security** (**TLS**) protocol to provide cryptographic protections.

There are numerous security considerations related to using a username/password-based approach for IoT devices. Some of these concerns include:

- Difficulty in managing large numbers of device usernames and passwords
- Difficulty securing the passwords stored on the devices themselves
- Difficulty managing passwords throughout the device lifecycle

Though not ideal, if you do plan on implementing usernames/passwords for IoT device authentication, consider taking these precautions:

1. Create policies and procedures for rotating passwords at least every 30 days for each device. Better yet, implement a technical control wherein the management interface automatically prompts you when password rotation is needed.
2. Establish controls for monitoring device account activity.
3. Establish controls for privileged accounts that support administrative access to IoT devices.
4. Segregate the password-protected IoT devices onto less-trusted networks.

## Symmetric keys

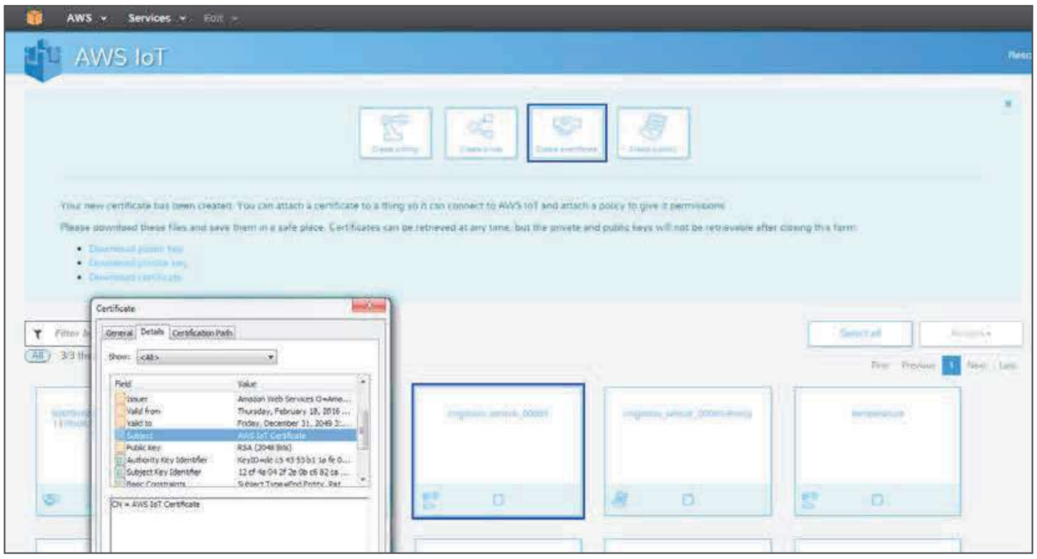
Symmetric key material may also be used to authenticate, as mentioned in *Chapter 5*, *Cryptographic Fundamentals for IoT Security Engineering*. **Message authentication codes** (**MACs**) are generated using a MAC algorithm (such as HMAC, CMAC, and so on) with a shared key and known data (signed by the key). On receiving side, an entity can prove the sender possessed the pre-shared key when the its computed MAC is shown to be identical to the received MAC. Unlike a password, symmetric keys do not require the key to be sent between the parties (except ahead of time or agreed on using a key establishment protocol) at the time of the authentication event. The keys will either need to be established using a public key algorithm, input out of band, or sent to the devices ahead of time, encrypted using **key encryption keys** (**KEK**).

## Certificates

Digital certificates, public key-based, are a preferred method for providing authentication functionality in the IoT. Although some implementations today may not support the processing capabilities needed to use certificates, Moore's law for computational power and storage is fast changing this.

