AWS Whitepaper

Device Manufacturing and Provisioning with X.509 Certificates in AWS IoT Core

Device Manufacturing and Provisioning with X.509 Certificates in AWS IoT Core: AWS Whitepaper

Table of Contents

This whitepaper is for historical reference only. Some content might be outdated and some links might not be available.

Device Manufacturing and Provisioning with X.509 Certificates in AWS IoT Core

Publication date: **November 17, 2022** (*Document revisions*)

Abstract

This whitepaper focuses on onboarding Internet of Things (IoT) devices in AWS IoT Core using unique identities. It covers the different options, challenges, and considerations for manufacturing and provisioning unique X.509 certificates and private keys into devices for certificate-based mutual authentication.

The whitepaper provides device makers with guidance on the appropriate AWS IoT provisioning options, based on the capabilities of their device and manufacturing process. It is not intended to cover Sigv4 and Custom Authorizer authentication methods.

This whitepaper is intended for technical architects, IoT cloud engineers, IoT security architects, and embedded engineers. This whitepaper assumes that the reader understands fundamental Public Key Infrastructure (PKI) and Transport Layer Security (TLS) concepts and terminology.

Are you Well-Architected?

The AWS Well-Architected Framework helps you understand the pros and cons of the decisions you make when building systems in the cloud. The six pillars of the Framework allow you to learn architectural best practices for designing and operating reliable, secure, efficient, cost-effective, and sustainable systems. Using the AWS Well-Architected Tool, available at no charge in the AWS Management Console, you can review your workloads against these best practices by answering a set of questions for each pillar.

For more expert guidance and best practices for your cloud architecture—reference architecture deployments, diagrams, and whitepapers, refer to the AWS Architecture Center.

Introduction

During the different phases of IoT device development and manufacturing, the way that these unique identities are provisioned and onboarded to AWS IoT Core can differ. Device makers are faced with a number of considerations during the lifecycle of an IoT device, including:

- Using a customer-owned Certificate Authority (CA), a third-party CA, or an AWS IoT created CA
- Using a hardware security module, such as a secure element or trusted platform module (TPM)
- Cloud resources needed to support the device provisioning process
- Device-level logic to implement onboarding procedures

This whitepaper explains the complexities of the device manufacturing supply chain, and assists device makers with recommendations based on the capabilities of their device, limitations of their manufacturing process, and device onboarding requirements of a service operator.

Resources in AWS IoT Core

For a device to connect to and communicate with AWS IoT Core, AWS IoT Core requires an IoT Thing, Certificate, and IoT Policy.

- **IoT Thing** — AWS strongly recommends that a device is registered as a Thing in the Thing registry. A Thing is a cloud-based representation of a physical device that includes a unique name and static attributes.
- **X.509 Certificate** — Each Thing must have an attached X.509 certificate. The certificate should be unique to a single Thing. The X.509 certificate contains public information including the signing CA (source of trust), public key, and expiration date. The public key is part of an asymmetrical key pair which includes a private key that is only stored on the device to ensure secrecy.
- **IoT Policy** — An IoT Policy is a document that defines the actions that the device is authorized to perform. The IoT Policy must be attached to the X.509 Certificate. An IoT Policy can be shared among many devices with the use of policy variables.

Resources on the device

For the device to connect to AWS IoT Core using TLS-based mutual authentication, the device needs to be provisioned with the Amazon Trust Services Root Certificate Authority, an X.509 certificate, a private key, and in some cases, the signing CA certificate for the device's client certificate.

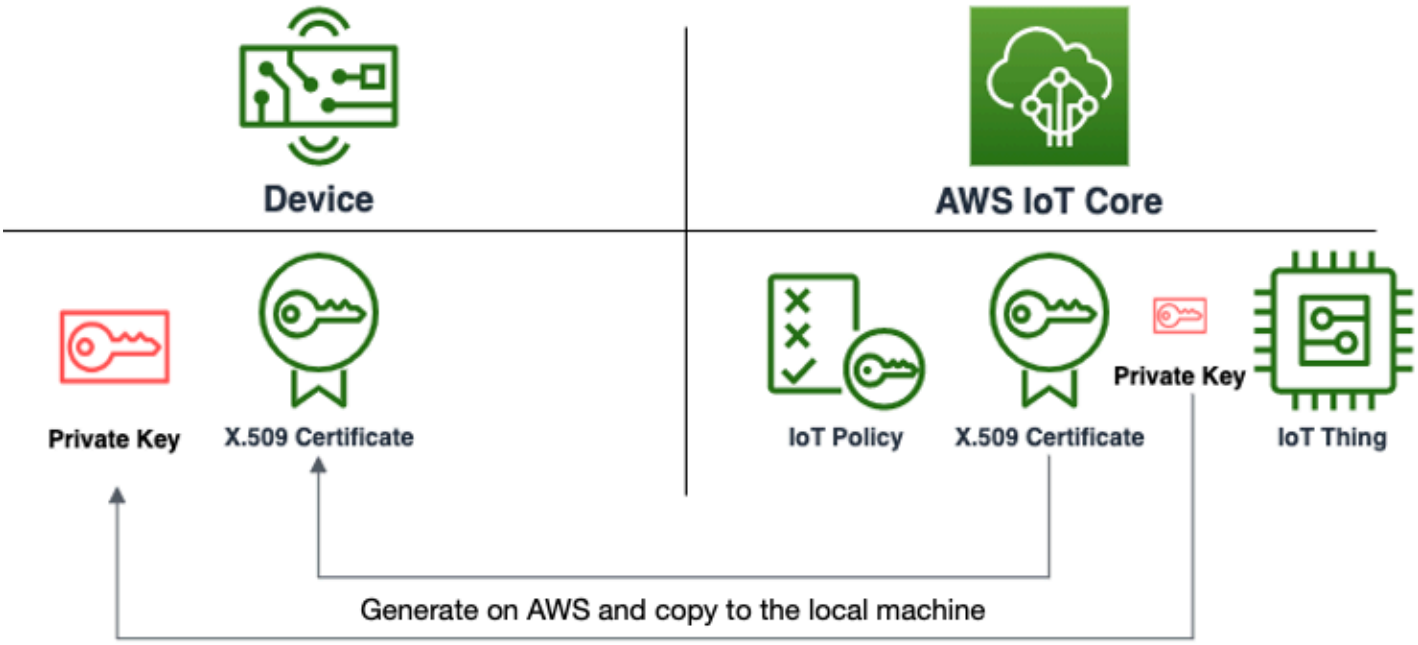
- **X.509 Certificate** — The same X.509 Certificate that is registered on AWS must also be present on the device. This certificate is presented during the TLS handshake with AWS IoT Core.
- **Private Key** — The private key of the device is asymmetrically paired with the public key that is presented with the X.509 certificate. The private key is ideally generated on the device using a True Random Number Generator and should never be exported from the device.
- **Signer Certificate Authority** — The device might need to send the X.509 certificate's issuing CA in the first TLS connection if the Just in Time device onboarding flow is used. Subsequent connections do not require the issuing CA certificate.
- **Service Root CA Certificate** — The Amazon Trust Services Root CA certificate is used by the device to verify that it is connecting to a genuine AWS IoT Amazon Trust Services Endpoint. The Root CA is used to validate the certificate chain presented by AWS IoT Core.

Device provisioning during development

In the early phases of an IoT project, a small number of devices are provisioned to AWS IoT for development and testing purposes. During this phase, developers often choose convenience over security and scalability.

For convenience, AWS IoT gives developers the ability to create a device certificate and keys in the cloud from AWS Management Console, AWS IoT Control Plane APIs, or the AWS Command Line Interface (AWS CLI). The certificate is issued by an ephemeral AWS intermediate CA, and the private key is generated on the AWS Cloud. The certificate, private key, and public key are then downloaded onto the developer's local machine.

The developer is responsible for manually provisioning the certificate and private key provided by AWS IoT Core to the device, where they are either added to the device's file system or stored in the device's firmware image.



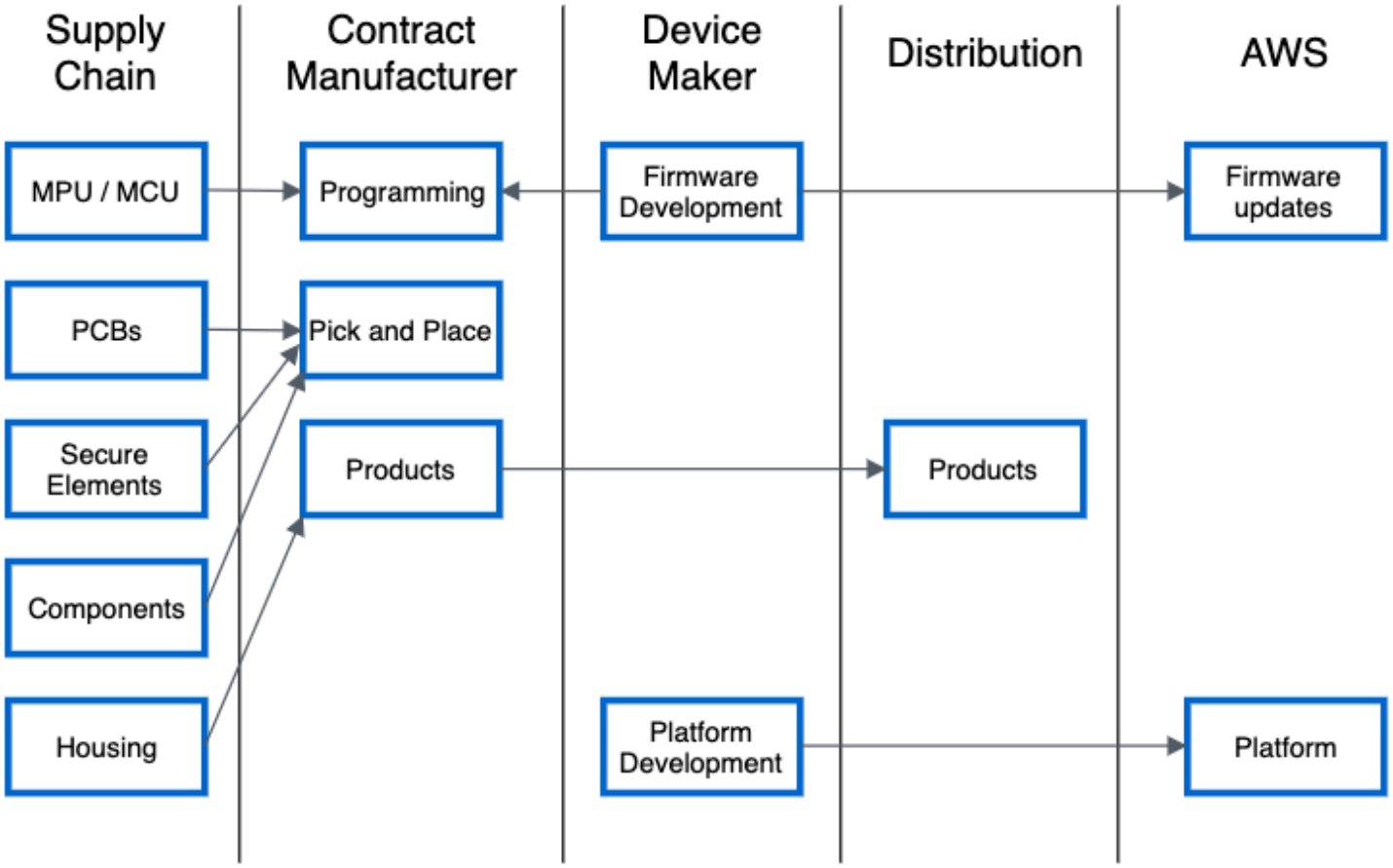
Process to provision devices during development using AWS IoT Core

This process should never be used in production because the private key is transmitted through an insecure environment such as the developer's laptop. If the private key is stored on the developer's local machine, even temporarily, it could be compromised by social engineering, user error, or a weak password. In production, the device private key should be generated on the device from a strong source of entropy and never exported.

