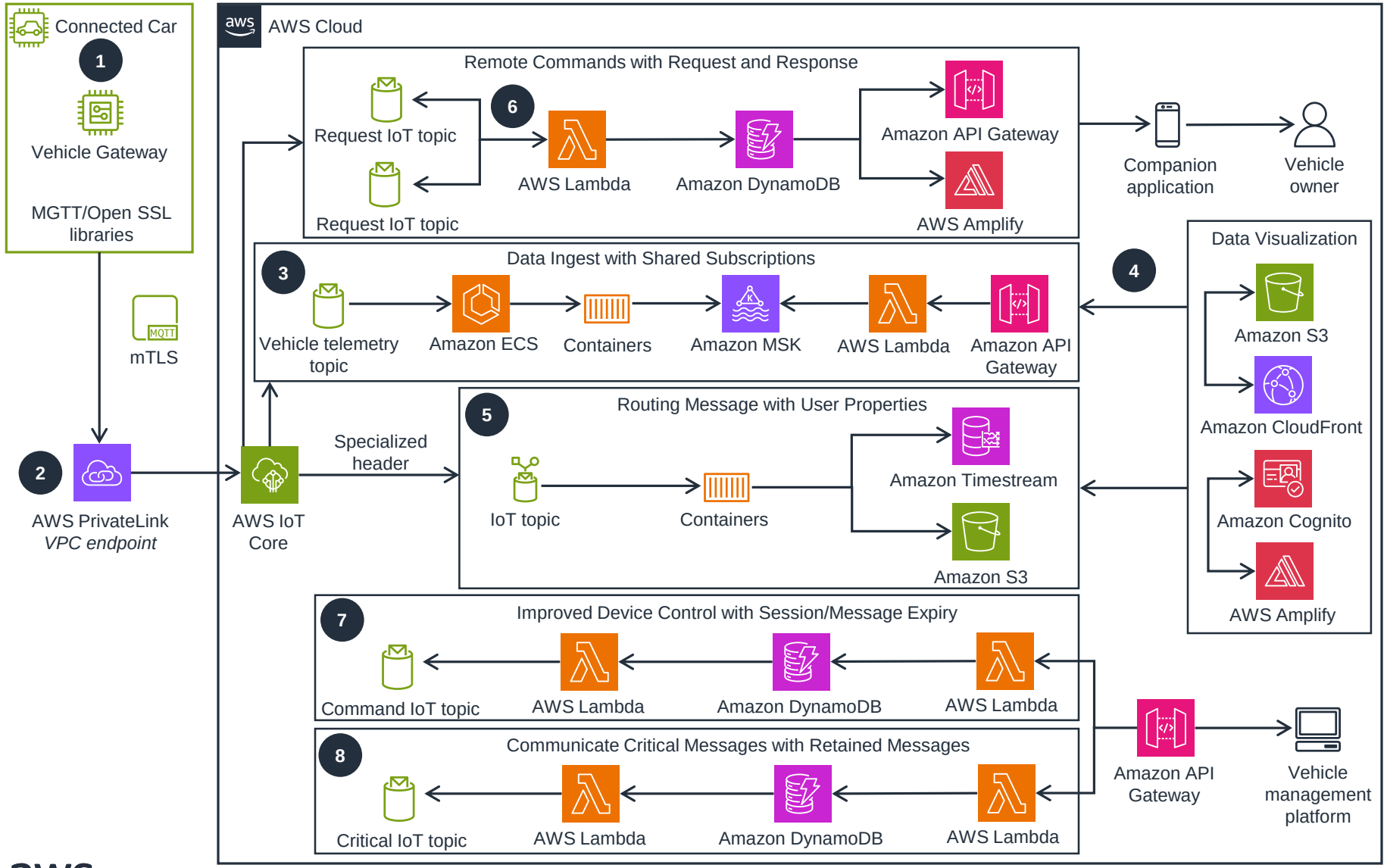
Guidance for Connected Vehicles on AWS

Modernized Connected Vehicles: This architecture diagram uses AWS IoT Core and MQTT to modernize your broker and gather, collect, and distribute data with your connected vehicle workloads.



1. Embedded in-vehicle devices with a unique identity principal (X.509 certificate) publish telemetry data to **AWS IoT Core** by using MQTT. To minimize in-vehicle software, only libraries necessary to connect to **AWS IoT Core** are implemented. All traffic is sent over the MQTT protocol and is secured using the latest versions of mTLS.