### X.509

Certificates come with a highly organized hierarchical naming structure that consists of organization, organizational unit(s), and **distinguished names** (**DN**) or **common names** (**CN**). Referencing AWS support for provisioning X.509 certificates, we can see that AWS allows for the one-click generation of a device certificate. In the following example, we generate a device certificate with a generic IoT Device common name and a lifetime of 33 years. The one-click generation also (centrally) creates the public/private key pair. If possible, it is recommended that you generate your certificates locally by 1) generating a key pair on the device and 2) uploading a CSR to the AWS IoT service. This allows for customized tailoring of the certificate policy to define the hierarchical units (OU, DN, and so on) that are useful for additional authorization processes:



Your new certificate has been created. You can attach a certificate to a thing so it can connect to AWS IoT and attach a policy to give it permissions.

Please download these files and save them in a safe place. Certificates can be retrieved at any time, but the private and public keys will not be retrievable after closing this form.

- Download public key
- Download private key
- Download certificate

| Field | Value |
|---|---|
| Issuer | Amazon Web Services O=Ama... |
| Valid from | Thursday, February 18, 2016 ... |
| Valid to | Friday, December 31, 2049 2:... |
| Subject | AWS IoT Certificate |
| Public key | RSA (2048 Bits) |
| Authority Key Identifier | KeyID=de c5 43 53 b1 1e fe 0... |
| Subject Key Identifier | 12 cf 4e 04 2f 2e 0b c6 82 ca ... |
| Basic Constraints | Subject Type=End Entity, Pat |

CN = AWS IoT Certificate

## IEEE 1609.2

The IoT is characterized by many use cases involving machine-to-machine communication and some of them involve communications through a congested wireless spectrum. Take connected vehicles, for instance, an emerging technology wherein your vehicle will possess **on-board equipment** (**OBE**) that can automatically alert other drivers in your vicinity to your car's location in the form of **basic safety messages** (**BSM**). The automotive industry, **US Dept. of Transportation** (**USDOT**), and academia have been developing CV technology for many years and it will make its commercial debut in the 2017 Cadillac. In a few years, it is likely that most new US vehicles will be outfitted with the technology. It will not only enable vehicle-to-vehicle communications, but also **vehicle-to-infrastructure** (**V2I**) communications to various roadside and backhaul applications. The **dedicated short range communications** (**DSRC**) wireless protocol (based on IEEE 802.11p) is limited to a narrow set of channels in the 5 GHz frequency band. To accommodate so many vehicles and maintain security, it was necessary to 1) secure the communications using cryptography (to reduce malicious spoofing or eavesdropping attacks) and 2) minimize the security overhead within connected vehicle BSM transmissions. The industry resolved to use a new, slimmer and sleeker digital certificate design, the IEEE 1609.2.

The 1609.2 certificate format is advantageous in that it is approximately half the size of a typical X.509 certificate while still using strong, elliptic curve cryptographic algorithms (ECDSA and ECDH). The certificate is also useful for general machine-to-machine communication through its unique attributes, including explicit application identifier (SSID) and credential holder permission (SSP) fields. These attributes can allow IoT applications to make explicit access control decisions without having to internally or externally query for the credential holder's permissions. They're embedded right in the certificate during the secure, integrated bootstrapping and enrollment process with the PKI. The reduced size of these credentials also makes them attractive for other, bandwidth-constrained wireless protocols.

## Biometrics

There is work being done in the industry today on new approaches that leverage biometrics for device authentication. The FIDO alliance (`www.fidoalliance.org`) has developed specifications that define the use of biometrics for both a password-less experience and for use as a second authentication factor. Authentication can include a range of flexible biometric types—from fingerprints to voice prints. Biometric authentication is being added to some commercial IoT devices (such as consumer door locks) already, and there is interesting potential in leveraging biometrics as a second factor of authentication for IoT systems.

For example, voice prints could be used to enable authentication across a set of distributed IoT devices such as **road side equipment** (**RSE**) in the transportation sector. This would allow an RSE tech to access the device through a cloud connection to the backend authentication server. Companies such as Hypr Biometric Security (`https://www.hypr.com/`) are leading the way towards using this technology to reduce the need for passwords and enable more robust authentication techniques.