Security risks are mitigated because the devices are not deployed in the field. The devices are tightly controlled by a developer or in a lab environment. A compromised device can be re-programmed and the certificate can be revoked.

The process of creating all resources in the AWS Cloud and copying the necessary keys to the device is manual and time consuming. Unique firmware and files need to be programmed into each device by the developer. For testing and development, most operations are performed in the lab environment, but the process is not scalable when entering the pilot or production phase of an IoT project. To scale an IoT project, there is typically a much more complex supply chain.

The device manufacturing supply chain



The IoT device manufacturing process

As an IoT project moves from a development phase to production, a supply chain is necessary between the device maker through to the customer.

Device makers specify the components, form factor, and functionality of a device. Device makers are also responsible for product design, developing hardware and software for products, developing and maintaining AWS Cloud applications, provisioning templates, policies, and resources, and the sales and marketing of the product. Most device makers do not have the ability to physically manufacture a large number of devices, and must outsource this manufacturing to a contract manufacturer.

During the prototyping stage, device makers might use high-mix, low-volume contract manufacturing to rapidly produce engineering samples in low volumes. When moving from the

prototyping to production phase, a high-volume contract manufacturer is used to take advantage of the economy of scale.

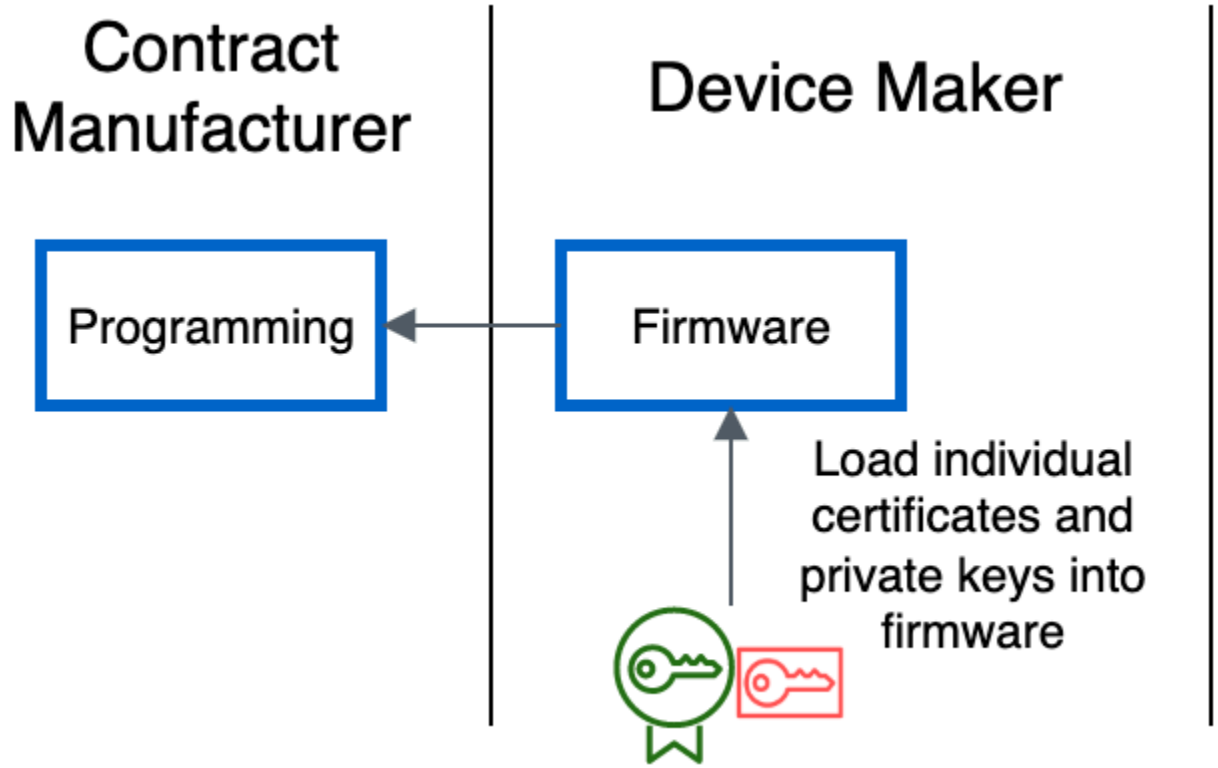
Low-mix, high-volume contract manufacturers have the production line, tooling, and processes in place to produce a large volume of devices. Contract manufacturers build devices to the specification provided by the device maker. This specification includes printed circuit board (PCB) schematics, a bill of materials, and the firmware that is loaded onto the device. Contract manufacturers place components onto PCBs (pick and place), program the processors with firmware and credentials provided by the device makers, and package the devices into a final product. Products are then provided to distribution channels to fulfill sales. Contract manufacturers must source the individual components to build the final product from their supply chain.

After the device is built, it can be sent to the device maker for direct distribution or engineering samples. The devices can also be sent directly to distributors or retailers who sell the device and fulfill the supply chain to end customers.

Provisioning device identity in the manufacturing supply chain

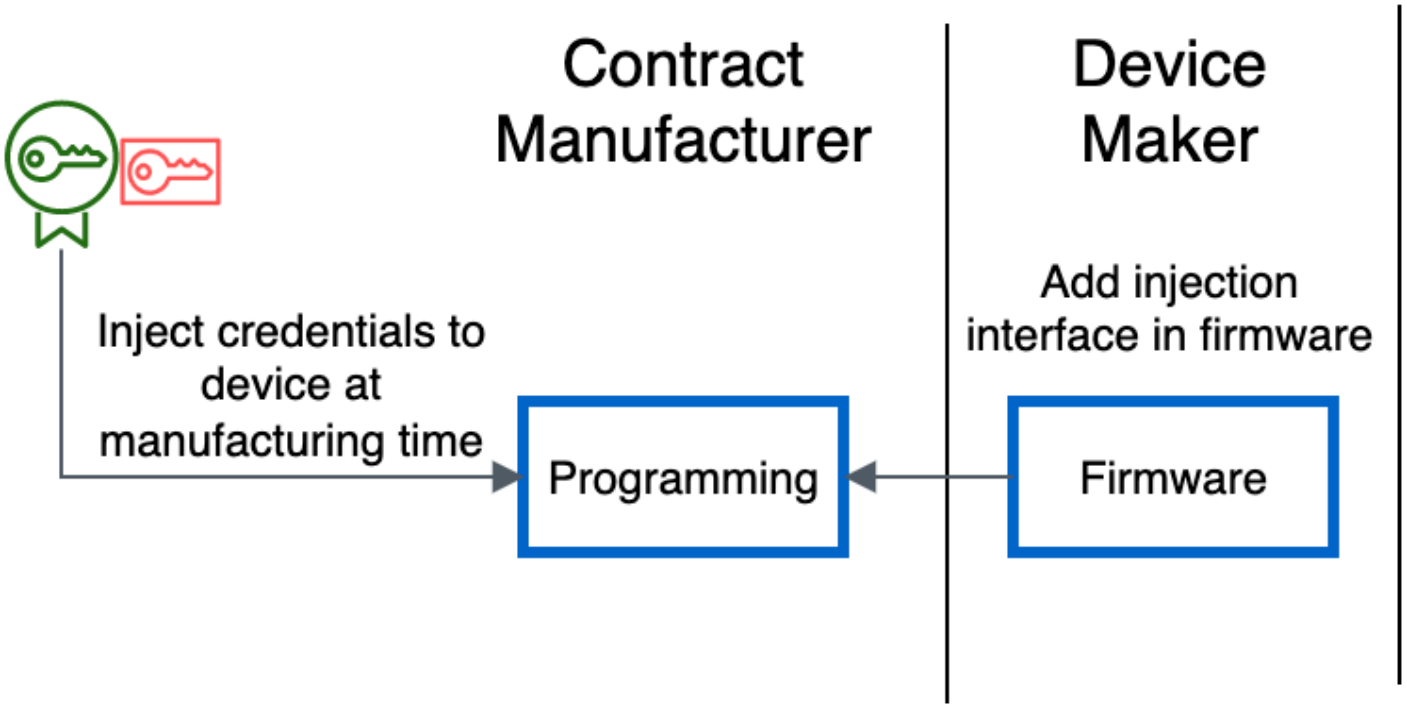
When creating the device manufacturing supply chain, device makers must determine when devices establish their private key and receive their unique X.509 certificate representing the cryptographic identity of the device.

Device makers may choose to provision devices with unique credentials in firmware. In this scenario, a unique firmware image is generated for each device that contains the X.509 certificate and private key. Each unique firmware image is then sent to a trusted contract manufacturer. Contract manufacturers must be able to program each individual firmware image onto the device's processor at manufacture time. In this scenario, the device maker maintains the most control over the supply chain and has the opportunity to pre-register each device on AWS IoT, but this adds considerable complexity to the contract manufacturing process. This process is typically suited for devices built in low volumes, where each device requires customization.



Provision credentials into firmware

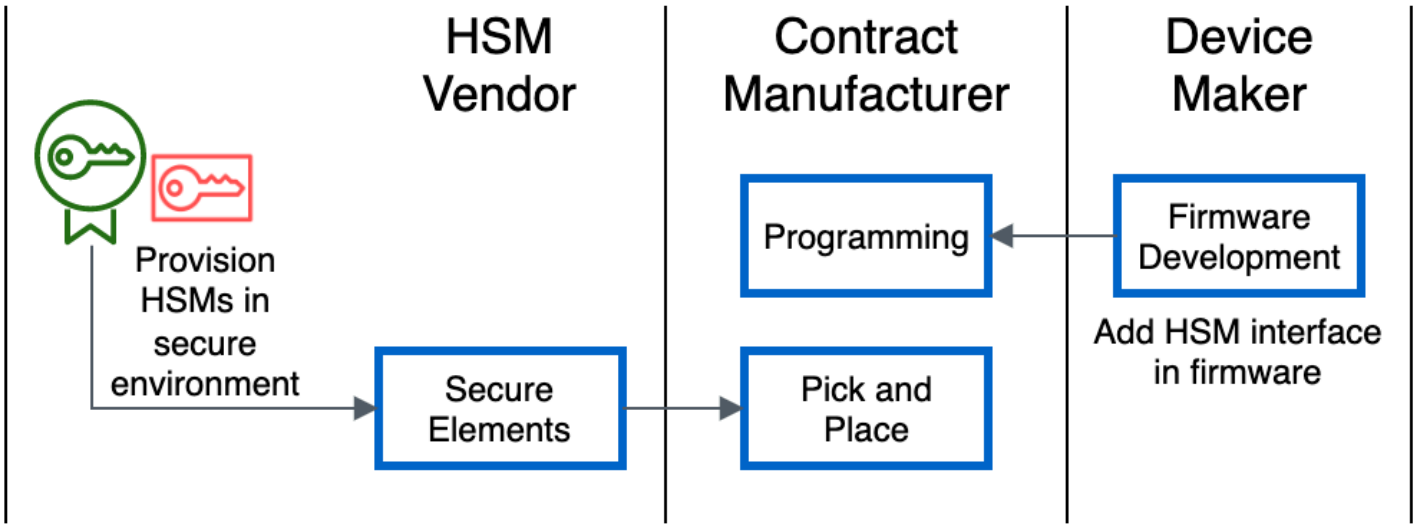
Customization can also be done by the contract manufacturer. Contract manufacturers may offer the ability to inject unique credentials during the manufacturing process. In this scenario, a single firmware image, known as a *golden image*, is provided to the contract manufacturer and the manufacturer is responsible for injecting unique credentials into each device. This can be performed during the device programming phase when the image is programmed before assembly of the PCB, or after assembly with an in-system programming interface prior to first boot of the device. Alternatively, the credential provisioning process can take place in a running system, but the golden image must then have additional logic so that the device can accept credentials over an open interface, such as Secure Shell (SSH), Network File System (NFS), or over a serial connection, and store those credentials to a secure place on the device. Security credentials and PKI may be handled and exposed to the contract manufacturer, so it's important that the provisioning process is performed in a secure environment by trusted individuals.