2. Use **AWS PrivateLink** and a private cellular network to connect your own **AWS IoT Core** endpoint.

3. Shared subscriptions on **AWS IoT Core** help client workers process data payloads coming in from millions of vehicles by load balancing across topics. Implement the topic alias feature to reduce payload size over the cellular connection.

4. Use **Amazon Elastic Container Service (Amazon ECS)** to decode, process, and persist the telemetry data. Using standard patterns, you can build a visualization application with **Amazon API Gateway** and **AWS Amplify** based on the data in **Amazon Timestream**.

5. **AWS IoT Core** routes encoded messages to their destination, such as **Timestream** for hot storage and **Amazon Simple Storage Service (Amazon S3)** for cold storage. The descriptor file is stored in **Amazon S3** to inform the rule how to decode the payload.

6. The request/response messaging pattern in **AWS IoT Core** asynchronously tracks responses to client requests. The client submits a remote command to their vehicle using an interface built by **Amplify** through **API Gateway**. The command is persisted to **Amazon DynamoDB** and delivered as a request to the device by using a request IoT topic, so the vehicle can report success or failure and state on the response IoT topic.

7. Use the Message Expiry feature of **AWS IoT Core** to specify how long to attempt to unlock a vehicle door before expiring the message on the broker. This pattern would call for a REST API through **API Gateway** and record the command in **DynamoDB** using **AWS Lambda** prior to posting to the **AWS IoT Core** topic.

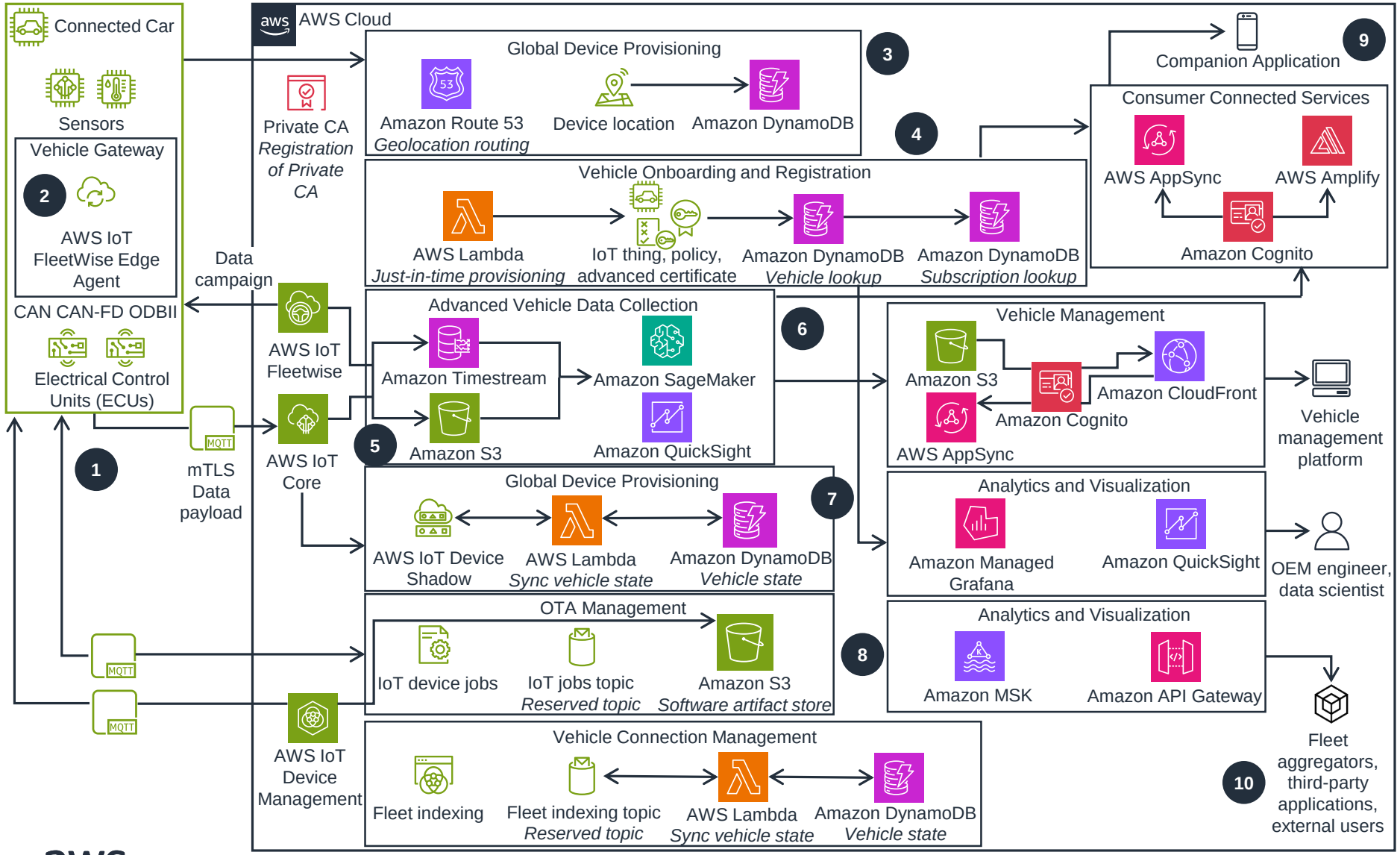
8. Critical system messages sent from the original OEM to the vehicle can use the **AWS IoT Core** Retained Messages function. The OEM can set a retained message flag on the payload to ensure the command is in the topic when the vehicle comes back online using the same services as Message Expiry.