## New work in authorization for the IoT

Progress toward using tokens with resource-constrained IoT devices has not fully matured; however, there are organizations working on defining the use of protocols such as OAuth 2.0 for the IoT. One such group is the **Internet Engineering Task Force** (**IETF**) through the **Authentication and Authorization for Constrained Environments** (**ACE**) effort. ACE has specified RFC 7744 *Use Cases for Authentication and Authorization in Constrained Environments* (reference: `https://datatracker.ietf.org/doc/rfc7744/`). The RFC use cases are primarily based on IoT devices that employ CoAP as the messaging protocol. The document provides a useful set of use cases that clarify the need for a comprehensive IoT authentication and authorization strategy. RFC 7744 provides valuable considerations for authentication and authorization of IoT devices, including:

- Devices may host several resources wherein each requires its own access control policy.
- A single device may have different access rights for different requesting entities.
- Policy decision points must be able to evaluate the context of a transaction. This includes the potential for understanding that a transaction is occurring during an emergency situation.
- The ability to dynamically control authorization policies is critical to supporting the dynamic environment of the IoT.

## IoT IAM infrastructure

Now that we have addressed many of the enablers of identity and access management, it is important to elaborate how solutions are realized in infrastructure. This section is primarily devoted to **public key infrastructures** (**PKI**) and their utility in securing IAM deployments for the IoT.

## 802.1x

802.1x authentication mechanisms can be employed to limit IP-based IoT device access to a network. Note though that not all IoT devices rely on the provisioning of an IP address. While it cannot accommodate all IoT device types, implementing 802.1x is a component of a good access control strategy able to address many use cases.

Enabling 802.1x authentication requires an access device and an authentication server. The access device is typically an access point and the authentication server can take the form of a RADIUS or some **authentication, authorization, and accounting** (**AAA**) server.

## PKI for the IoT

*Chapter 5*, *Cryptographic Fundamentals for IoT Security Engineering*, provided a technical grounding of topics related to cryptographic key management. PKIs are nothing more than instances of key management systems that have been engineered and standardized exclusively to provision asymmetric (public key) key material in the form of digital credentials, most commonly X.509 certificates. PKIs may be isolated to individual organizations, they may be public, Internet-based services, or they may be government-operated. When needing to assert an identity, a digital certificate is issued to a person or device to perform a variety of cryptographic functions, such as signing messages in an application or signing data as part of an authenticated key exchange protocol such as TLS.

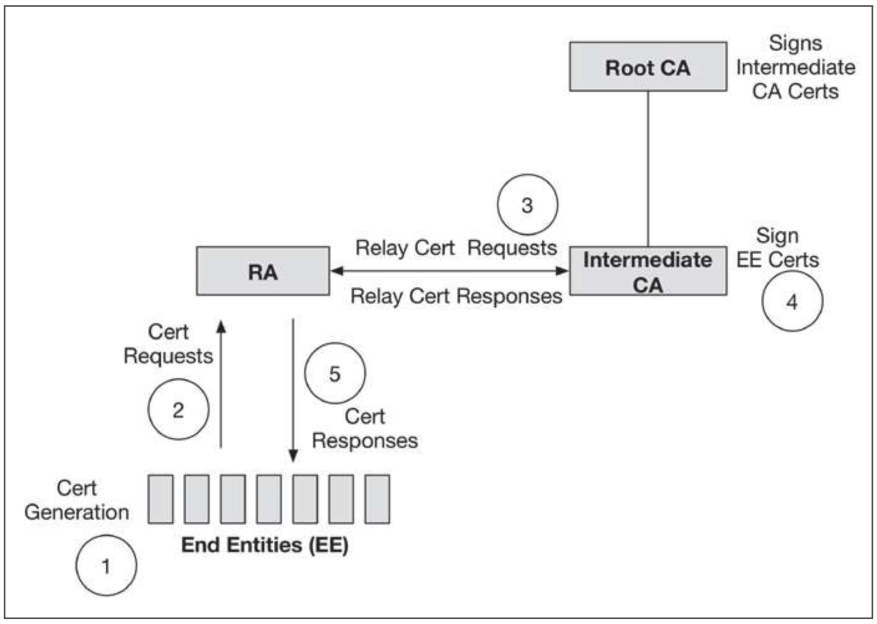
There are different workflows used in generating the public and private key pair (the public needing to be integrated into the certificate), but as we mentioned earlier, they generally fall into two basic categories: 1) self-generated or 2) centrally generated. When self-generated, the end IoT device requiring the digital certificate performs a key pair generation function, for example as described in FIPS PUB 180-4. Depending on the cryptographic library and invoked API, the public key may be raw and not yet put into a credential data structure such as X.509 or it may be output in the form of an unsigned certificate. Once the unsigned certificate exists, it is time to invoke the PKI in the form of a **certificate signing request** (**CSR**). The device sends this message to the PKI, then the PKI signs the certificate and sends it back for the device to use operationally.