Inject credentials at manufacturing time

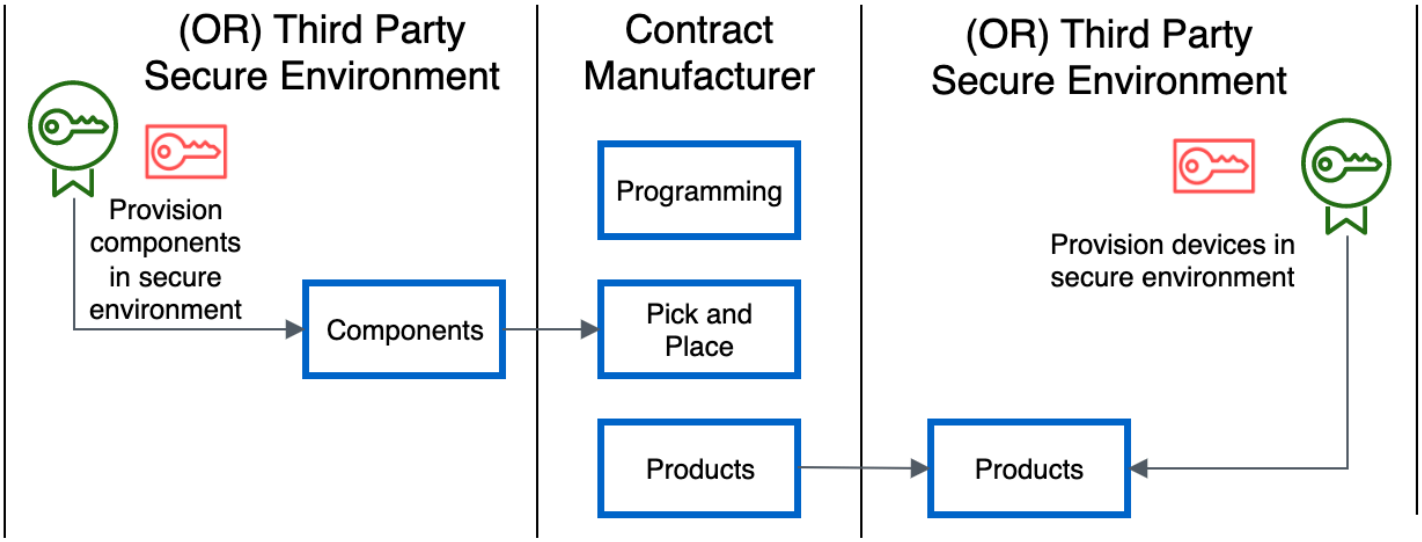
Introducing customization for each device image at manufacturing time can add valuable time to produce each device and additional logistical overhead, because the manufacturer must track that customization for each device produced. This can lead to increased cost per unit to the device maker charged by the contract manufacturer, due to the additional time using the manufacturer's production line.

To isolate and protect device keys from the firmware, device makers may choose to use a hardware security module (HSM), such as a secure element or trusted platform module (TPM) on their device. By using an HSM to separate the device keys from the firmware, the HSM can be provisioned with credentials independent of the firmware image, either during device manufacturing, or by the HSM vendor. HSM vendors offer services to generate private keys and sign X.509 certificates in the vendor's secure facility. This allows device makers to provide a single standard firmware image to the contract manufacturer with logic to communicate with the pre-provisioned HSM. The contract manufacturer is responsible for placing the HSM onto the PCB, but the credentials are not exposed and no extra processes are necessary to provision the credentials.



Hardware Security Module vendor provisions credentials to a secure element

Alternatively, device makers may use a trusted distributor or third party in the supply chain to program and provision device credentials to isolate this critical step from the end contract manufacturer producing the finished devices. Third parties and distributors offer value-added services such as secure programming of applications and credentials. This may happen at the component level (HSM, MCU, and so on) before a device is assembled by the end contract manufacturer, or after the finished devices have been produced in the supply chain to fulfill sales to end customers.



Third-party vendor provisions credentials in a secure environment

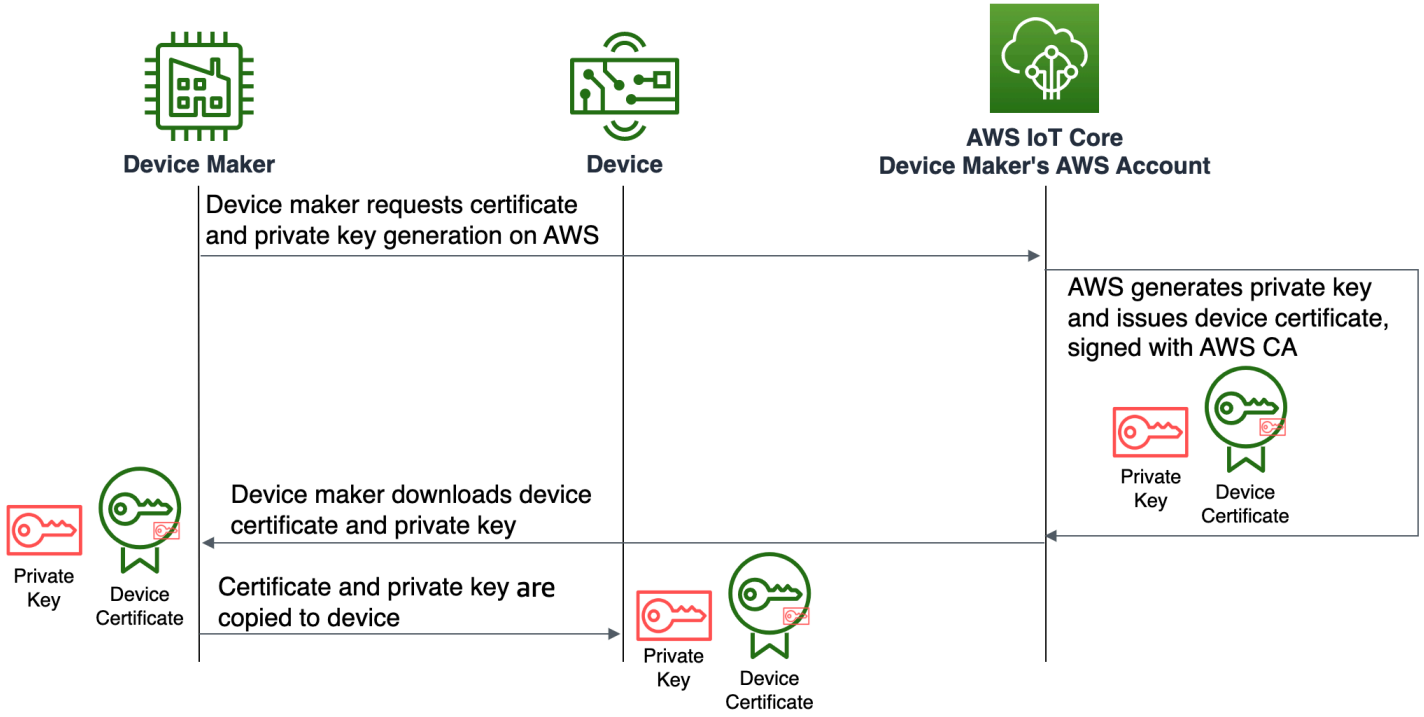
Devices not provisioned in the supply chain

Manufacturers, component vendors, or distributors might have minimum order quantities before they provide customization services. Introducing customization or a pre-provisioned hardware security module can be cost prohibitive for low-volume devices. In consumer products, sometimes it is not known where the device should be provisioned at the time of manufacturing. In these scenarios, devices come out of manufacturing with no unique credentials, and are provisioned once deployed in the field using an onboarding service such as AWS IoT fleet provisioning.