Reviewed for technical accuracy November 13, 2023

AWS Reference Architecture

Guidance for Connected Vehicles on AWS

Data Gathering and Processing: This architecture diagram demonstrates how to gather, process, analyze, and act on connected vehicle data using AWS IoT Core. It also shows how to visualize and share data with stakeholders.



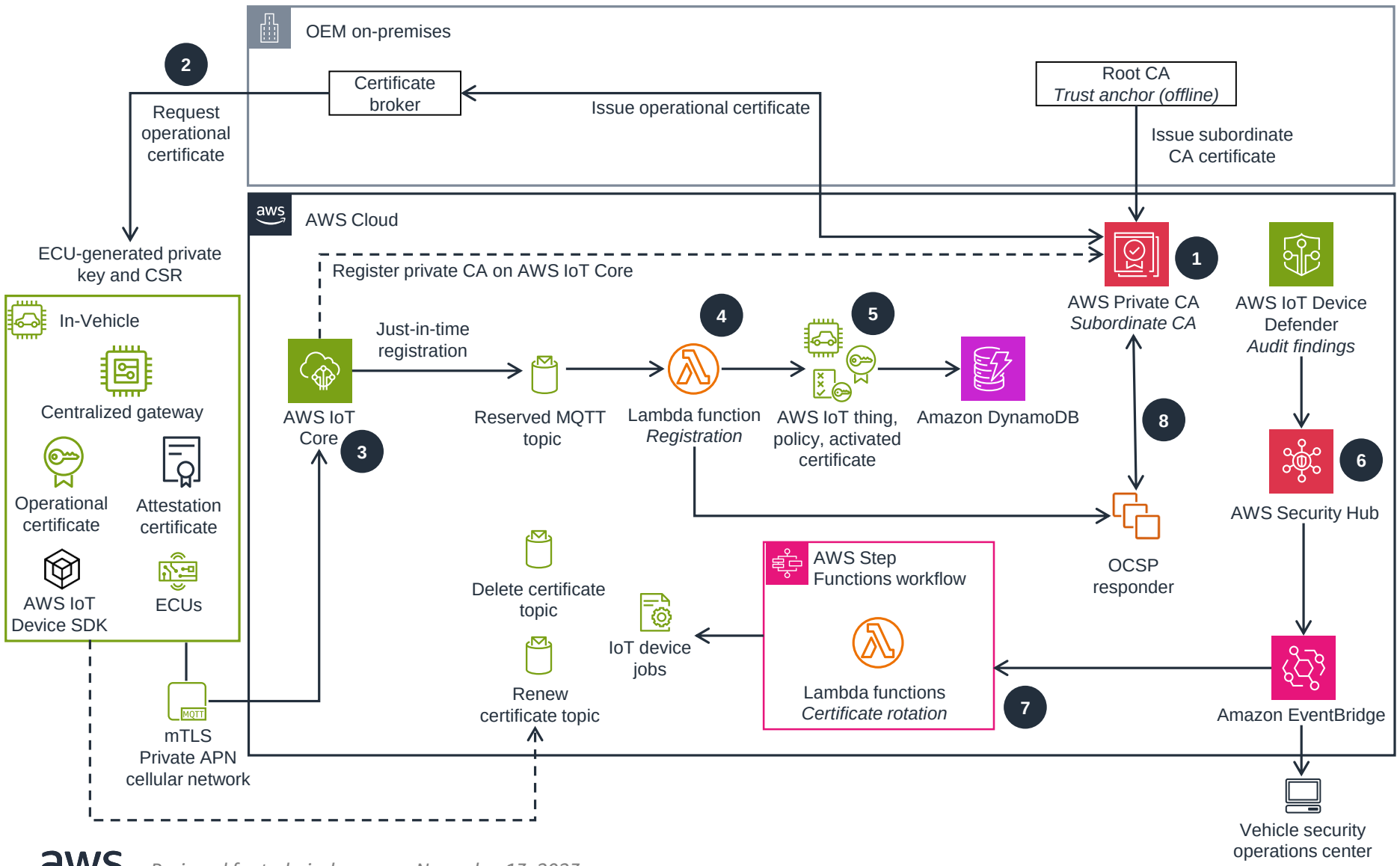
1. Acting as an IoT device, the connected vehicle with a unique identity principal (X.509 certificate) uses sensors to collect, analyze, and act upon data and **AWS IoT Core** as an edge-to-cloud communication mechanism.
2. The **AWS IoT FleetWise** Edge Agent communicates with the vehicle's network through **AWS IoT Core** as defined by data campaigns. You control every step of the process and maintain full data ownership and control of proprietary information.
3. Use **AWS IoT Core** with **Amazon Route 53** to choose an AWS Region based on geo-location or latency. Register your devices automatically when they connect for the first time to **AWS IoT Core**. The DNS lookup returns an IoT endpoint from one of many Regions depending on device location.
4. **AWS IoT Core** supports registered client certificates signed by any root or intermediate certificate authorities (CAs). The **Lambda** function validates the gateway and creates the IoT thing, policy, and certificate to register the vehicle.
5. **AWS IoT FleetWise** enables time- and event-based data collection campaigns that send relevant data to **Timestream** or **Amazon S3**.
6. Use **Amazon SageMaker** to improve Advanced Driver Assistance Systems and Autonomous Vehicle (ADAS/AV) models and vehicle design. Analyze data from **AWS IoT FleetWise** in **Amazon QuickSight** to improve vehicle quality.
7. **AWS IoT Device Shadow** makes a vehicle's state available to other downstream services, providing a built-in mechanism to update the vehicle's state from the cloud, even if not connected **to AWS IoT Core**.
8. Use **AWS IoT Device Management** to implement over-the-air (OTA) management through IoT jobs and use **AWS IoT Core fleet indexing** to manage state, connectivity, and device violations to troubleshoot your device fleet.
9. Implement a companion application with **Amplify** using **Amazon Cognito** for authentication and **AWS AppSync** with GraphQL mutations enabled to **DynamoDB** for a near real-time view of the vehicle's last known state.
10. **Amazon Managed Streaming for Apache Kafka (Amazon MSK)** allows anonymized and aggregated telemetry data to be published to other data aggregators and external users. **Amazon Managed Grafana** and **QuickSight** provide vehicular data visualizations.