## PKI primer

Public key infrastructures are designed to provision public key certificates to devices and applications. PKIs provide verifiable roots of trust in our Internet-connected world and can conform to a wide variety of architectures. Some PKIs may have very deep trust chains, with many levels between an **end entity** (such as an IoT device) and the top-most level root of trust (the root certificate authority). Others may have shallow trust chains in which there is only the one CA at the top and a single level of end entity devices underneath it. But how do they work?

Supposing an IoT device needs a cryptographically strong identity, it wouldn't make sense for it to provision itself with that identity because there is nothing inherently trustworthy about the device. This is where a trusted third party, the PKI certificate authority, comes into play and can vouch for the identity and in some cases the trust level of the device. Most PKIs do not allow end entities to directly interact with the CA, the entity responsible for cryptographically signing end entity certificates; instead they employ another subservient PKI node called a **registration authority** (**RA**). The RA receives certificate requests (typically containing the device's self-generated, but unsigned, public key) from end entities, verifies that they've met some minimum criteria, then passes the certificate request to the certificate authority. The CA signs the certificate (typically using RSA, DSA, or ECDSA signature algorithms), sending it back to the RA and finally the end entity in a message called the **certificate response**. In the certificate response message, the original certificate generated by the end entity (or some other intermediary key management system) is fully complete with the CA's signature and explicit identity. Now, when the IoT device presents its certificate during authentication-related functions, other devices can trust it because they 1) receive a valid, signed certificate from it, and 2) can validate the signature of the CA using the CA's public key trust anchor that they also trust (securely stored in their internal trust store).

The following diagram represents a typical PKI architecture:



In the preceding diagram, each of the **End Entities** (**EE**) can trust the others if they have the certificate authority keys that provide the chain of trust to them.

End entities that possess certificates signed by different PKIs can also trust each other. There are a couple of ways they can do this:

- **Explicit trust**: Each supports a policy that dictates that it can trust the other. In this case, end entities only need to have a copy of the trust anchor from the other entity's PKI to trust it. They do this by performing certificate path validation to those pre-installed roots. Policies can dictate the quality of the trust chain that is acceptable to rely on during certificate path validation. Most trust on the Internet today works like this. For example, web browsers explicitly trust so many web servers on the Internet merely because the browser comes pre-installed with a copy of the most common Internet root CA trust anchors.

- **Cross-certification**: When a PKI needs stricter cohesion in the policies, security practices, and interoperability of their domain with other PKIs, they can either directly cross-sign (each becomes an issuer for the other) or create a new structure called a PKI bridge to implement and allocate policy interoperability. The US Federal government's Federal PKI is an excellent example of this. In some cases, a PKI bridge needs to be created to provide a transition time between old certificates' cryptographic algorithms and new ones (for example, the Federal PKI's SHA1 bridge for accommodating older SHA1 cryptographic digests in digital signatures).