Managing the device public key infrastructure

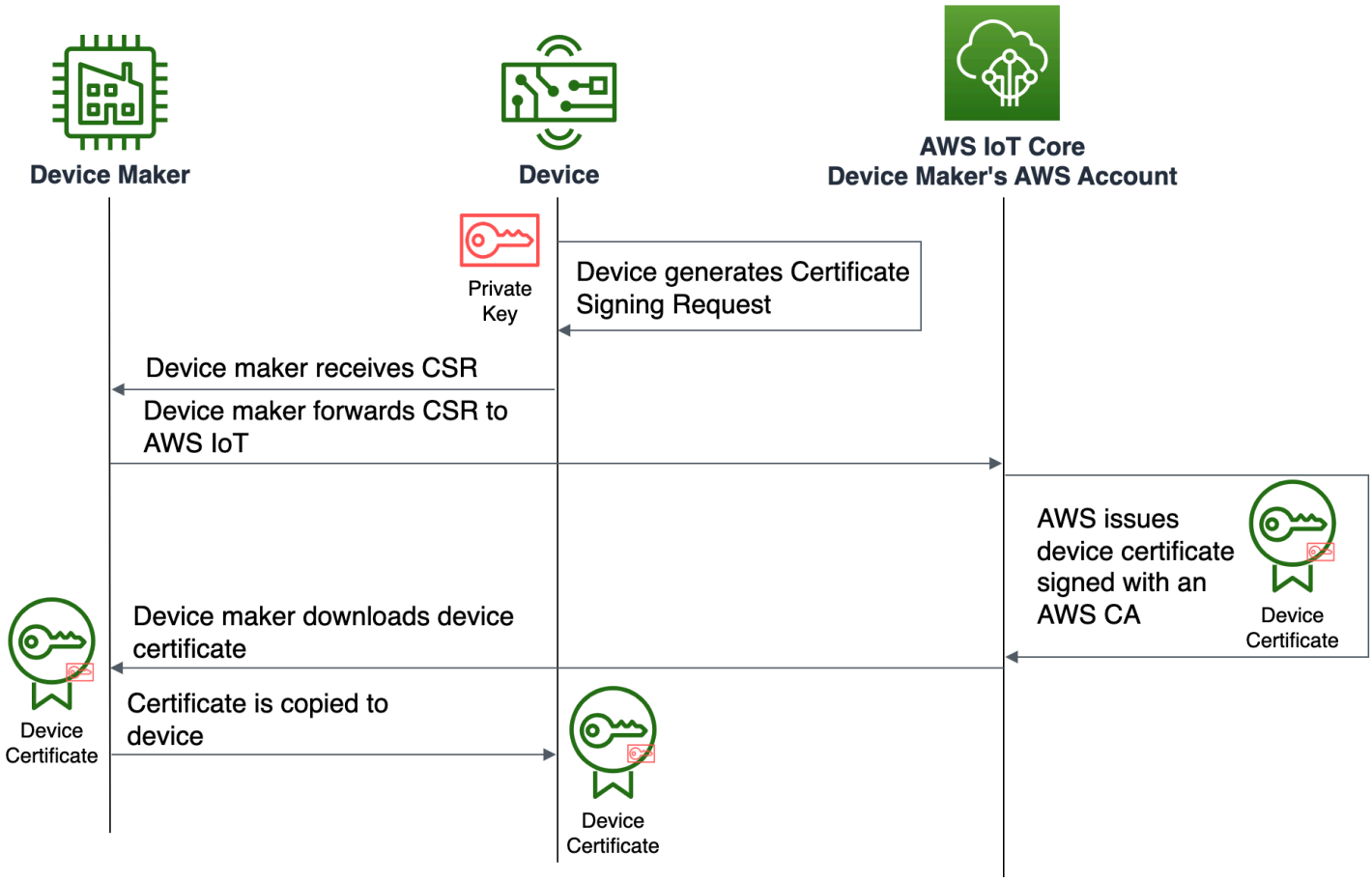
The device public key infrastructure (PKI) consists of Certificate Authorities (CAs) that issue and sign X.509 device certificates to establish a source of trust for a device. Device makers must decide whether they want to use AWS IoT to generate device certificates, their own CA, or a third-party CA.

AWS IoT provides APIs to generate large numbers of X.509 certificates and private keys in the cloud. The X.509 certificates are signed by an ephemeral AWS CA and are registered in the device maker's AWS IoT registry at creation. Once created, the device maker must download the certificate and private key and deliver them to the device during manufacturing.



X.509 certificate and private key generated on AWS

If the device already has a private key, a Certificate Signing Request can be sent to AWS to sign the certificate without exposing the private key on the device.



Certificate Signing Request made by device to AWS

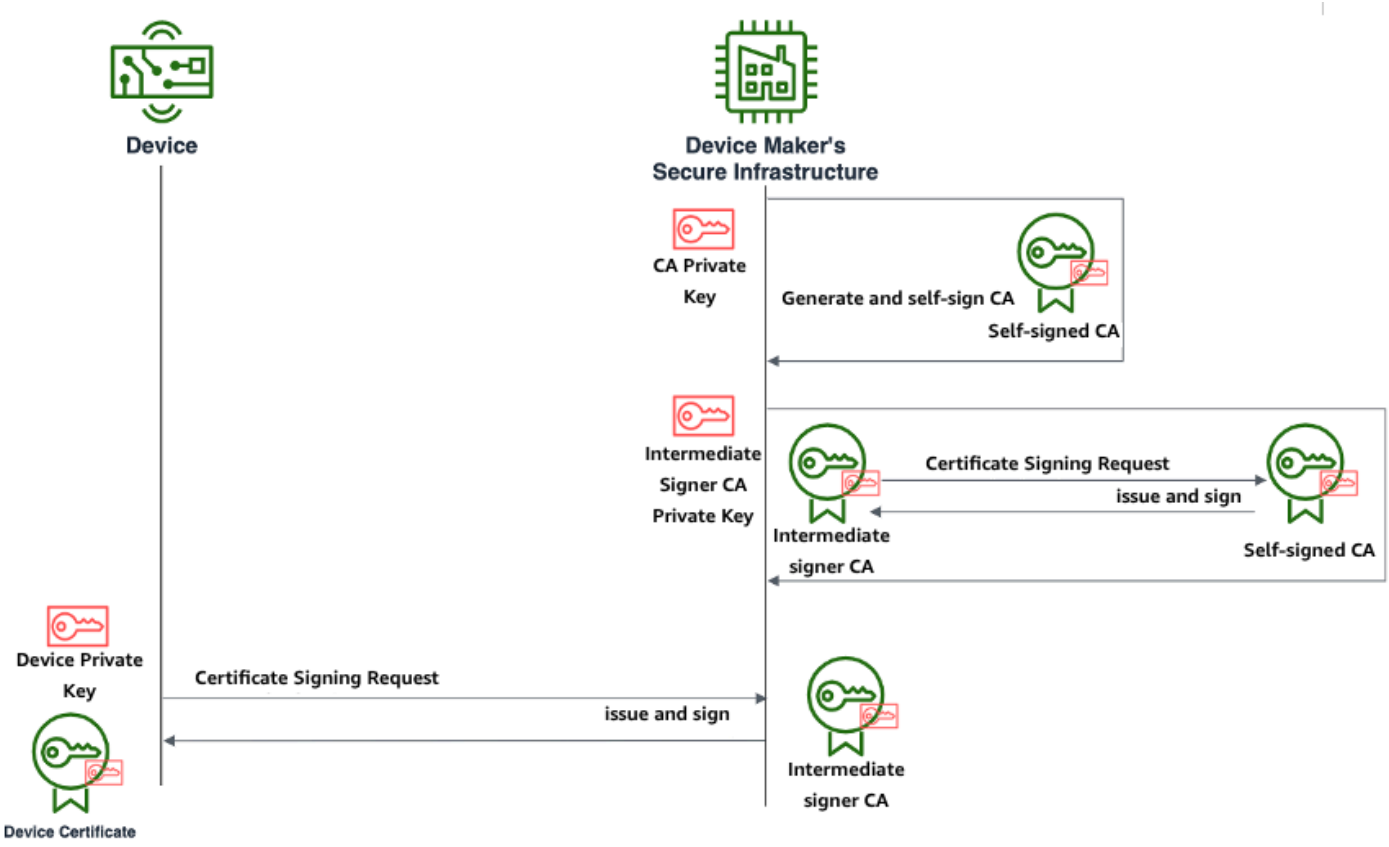
The certificates and private keys must be included in the firmware of each device, or provided to the contract manufacturer to deliver to the device. The device certificates are signed by a CA that is protected under the AWS Shared Responsibility Model, and the device maker does not need to implement their own controls on the CA. The device maker is responsible for the authorization policies granted to users of the AWS account to ensure only authorized users can generate new certificates.

Large enterprises may have their own self-signed root CA. A self-signed CA provides the greatest level of flexibility and control over the public key infrastructure. It is necessary to employ strict security protocols to protect the CA from being compromised, such as being air gapped with strict physical access controls. Intermediate CAs signed by the root CA are typically established though a key signing ceremony.

Enterprise PKI typically consists of a chain of one or more intermediate signing certificate authorities to enable compartmentalization and severability in the PKI. This allows the device

maker to use separate signing CAs across multiple contract manufacturers for more strict control of the certificate revocation list by allowing an intermediate certificate to be revoked without affecting the rest of the certificate infrastructure.

If the device maker wants to maintain control of the CA and PKI, AWS IoT provides the option to use a customer-owned signing CA. Outside of the AWS cloud, devices typically interact with a customer-owned signing service through a secure network channel to provide a Certificate Signing Request to an intermediate signing CA during the manufacturing process. If the device cannot access the CA directly, the certificates and private keys can be pre-generated and loaded onto the device in firmware, on a hardware security module, or delivered over a secure local connection in the manufacturing process. The certificates must be registered and activated on AWS IoT before the device can connect.

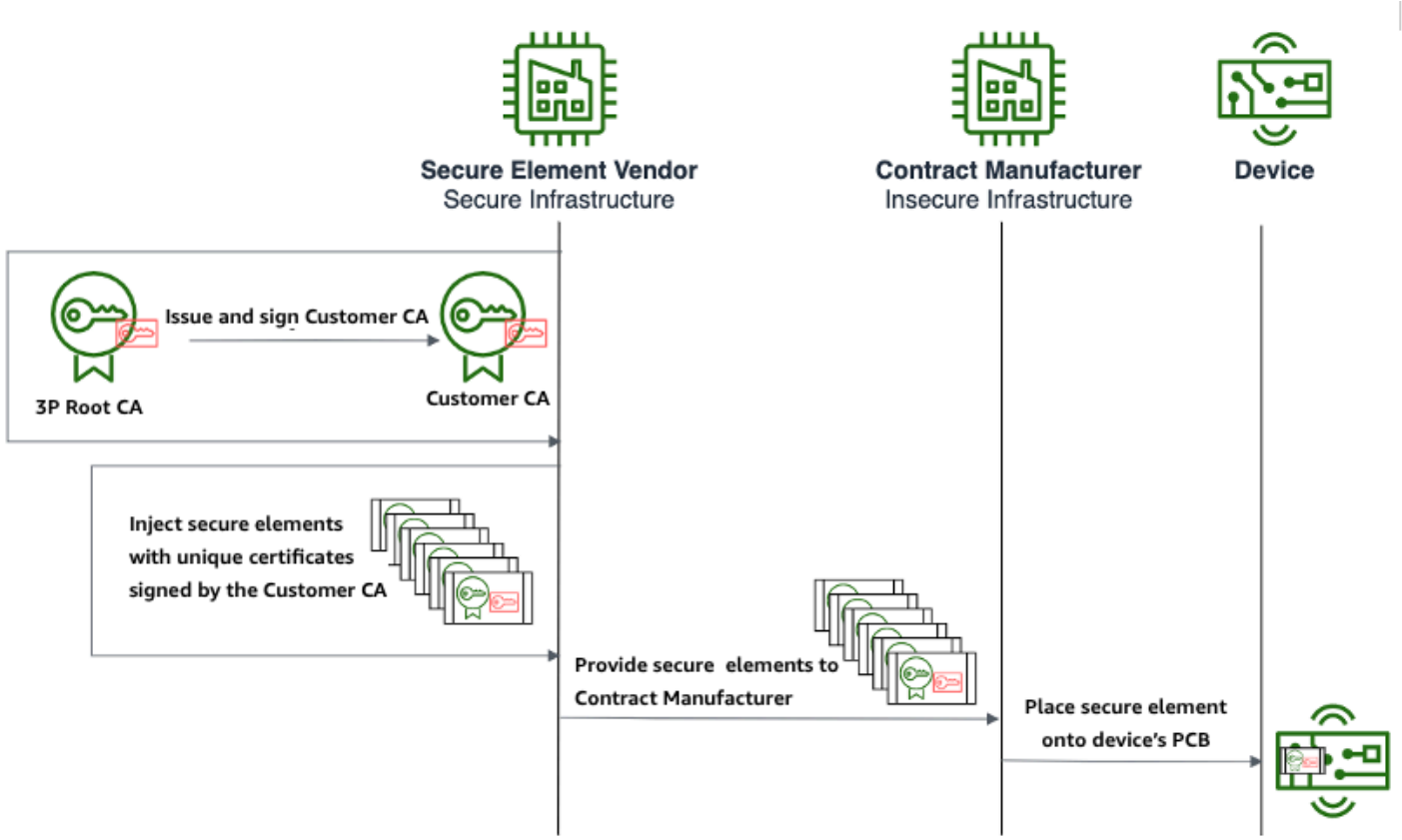


Self-signed certificate authority and intermediate signer CA infrastructure

AWS Certificate Manager (ACM) is a managed service on AWS that can generate and sign X.509 certificates in the cloud. The flexibility of ACM allows customers to bring their own CA and perform certificate signing operations on AWS. AWS protects the physical infrastructure where the CA is

held on the ACM service, and the device maker is responsible for enacting appropriate policies for users that have access to the ACM service in their account.