Reviewed for technical accuracy November 13, 2023

AWS Reference Architecture

Guidance for Connected Vehicles on AWS

Operational Certificate Lifecycle: This architecture diagram shows an operational certificate lifecycle by demonstrating how to secure connected vehicles with provisioning, the Online Certificate Status Protocol (OCSP), and certificate rotation.



1. A subordinate CA is created in **AWS Private Certificate Authority** (AWS Private CA) with a CA certificate signed by the offline root CA. The subordinate CA certificate is registered with **AWS IoT Core**.

2. The electronic control unit (ECU) generates a private key and certificate signing request (CSR) and uses its existing attestation certificate to authenticate to the certificate broker. The certificate broker issues the operational certificate by calling **AWS Private CA**. The broker sends the signed operational certificate to the ECU.

3. The ECU uses TLS with the operational certificate to connect to **AWS IoT Core**, which validates the client certificate was signed by the registered CA certificate and that the certificate is not expired. Because this is the first use of the certification, it is not registered with **AWS IoT Core**; **AWS IoT Core** creates a pending activation certificate and the ECU is disconnected. **AWS IoT Core** publishes a message to a reserved MQTT topic.

4. An IoT rule on the reserved MQTT topic invokes the **Lambda** registration function. This implements custom logic, such as generating an **AWS IoT Core** policy specific to the ECU using information from a **DynamoDB** table, and custom authentication, such as checking the certificate against the Online Certificate Status Protocol (OCSP) responder provided by **AWS Private CA**.

5. The function creates an IoT thing and policy and changes the certificate status to active. The ECU can retry the connection and communicate to topics in **AWS IoT Core**.

6. **AWS IoT Device Defender** publishes audit findings to **AWS Security Hub**, which sends events to **Amazon EventBridge** to initiate targets (such as an **AWS Step Functions** workflow) and rotate the certificate. **EventBridge** can also send audit findings to your vehicle security operations center.