In terms of the IoT, many Internet-based PKIs exist today that can provision certificates to IoT devices. Some organizations operate their own on the fly. To become a formally recognized PKI on the Internet can be a significant endeavor. A PKI will require significant security protections and need to meet strict assurance requirements as implemented in various PKI assurance schemes (such as WebTrust). In many cases, organizations obtain service contracts with PKI providers that operate certificate authorities as a service.

## Trust stores

We diverge momentarily from infrastructure to discuss where the PKI-provisioned credentials end up being stored in devices. They are frequently stored in internal trust stores. Trust stores are an essential IoT capability with regard to the protection of digital credentials. From a PKI perspective, a device's trust store is a physical or logical part of the IoT device that securely stores public and private keys, often (and better when) encrypted. Within it, both the device's private/public keys and its PKI roots of trust are stored. Trust stores tend to be strongly access-controlled sections of memory, often only accessible from OS kernel-level processes, to prevent unauthorized modification or substitution of public keys or reading/copying of private keys. Trust stores can be implemented in hardware, as in small **hardware security modules** (**HSM**) or other dedicated, secure processors. They can also be implemented solely in software (such as with many instances of Windows and other desktop operating systems). In many desktop-type deployments, credentials can be maintained within **trusted platform modules** (**TPMs**), dedicated chips integrated into a computer's motherboard, though TPMs have not made a large penetration of the IoT market as of yet. Other enterprise-focused mobile solutions exist for secure storage of sensitive security parameters. For example, Samsung Knox provides mobile device secure storage through its Knox workspace container (secure hardware root of trust, secure boot, and other sensitive operational parameters).

IoT devices can depend on PKIs in different ways or not at all. For example, if the device uses only a self-signed credential and is not vouched for by a PKI, it still should securely store the self-signed credential in its trust store. Likewise, if the device has externally provisioned an identity from a PKI, it must maintain and store critical keys pertinent to that PKI and any other PKI that it inherently or indirectly trusts. This is accomplished through the storing of certificate authority public key trust anchors and often the intermediate certificates as well. When deciding to trust an external entity, the entity will present the IoT device with a certificate signed by a certificate authority. In some cases (and in some protocols), the entity will provide the CA certificate or a complete trust chain, along with its own certificate so that it can be validated to a root.

Whether or not an IoT device directly supports PKI, if it uses public key certificates to validate another device's authenticity or presents its own certificates and trust chains, it should do so using digital credentials and trust anchors securely stored in its trust store. Otherwise, it will not be protected from access by malicious processes and hackers.

## PKI architecture for privacy

Privacy has many facets and is frequently not a concept directly associated with PKIs. PKIs, by design, are there to provide trusted identities to individuals and devices. When initiating electronic transactions, one usually wants to specifically identify and authenticate the other party before initiating sensitive transactions with them.

Anonymity and the general ability to operate in networks and RF environments without being tracked, however, are becoming increasingly important. For instance, suppose a system needs to provision anonymous trusted credentials to a device so that other entities have the ability to trust it without explicitly knowing its identity. Consider further that the PKI design itself needs to limit insider threats (PKI operators) from being able to associate certificates and the entities to which they are provisioned.

The best example of this is reflected in the emerging trend of anonymous PKIs, one of the best known being the forthcoming **security credential management system** (**SCMS**) designed for the automotive industry's connected vehicles initiative. The SCMS provides a fascinating look at the future of privacy-protected IoT trust. The SCMS, now in a Proof of Concept phase, was specifically engineered to eliminate the ability of any single node of the PKI from being able to ascertain and associate SCMS credentials (IEEE 1609.2 format) with the vehicles and vehicle operators to which they are provisioned.

1609.2 certificates are used by OBE, embedded devices in the automobile to send out BSM to surrounding vehicles to enable the vehicles to provide drivers with preemptive safety messages. In addition to vehicle use, 1609.2 credentials will be used by networked and standalone **roadside units** (**RSU**) mounted near traffic signal controllers to provide various roadside applications. Many of the connected vehicle applications requiring enhanced privacy protections are safety-focused, but many are also designed to improve traffic system and mobility performance, environmental emissions reduction, and others.