If the device maker does not want to maintain its own CA, but still wants to control the PKI for their assets, they can use CA services from third parties. These CA service companies generate an intermediate signer CA that is customized to the device maker's specification or they sign certificates from their own root CA. Third-party CAs give the device maker the ability to generate and sign X.509 certificates, but the third party maintains the physical security of the CA. Hardware security module vendors typically offer this service to pre-provision their modules before shipping them to the contract manufacturer.



Third-party Certificate Authority with Hardware Security Modules

Provisioning identity in AWS IoT Core for device connections

AWS IoT Core has different options to provision and onboard a large number of devices, depending on the capabilities of the device, and if the devices have their unique X.509 certificate and private keys on them before being sold to the end customer.

If the manufacturing chain allows the device maker to provision unique credentials into the device at manufacturing time or in distribution, device makers can use Just in Time Provisioning (JITP), Just in Time Registration (JITR), or Multi-Account Registration (MAR).

If it is not possible to establish unique credentials on the device before it is sold to the end customers, device makers may use Fleet Provisioning to onboard their devices.

Just-in-Time Provisioning

Devices that use Just-in-Time Provisioning (JITP) have certificates and private keys present on the device before onboarding to AWS IoT. The certificates must be signed with the customer's designated CA, and that CA must be registered in AWS IoT. Certificates generated on AWS IoT cannot be used with the just-in-time method. The customer must know which account the device will connect to before provisioning.

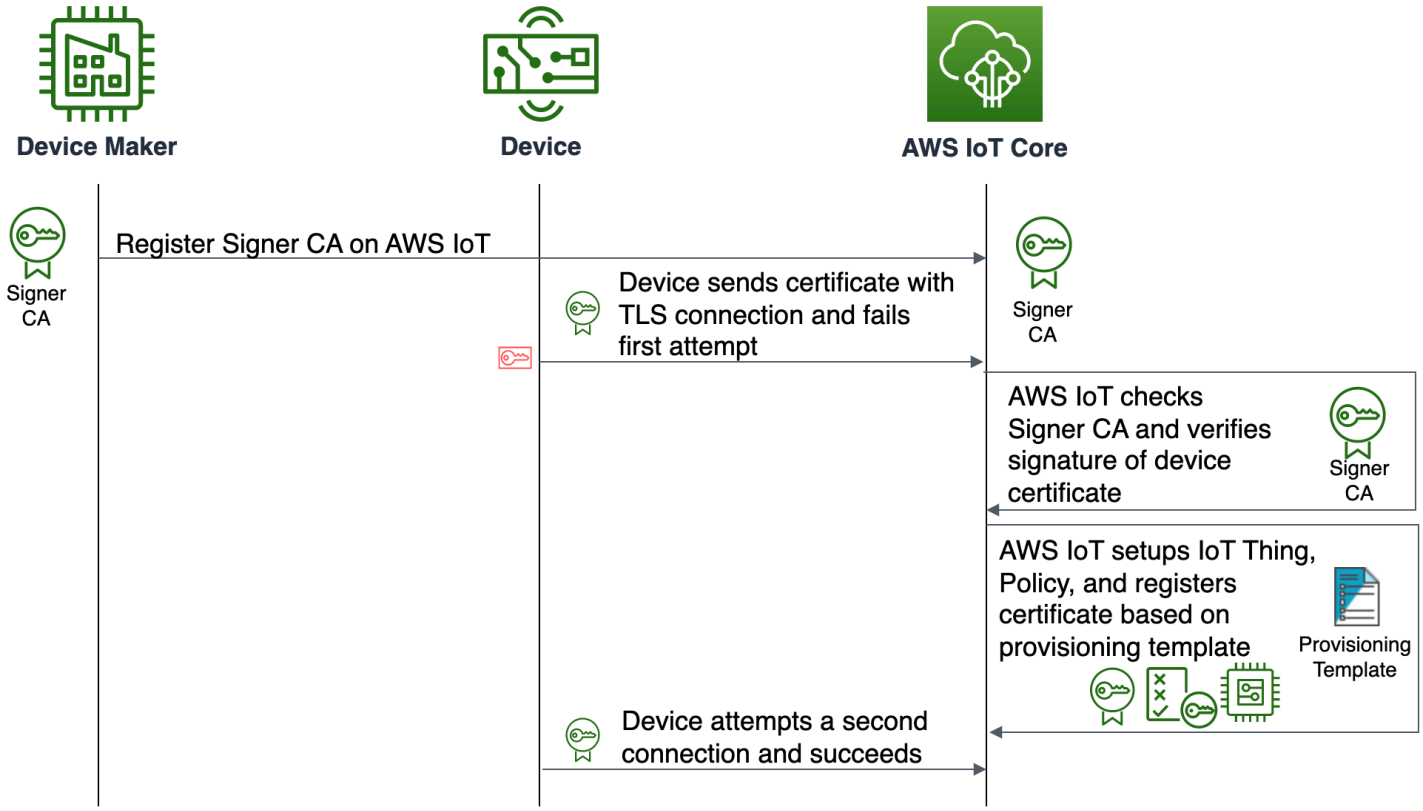
Setup

Using JITP, the device connects to AWS IoT Core, and the certificate's signature is verified against the registered CA. After verification, a provisioning template registers the Thing and certificate, and assigns a policy to the device. The device maker is responsible for registering the signer CA and attaching a provisioning template to the CA.

Device logic

When the device connects to AWS IoT Core for the first time, the device certificate must be sent during the TLS handshake. The signer CA must also be sent during the TLS handshake if the device does not send the Service Name Indicator (SNI) during the connection. The TLS handshake will fail at the first connection. This happens because the certificate has not been pre-loaded into the AWS IoT account. The device-supplied certificate is registered and activated in AWS IoT Core during the provisioning process. The device must have logic to reconnect to AWS IoT Core after a short

time period. If the provisioning operation has succeeded, the device will connect to AWS IoT Core successfully.



Just-in-Time Provisioning process

Just-in-Time Registration

Just-in-Time Registration (JITR) should be used if additional custom logic is required when the device is registered on AWS IoT Core. Like JITP, certificates and private keys must be present on the device before onboarding and the signer CA must be loaded into the AWS account before the device onboards.

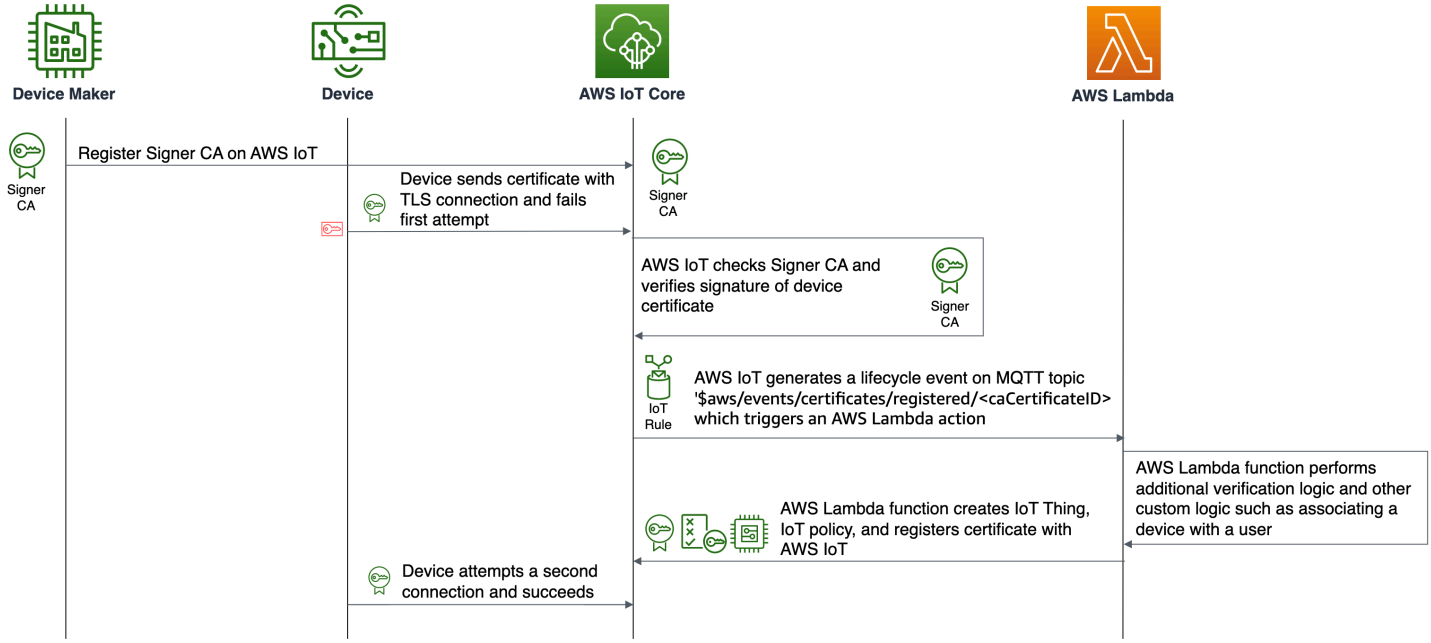
Setup

When the device first connects to AWS IoT Core, the certificate's signature is verified against the Certificate Authority and the certificate is registered in an inactive state. AWS IoT Core generates a Lifecycle event on the MQTT topic `$aws/events/certificates/registered/<caCertificateID>`. The device maker sets up an IoT Rule that cues an AWS Lambda function whenever a message is published on that topic.

The Lambda function can perform actions such as secondary validation against an allow list or Certificate Revocation List, register the device to a particular user on the cloud platform, or initiate additional onboarding workflows. The Lambda function sets the certificate's state to Active after additional validation is completed. The Lambda function should also create the IoT Thing, set up the Policy, and associate both with the certificate.

Device logic

Just like JITP, the client certificate must be sent during the TLS handshake, and the signer CA must be sent if the device does not support SNI. The device will fail to make the first connection to AWS IoT Core. The device must contain logic to reconnect to AWS IoT Core after the first failed connection. If the Lambda function activates the certificate, the second device connection will succeed.



Just-in-Time Registration process

Use cases for Just-in-Time Provisioning and Registration

JITP and JITR are used when the device maker knows at manufacturing time into which AWS accounts and/or Regions the device will be onboarded. The CA must be registered in the account and Region before the devices attempt a first connection. The just-in-time process requires a one-time setup and scales to millions of devices.

Manual registration without a CA

Existing device client certificates can be registered manually, without requiring a registered CA in one or more AWS IoT Core accounts. Once a unique certificate and private key are provisioned on the device, the certificate can be registered to IoT Core to enable the device to connect. Certificates are typically signed with a CA, but that CA is not required to be registered with AWS IoT Core with this manual registration method. Certificates can be registered in any Region and in any AWS account where the device maker has access. This method is often referred to as Multi-Account Registration, or MAR.

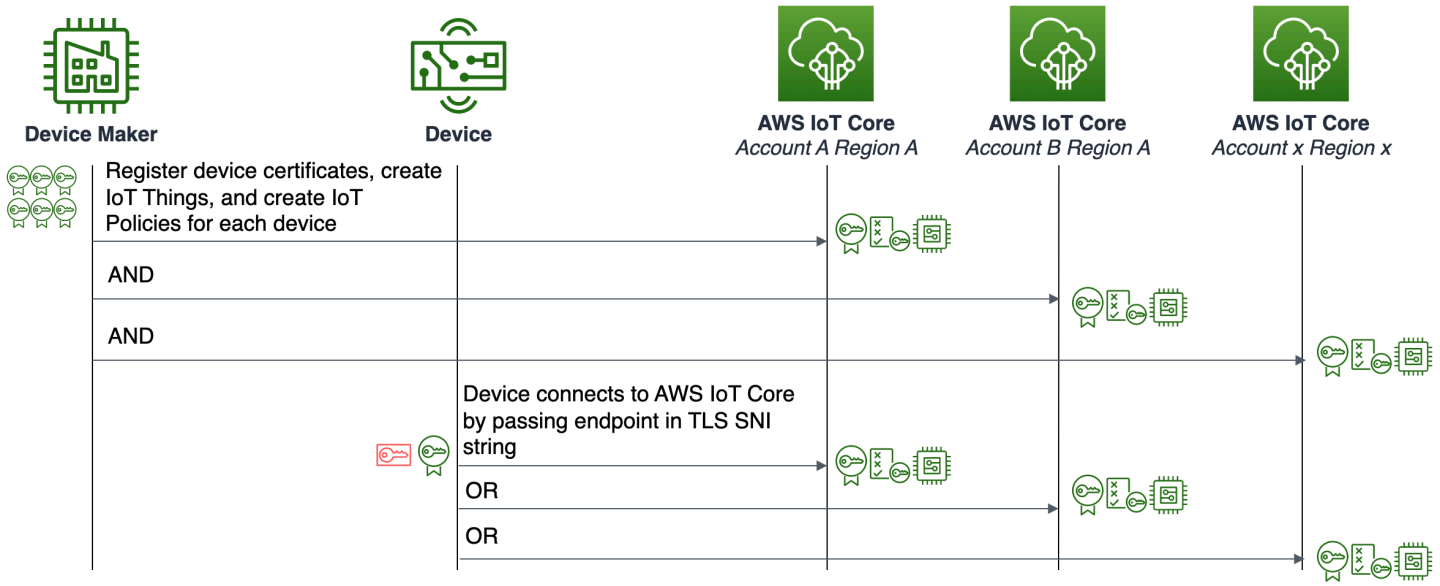
Setup

Device makers are required to set up the necessary resources in AWS for each device. The resources are set up before the device connects to AWS IoT Core for the first time.

Some vendors provide hardware security modules that are pre-provisioned with a private key and certificate. The certificate is signed with the vendor's own CA, and the vendor provides a manifest of certificates to the device maker. The device maker is responsible for registering the certificates in each account and Region using the AWS Management Console, AWS IoT Control Plane API, or the AWS Command Line Interface (AWS CLI).

Device logic

The device's TLS stack must support the SNI extension, and the AWS IoT Core endpoint is passed in the SNI string from the device. No additional device logic is required.



Manual registration without a CA process

Use cases for manual registration without a CA

Manual registration without a CA is used if flexibility is needed when determining which account and Regions a device may connect to. Device makers and service providers might have multiple AWS accounts used for sandbox, testing and production. Certificates can be pre-registered in each AWS account, and the device can connect to different accounts throughout its lifecycle by changing the IoT endpoint on the device. Devices can be sold globally, and the certificate can be registered in multiple AWS Regions as well as multiple AWS accounts within the same Region. This method allows the device makers to provide the public X.509 client certificate of a device to an IoT service provider so that they may manually register it to one or more their accounts.

Fleet provisioning

There are cases when provisioning unique credentials to a device at manufacturing time is not feasible due to technical limitations, cost, or application specific requirements. Devices that are not customized from the manufacturer are sold to customers without a unique identity. Fleet provisioning provides two ways to provision devices with unique credentials after they are delivered to end customers: by trusted user or by claim.

Fleet provisioning by trusted user

AWS IoT Core provides an application programming interface (API) that allows mobile applications to generate temporary certificates and private keys. The device leaves the manufacturing facility with no unique credentials, and only trusted users are able to provision the device with its unique credentials.

An installer uses a mobile application and authenticates with AWS. The mobile application should call the CreateProvisioningClaim API that returns a temporary X.509 certificate and private key that is valid for five minutes. Using the mobile application, these temporary credentials are delivered to the device. The device connects to AWS IoT and exchanges the temporary credentials for a unique X.509 certificate signed with an ephemeral AWS CA and corresponding private key. During this workflow, the AWS resources including Thing, Policy, and Certificate are set up in the AWS account.

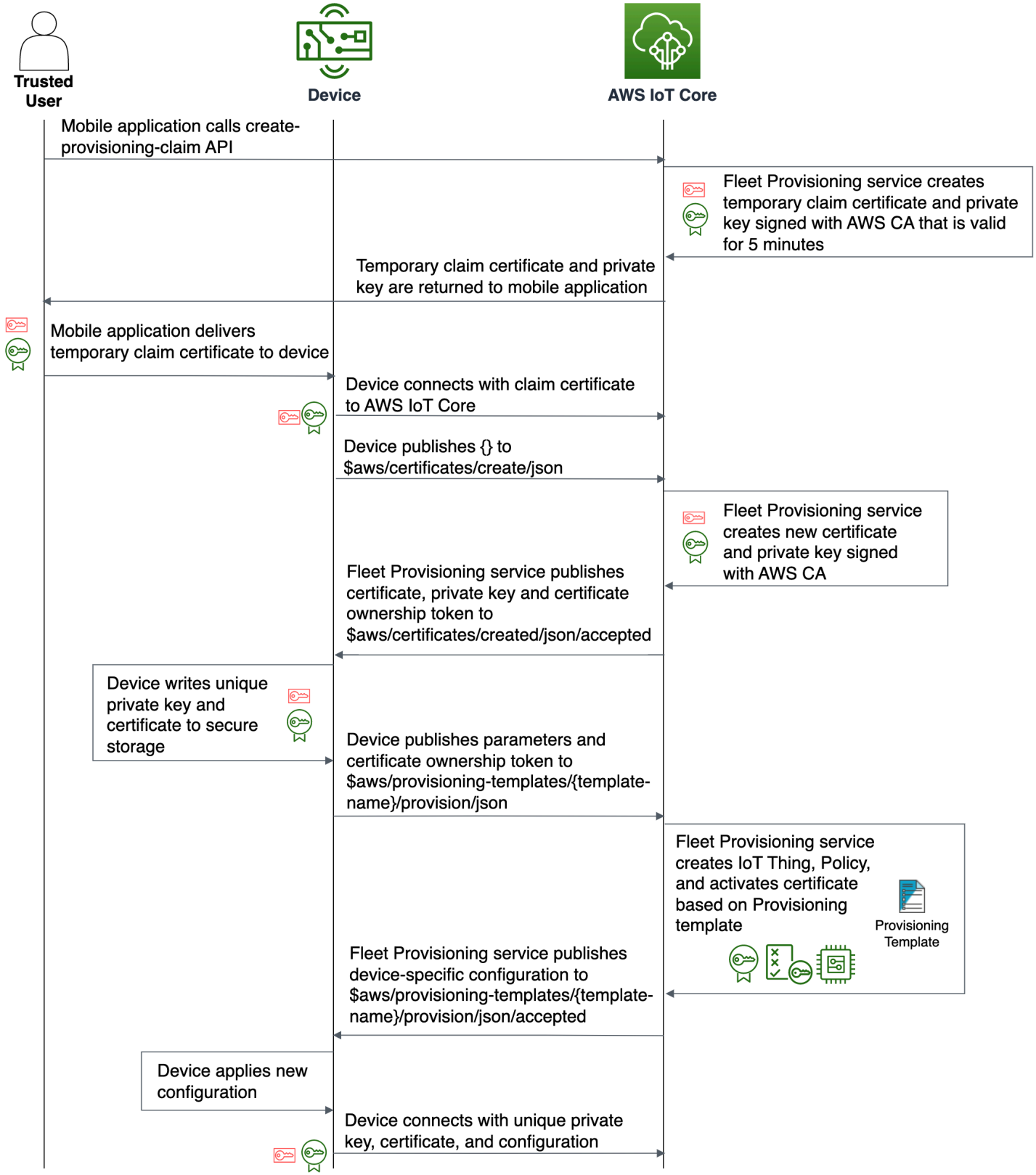
Setup

Device makers that use the Trusted User flow must develop and maintain a mobile application that uses the `CreateProvisioningClaim` API. The fleet provisioning template must be set up

and maintained in AWS IoT Core by the device maker. Optionally, a pre-provisioning AWS Lambda function is recommended to provide additional authentication steps during the provisioning process.

Device logic

Devices must have the ability to accept temporary credentials over a secure connection such as Bluetooth Low Energy, WiFi, or USB. Devices must implement the logic necessary to publish and subscribe to fleet provisioning MQTT topics, accept the permanent credentials, and write the credentials to secure storage.



Fleet provisioning by Trusted User process

Use cases for fleet provisioning by trusted user

The credentials are never exposed to the manufacturing supply chain. Fleet provisioning by trusted user is the recommended approach when a high degree of security is needed, when the manufacturing chain is not trusted, or when it is not possible to provision devices in the manufacturing chain. Trusted users must be authorized to perform the provisioning operations and the device used to generate the temporary credentials should be secure.

Fleet provisioning by claim

If a device does not have the capability to securely generate unique keys or the manufacturing supply chain is not equipped to uniquely provision devices at manufacturing time, the device maker should use a shared claim certificate that gets loaded on the devices. AWS IoT Core provides a path for these devices to receive a unique identity when they first connect to AWS IoT Core.