Given the versatility of so many IoT application use cases, the most sensitive privacy-impacting IoT devices (for example, medical devices) may increasingly begin to make use of no-backdoor, privacy-protecting PKIs, especially when civil liberties concerns are obvious.

## Revocation support

When authenticating in a system using PKI credentials, devices need to know when other devices' credentials are no longer valid, aside from expiration. PKIs routinely revoke credentials for one reason or another, sometimes from detection of compromise and rogue activity; in other cases, it's simply that a device has malfunctioned or otherwise been retired. No matter the reason, a revoked device should no longer be trusted in any application or network layer engagement.

The conventional method of doing this is for CAs to periodically generate and issue **certificate revocation lists** (**CRL**), a cryptographically signed document listing all revoked certificates. This requires that that end devices have the ability to reach out through the network and frequently refresh CRLs. It also requires turnaround time for 1) the CA to generate and publish the CRL, 2) end devices to become aware of the update, and 3) end devices to download it. During this interval of time, untrusted devices may yet be trusted by the wider community.

### OCSP

Given the potential latency and the need to download large files, other mechanisms have evolved to more quickly provide revocation information over networks, most notably the **online certificate status protocol** (**OCSP**). OCSP is a simple client/server protocol which allows clients to simply ask a server whether a given public key credential is still valid. The OCSP server is typically responsible for the CA's **Certificate Revocation List** (**CRL**) and using it to generate an OCSP proof set (internally signed database of proofs). These sets are then used to generate OCSP response messages to the requesting clients. OCSP proof sets can be generated periodically for different time intervals.

### OCSP stapling

OCSP stapling resolves some of the challenges of having to perform the latency-inducing, secondary client-server OCSP call just to obtain revocation information. OCSP stapling simply provides a pre-generated OCSP response message, in conjunction with the server's certificate (such as during a TLS handshake). This way, clients can verify the digital signature on the pre-generated OCSP response (no additional handshakes necessary) and make sure the CA still vouches for the server.

### SSL pinning

This technique may apply more to IoT device developers that require their devices to communicate with an Internet service (for example, for passing usage data or other information). In order to protect from the potential compromise of the trust infrastructure that provisions certificates, developers can pin the trusted server certificate directly into the IoT device trust store. The device can then check the server certificate explicitly against the certificate in the trust store when connecting to the server. In essence, SSL pinning doesn't place full trust in the certificate's trust chain; it only trusts the server if the received server certificate is identical to the pinned (stored) certificate and the signature is valid. SSL pinning can be used in a variety of interfaces, from web server communications to device management.

## Authorization and access control

Once a device is identified and authenticated, determining what that device can read or write to other devices and services is required. In some cases, being a member of a particular **community of interest** (**COI**) is sufficient, however in many instances there are restrictions that must be put in place even upon members of a COI.

## OAuth 2.0

To refresh, OAuth 2.0 is a token-based authorization framework specified in IETF RFC 6749, which allows a client to access protected, distributed resources (that is, from different websites and organizations) without having to enter passwords for each. As such, it was created to address the frequently cited, sad state of password hygiene on the Internet. Many implementations of OAuth 2.0 exist, supporting a variety of programming languages to suit. Google, Facebook, and many other large tech companies make extensive use of this protocol.

The IETF ACE Working Group has created working papers that define the application of OAuth 2.0 to the IoT. The draft document may be promoted to an RFC in the future. The document is designed primarily for CoAP and includes as a core component a binary encoding scheme known as **concise binary object representation** (**CBOR**) that can be used within IoT devices when JSON is not sufficiently compact.

Proposed extensions to OAuth 2.0 have also been discussed, for example, extending the messaging between an AS and a client to determine how to connect securely with a resource. This is required given that the use of TLS is expected with typical OAuth 2.0 transactions. With constrained IoT devices that employ CoAP, this is not a valid assumption.

The constrained device-tailored version of OAuth 2.0 also introduces a new authorization information format. This allows for access rights to be specified as a list of **uniform resource indicators** (**URIs**) of resources mapped with allowed actions (for example, GET, POST, PUT, and DELETE). This is a promising development for the IoT.

From a security implementation perspective, it's important to step back and keep in mind that OAuth is a security framework. Security frameworks can be something of an oxymoron; the more flexible and less specific the framework is regarding implementation, the wider the latitude to create insecure products. It's a tradeoff we frequently encounter in the world of public standards, where the goals of a new security standard somehow have to be met while satisfying the interests of many stakeholders. Typically, both interoperability and security suffer as a result.

With that in mind, we identify just a few of the many security best practices regarding OAuth2. We encourage readers to visit IETF RFC 6819 for a more thorough treatment of OAuth2 security considerations (`https://tools.ietf.org/html/rfc6819#section-4.1.1`):

- Use TLS for authorization server, client, and resource server interactions. Do NOT send client credentials over an unprotected channel.
- Lock down your authorization server database and the network in which it resides.
- Use high entropy sources when generating secrets.
- Securely store your client credentials: `client_id` and `client_secret`. These parameters are used to identify and authenticate your client application to the API when requesting user account access. Unfortunately, some implementations hard-code these values or distribute them over less protected channels, making them attractive targets for attackers.

- Make use of the OAuth2 state parameter. This will allow you to link the authorization requests with redirect URIs needed for delivery of the access token.
- Don't follow untrusted URLs.
- If in doubt, lean toward shorter expiry times for authorization codes and tokens.
- Servers should revoke all tokens for an authorization code that someone is repeatedly attempting to redeem.

Future IoT implementations that make use of OAuth 2.0 and similar standards greatly need secure by default implementations (library APIs) to reduce developers' exposure to making critical security errors.

## Authorization and access controls within publish/subscribe protocols

The MQTT protocol provides a good exemplar for understanding the need for finer-grained access controls. As a publish/subscribe protocol, MQTT allows clients to write and read topics. Not all clients will have permissions to write all topics. Not all clients will have permissions to read all topics either. Indeed, controls must be put in place that restrict the permissions of clients at the topic level.

This can be achieved in a MQTT broker by keeping an access control list that pairs topics with authorized publishers and authorized subscribers. The access controls can take as input the client ID of the MQTT client, or depending on the broker implementation, the username that is transmitted in the MQTT connect message. The broker performs a topic lookup when applicable MQTT messages arrive to determine if the clients are authorized to read, write, or subscribe to topics.

Alternatively, since MQTT is often implemented to operate over TLS, it is possible to configure the MQTT broker to require certificate-based authentication of the MQTT client. The MQTT broker can then perform a mapping of information in the MQTT client X.509 certificate to determine the topics to which the client has permission to subscribe or publish.

## Access controls within communication protocols

There are different access control configurations that can be set in other communication protocols as well. For example, ZigBee includes the ability for each transceiver to manage an access control list to determine whether a neighbor is trusted or not. The ACL includes information such as the address of the neighbor node, the security policy in use by the node, the key, and the last **initialization vector** (**IV**) used.

Upon receiving a packet from a neighbor node, the receiver consults the ACL and if the neighbor is trusted, then the communication is allowed. If not, the communication is either denied or an authentication function is invoked.

## Summary

This chapter provided an introduction to identity and access management for IoT devices. The identity lifecycle was reviewed and a discussion on infrastructure components required for provisioning authentication credentials was provided, with a heavy focus on PKI. There was a look at different types of authentication credentials and a discussion on new approaches to providing authorization and access control for IoT devices was also provided.

In the next chapter, we visit the complex ecosystem in which IoT privacy concerns need to be addressed and mitigated. Security controls, such as effective identity and access management discussed in this chapter, represent only one element of the IoT privacy challenge.