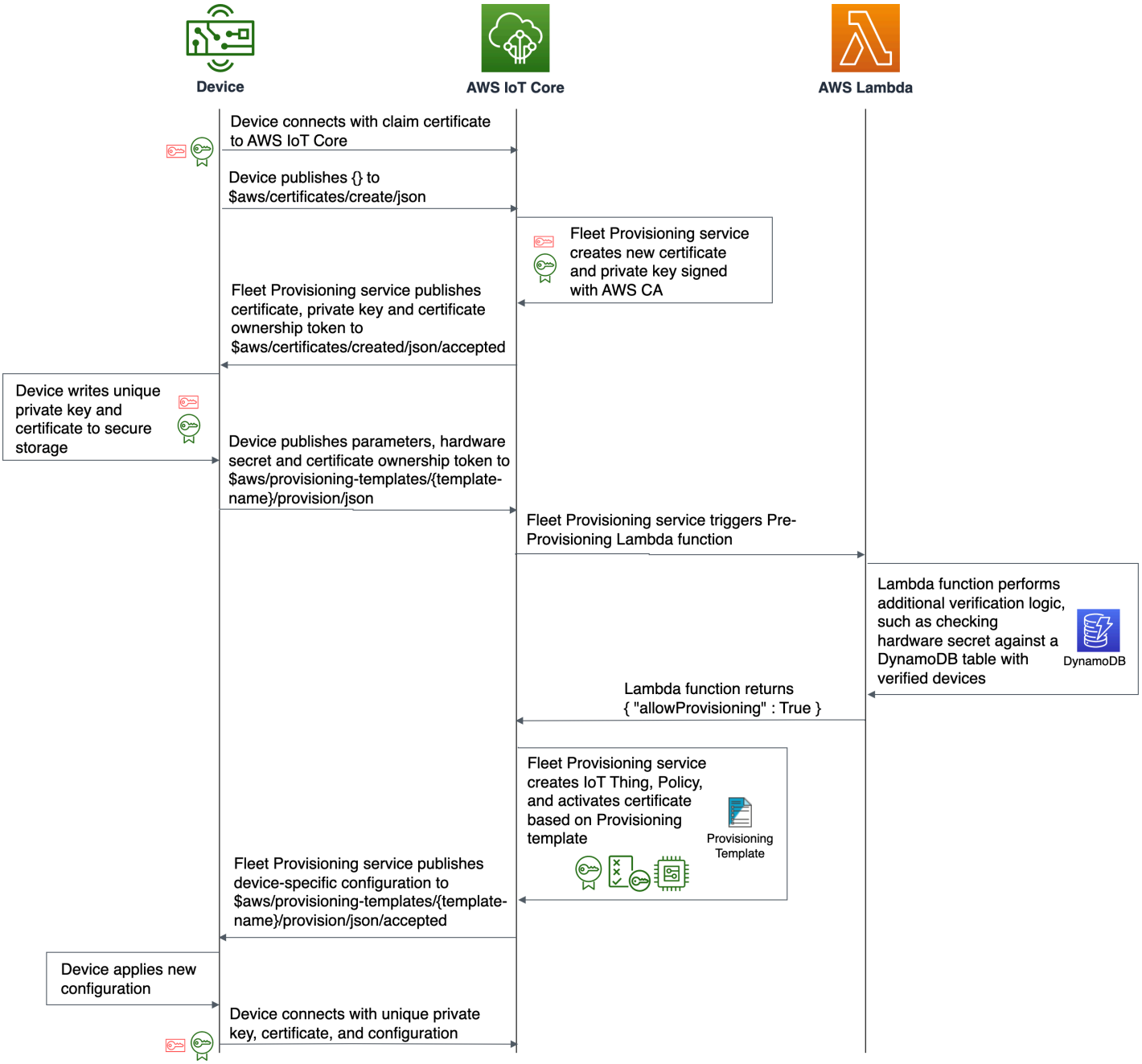
Fleet provisioning by claim requires device makers to use a shared claim certificate and key on all devices. The claim certificate should be unique per batch of devices to limit the number of devices affected in the event that the claim certificate's private key is compromised and the certificate needs to be revoked. If the claim certificate is revoked, devices that share this certificate will no longer be able to use it to complete the provisioning process. The firmware containing the claim certificate is loaded by the contract manufacturer without the need to perform any customization. When the device establishes a connection with AWS IoT Core for the first time, it exchanges the claim certificate for a unique X.509 certificate signed by the AWS certificate authority and a private key. The device should send a unique token, such as an embedded hardware secret with its provisioning request that the fleet provisioning service can use to verify against an allow list using the pre-provisioning Lambda in order to validate the authenticity of the device.

Setup

Device makers that use fleet provisioning by claim must maintain a fleet provisioning template and AWS Lambda function with additional verification logic. The claim certificates must be protected and frequently audited to prevent misuse.

Device logic

Devices must implement the logic necessary to publish and subscribe to fleet provisioning MQTT topics, accept the permanent credentials, and write the credentials to secure storage.



Fleet provisioning by claim process

Use cases for fleet provisioning by claim

Fleet provisioning by claim should only be used when it is not possible to provision unique credentials in the device, and when it is not possible to use the provisioning by trusted user option. Devices that enter insecure channels such as a third-party distributor should not use fleet

provisioning by claim because of the risk in exposing the claim credentials that are common to every device prior to the initial connection to AWS IoT Core.

Reference architectures

The provisioning approaches described in this whitepaper provide the building blocks for a device provisioning and onboarding solution to AWS IoT Core. Device makers might require additional components or a combination of these approaches in order to meet their requirements. AWS provides several open-source device onboarding and provisioning architectures that are deployable and extensible for device makers and service providers. This section explores two onboarding architectures that build upon these: zero-touch provisioning (ZTP) and the Device Lobby.

Zero-Touch Provisioning

Device makers may use a ZTP service provider to manage their public key infrastructure, and provision certificates and private keys to their device. ZTP service providers allow device makers to manage device security, identity and registration independent from the device manufacturing supply chain. Devices will be manufactured with a unique identifier known to the ZTP service provider and stored on a secure location on the device. This unique identifier is traded for a device certificate during the onboarding stage of the device's lifecycle over the air and automatically.

ZTP provides several benefits to the device maker. The device maker can manage their device manufacturing supply chain without prior knowledge of where the device will ultimately connect. The onboarding and registration of the devices are done programmatically and in large quantities, usually separated by manufacturing batches. This simplifies onboarding a large number of devices to multiple AWS accounts. The service provider's ZTP service is a known and trusted authority that the device can fall back to during factory resets, transfer of device ownership, and decommissioning. The service provider may also provide value-add services such as remote attestation, third-party Certificate Authorities, and secure manufacturing infrastructure.

ZTP is a common design pattern implemented by telecommunications providers for cellular IoT devices and hardware security module vendors.

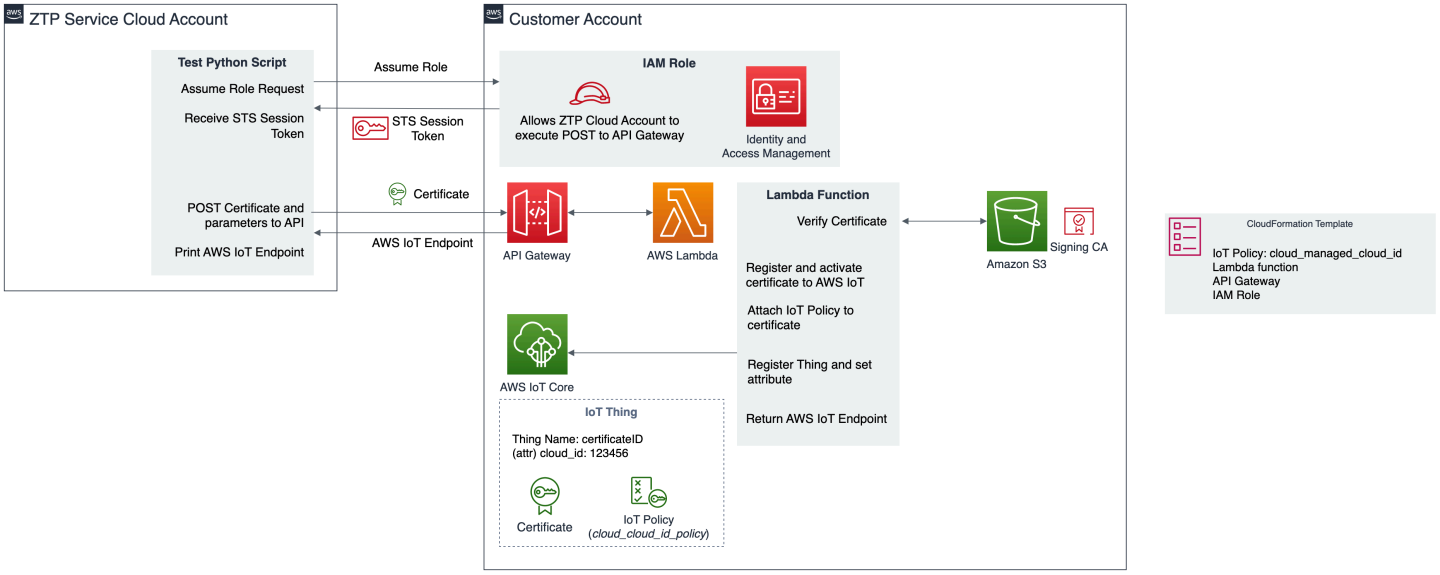
ZTP implementation

ZTP service providers provision identities into devices in their own secure infrastructure. Devices first connect to the service provider's ZTP service, usually through a device agent that the service provider provides. The service provider is responsible for managing the customer's Public Key Infrastructure, device certificates, and connectivity management.

The ZTP service provider is also responsible for registering the device to the device maker's AWS account. In this scenario, the device maker must have a secure API for the service provider to register the X.509 certificate, Thing name, and IoT Policy into their account. Once the service provider registers the certificate to the customer's account, the device can securely connect to the customer's AWS IoT broker endpoint.

AWS has mechanisms to keep this transaction secure, robust, and reliable. The customer creates an AWS Identity and Access Management (IAM) role for the service provider to assume. IAM provides fine-grained access control across all of AWS. With IAM, you can specify who can access which services and resources, and under which conditions. The IAM Role implements a policy that ensures the service provider can only access specific APIs and actions within the device maker's account, such as registering an IoT Thing and Certificate.

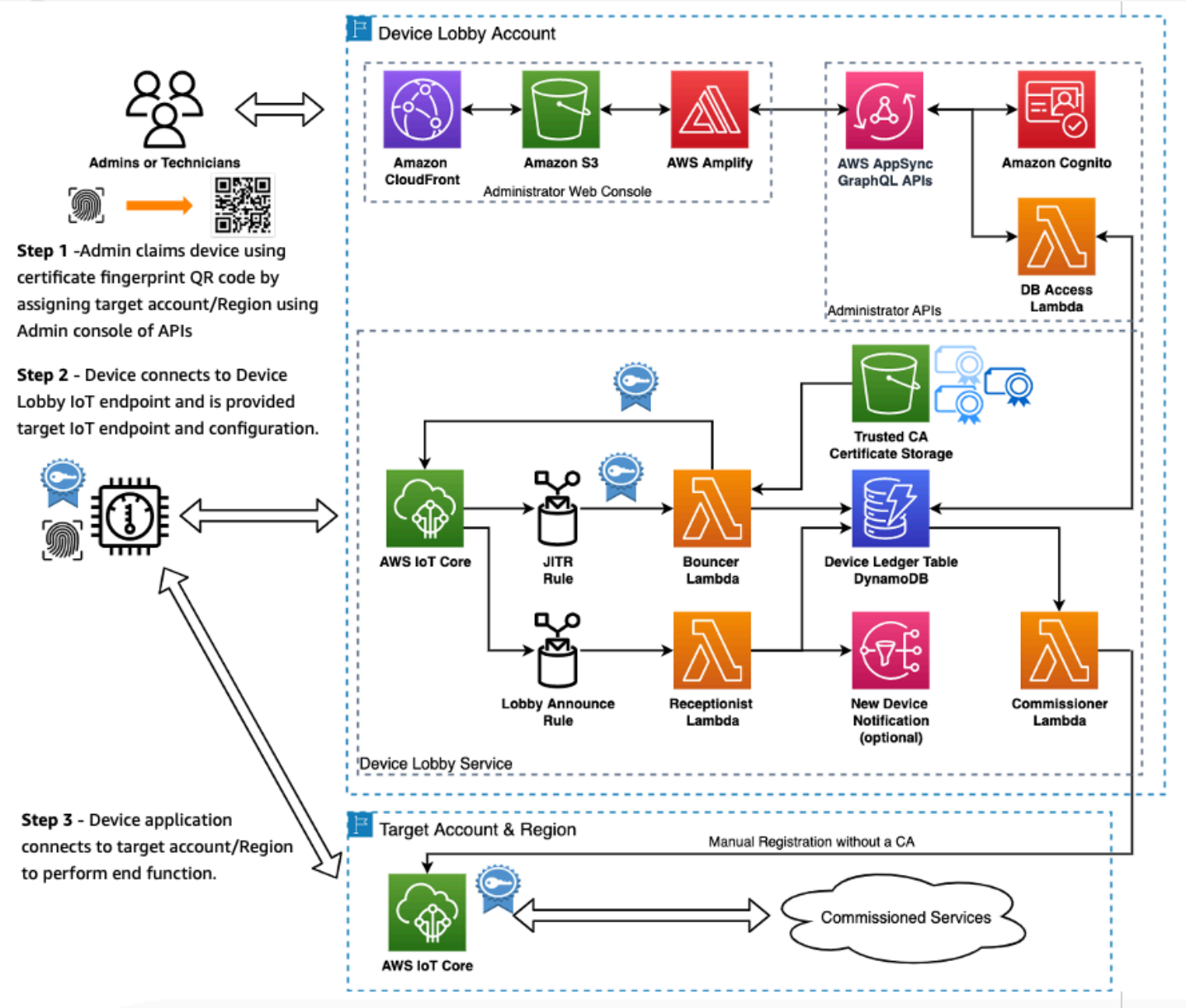
A ZTP reference implementation is provided on the `aws-samples` Github repository, which includes a CloudFormation template, AWS Lambda sample code, a sample third-party CA, a deployment script, and a testing script. The reference implementation sets up a secure API and IAM role in the device maker's account. The ZTP service provider assumes the IAM role and sends an HTTP POST to the secure API with a device certificate. The device certificate is verified against a known certificate authority. The device certificate, IoT Thing name, and an IoT Policy are set up in the device maker's account. The device maker's AWS IoT broker endpoint is returned to the ZTP service provider to forward to the device. The device can then connect to the device maker's AWS IoT endpoint.



Zero-touch provisioning reference architecture

Device lobby

The IoT device lobby architecture establishes an entry point in AWS Cloud infrastructure to route or bootstrap devices to end cloud services. It provides a serverless infrastructure to associate a device identity by using the X.509 fingerprint to a target AWS account or Region, regardless of whether a device is turned on. The device fingerprint can be printed as a QR code for easy scanning and onboarding of devices to target accounts and Regions by the administrator APIs at any point in a device lifecycle without reprovisioning. This helps fleet operators take advantage of the AWS IoT global footprint, and can effectively decouple the manufacturing and provisioning of devices from the end cloud services where they connect once deployed in the field.



Device lobby onboarding architecture

The architecture combines JITR and manual registration methods described earlier in this whitepaper with the inclusion of a global Amazon DynamoDB table to act as the device ledger and an administrative interface for managing the ledger to claim and route devices. JITR enables the initial connection to the lobby for previously unseen devices when they present a certificate signed by a CA trusted by the Device Lobby account. Manual registration without a CA is then used by the backend to register devices into target accounts or Regions once claimed by an administrator. MQTT topics are used to control the interaction with the device and service.

Use cases for the Device lobby:

- **Commissioning devices for a target account or Region during installation -** A technician or user installs the device, and the customer account or Region must be associated at the time of install – for example, technicians claiming devices at time of install using authorized mobile device or app.
- **Offline commissioning of devices in the supply chain** - Devices are delivered to a customer and need to connect to the correct endpoint when powered up. The customer account or Region is known at time of shipment, but the device is in a box and not powered on – for example, offline binding in the supply chain.
- **Account or Region migration of previously fielded devices** - Devices need to be moved among accounts, environments, or Regions without changing credentials for factory returns, refurbishment, redeployment, or simply moving devices during development between dev/staging/prod environments.
- **Disaster recovery orchestration** - A given Region becomes unavailable and devices need to be dynamically routed in the field to another account or Region with functioning application infrastructure.

Device lobby implementation

Building a device lobby implementation on AWS requires both device application logic and AWS Cloud components.

Device logic

The device onboarding logic required to work with the Device Lobby service is intended to be simple and present a minimal impact to the ROM/RAM footprint of the device. Two MQTT topics

are used to control the interaction with the service in order for the most constrained devices to not require any additional protocol stacks.

The device is required to implement an onboarding state machine that determines its operational mode and whether to connect to the Device Lobby endpoint(s) to be commissioned, or to the target account endpoint to perform its end function. The commissioned state and target IoT endpoint can be kept in non-volatile storage, so passage through the lobby for the routing happens only once or until cleared by a factory reset. A factory fresh device could be provisioned with one or more lobby account IoT endpoints to use by default, or the endpoint could be configured by the end user.

The device should always inform the cloud of its core identity presented as the X.509 device certificate. As the identity of a device is not known to the Device Lobby prior to first connection, the X.509 certificate provisioned to the device should contain everything needed to uniquely identify the device and establish the source of trust. The device needs to be provisioned with the keys and X.509 certificate issued by the manufacturer's PKI. The certificate should contain the unique name of the device in the CN field of the certificate Subject and have a long-lived expiration that exceeds the expected lifecycle of the device.

With properly established credentials and lobby endpoint configured, the device can then be routed to any end AWS account at any point in its lifecycle with the Device Lobby administrator APIs or console.

Cloud service components:

- Device ledger table – The device ledger table is implemented as a global DynamoDB table that holds the device identity and target account/Region association. The device certificate fingerprint serves as the primary key for the table. New fingerprint entries are made by either:

- The Bouncer Lambda for new, unclaimed devices connecting to the service, or
- Administrators using the AppSync APIs and/or web console.

- **Bouncer Lambda** – This Lambda function is responsible for admitting devices to connect to the Device Lobby account. In the example implementation, the Just-in-time Registration flow is used to verify a device's certificate against a trusted CA, register the device certificate, extract the Thing name and register or activate the device so it can connect to the Device Lobby account with minimal permissions. Other access methods to the lobby are possible, such as fleet provisioning or the custom authorizer feature of IoT Core.

- **Receptionist Lambda –** Once a device has successfully connected to the lobby account, this Lambda function is cued through the basic ingest lobby rule, where the device publishes to announce its presence in the lobby. The lobby rule extracts the certificate ID or fingerprint from the certificate used for the connection and device name from the MQTT client ID, and passes any other MQTT payload data on to the Receptionist Lambda. When initiated by a device announcement, the Lambda function will attempt to retrieve the commissioned IoT endpoint for the fingerprint and will publish it back on the device-specific lobby topic. If no endpoint exists in the ledger table, the function does nothing and the device can loiter in the lobby indefinitely, periodically announcing its presence until a commissioned IoT endpoint is returned.
- **Commissioner Lambda –** This function is responsible for watching the stream from the device ledger table and registering devices into the target account or Region using manual registration without a CA. The function registers a device only when a complete commissioning entry is available in the table. A complete entry includes the device certificate and Thing name provided by the device in the initial connection to the lobby, and a target account or Region that has been written by the administrative APIs. The function assumes a role in the target account with the appropriate permissions to read the target IoT endpoint and register certificates, Thing names, and policies. Once the device and certificate are registered in the target account, the function writes the commissioned IoT endpoint to the device entry in the ledger table, indicating that the device has been successfully registered in the target.
- **Administrative APIs and web portal** – Administration of the device ledger table is performed through AWS AppSync GraphQL APIs that are authenticated by Amazon Cognito. The enables strict access control to the device ledger table using administrator-specified user pools. From an administrative standpoint, only the device fingerprint (QR code) and the target account or Region are needed to claim and route devices through the lobby. The sample web portal provides a basic, authenticated React application for monitoring devices in the lobby, and claiming them with using a built in QR code reader function.
- **Target AWS account –** To receive devices routed by the lobby, the target account must have a role defined for the Commissioner Lambda function to assume and the trust linkage established back to the Commissioner Lambda function Amazon Resource Name (ARN) in the lobby account.

The goal of the device lobby architecture is to enable device manufacturers or product teams to produce single-SKU IoT devices that can be easily and securely onboarded by service operators to any AWS account/Region by simply scanning a QR code.

A reference implementation and quickstart guide for the Device Lobby architecture is available on the `aws-samples` Github repository, which includes a CloudFormation template, deployment scripts, admin console web app with QR reader, and sample device implementations.

Conclusion

Moving an IoT project from proof of concept to a large-scale deployment is a complex challenge. Device makers and service operators must make multiple decisions that fundamentally impact the device's capabilities, level of security, bill of materials, and product and operational cost. Production and deployment decisions must be considered early on in the development process because they will impact manufacturing and deployment costs.

Device makers must consider the following when creating IoT devices:

- Who will maintain the public key infrastructure for their project?
- What capabilities should they build into their device?
- What capabilities do their contract manufacturer have to provide customization at manufacturing times?

  Service operators must also consider similar questions when considering which devices to use:

- How are devices trusted to connect to the service platform?
- How and when will devices onboard to the service platform from a manufacturer?
- Will the IoT service require multi-Region or multi-account device deployments?

This whitepaper has outlined the options that AWS IoT provides for device makers and service operators as they answer these questions. No matter the capabilities of the device and manufacturing process, the level of trust in the manufacturing supply chain or security requirements, AWS IoT provides options to onboard and trust devices at scale in a secure way.

Contributors

Contributors to this document include:

- David Walters, Senior Partner Solutions Architect, IoT
- Michael Schy, Senior Partner Solutions Architect, IoT
- Ashok Bhaskar, Senior Partner Solutions Architect, IoT
- Tim Mattison, Principal Partner Solutions Architect, IoT
- Alok Jha, Senior Product manager, AWS IoT Core
- Ben Cooke, Senior Partner Solutions Architect, IoT

Document revisions

To be notified about updates to this whitepaper, subscribe to the RSS feed.

Change	Description	Date
Refresh including updated service features, reference architectures, and restructuring	Updates.	November 17, 2022
Initial publication	Whitepaper published.	December 1, 2020

Notices

Customers are responsible for making their own independent assessment of the information in this document. This document: (a) is for informational purposes only, (b) represents current AWS product offerings and practices, which are subject to change without notice, and (c) does not create any commitments or assurances from AWS and its affiliates, suppliers or licensors. AWS products or services are provided “as is” without warranties, representations, or conditions of any kind, whether express or implied. The responsibilities and liabilities of AWS to its customers are controlled by AWS agreements, and this document is not part of, nor does it modify, any agreement between AWS and its customers.

AWS Glossary

For the latest AWS terminology, see the AWS glossary in the *AWS Glossary Reference*.