7. The workflow orchestrates **Lambda** functions that use IoT jobs to push the ECU to generate a new CSR and send it to a topic. Upon receiving the CSR, it issues a new operational certificate by calling **AWS Private CA** to sign the certificate, registers the certificate in **AWS IoT Core** with a policy, and sends the certificate to the ECU. Once the ECU has successfully installed and tested the new operational certificate, the workflow revokes the old certificate.

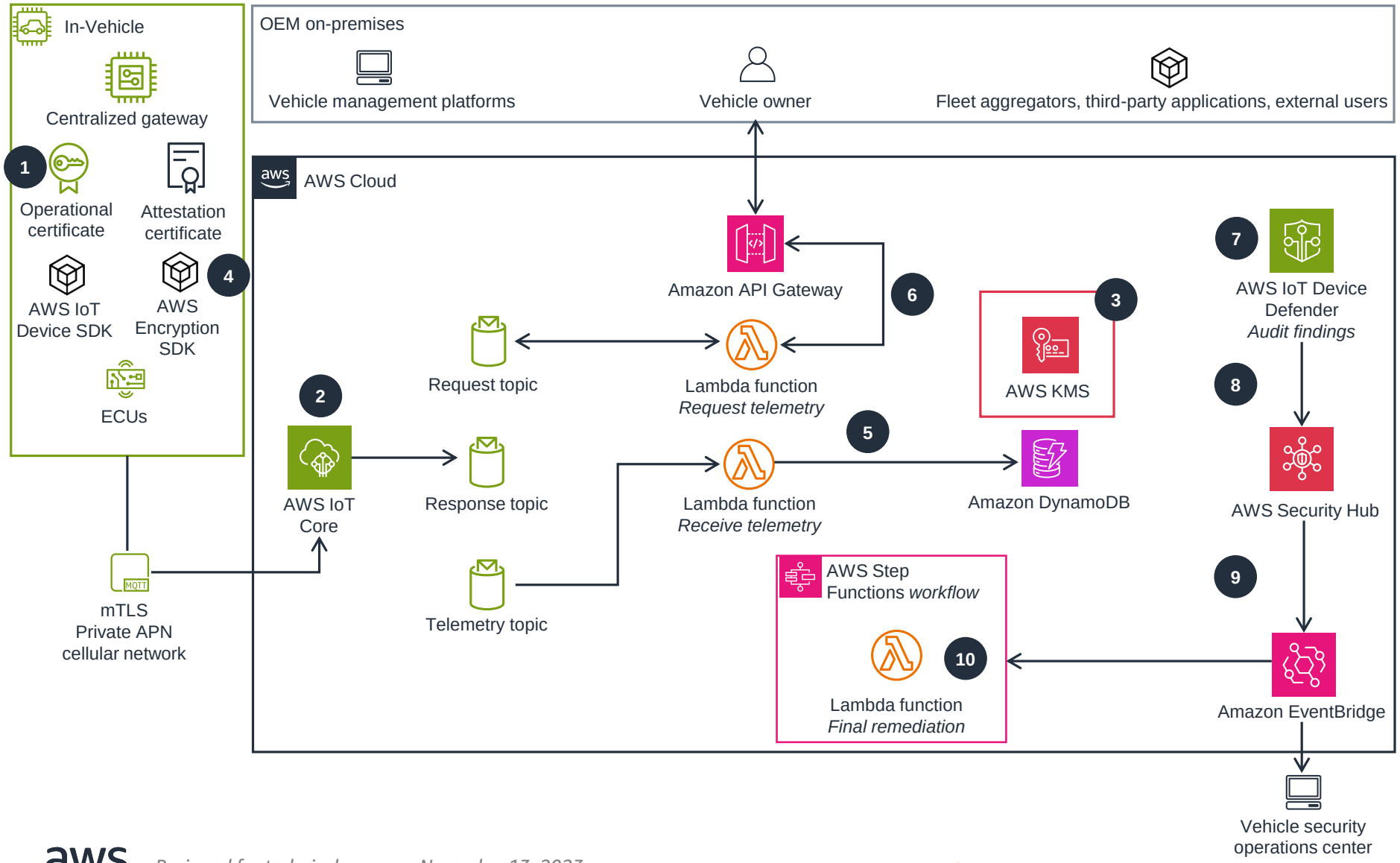
8. You can invoke **AWS Private CA** APIs to revoke a certificate and update certificate status in **AWS IoT Core**.

Reviewed for technical accuracy November 13, 2023

AWS Reference Architecture

Guidance for Connected Vehicles on AWS

Encryption and Monitoring Security: This architecture diagram shows how to secure a connected vehicle with AWS encryption and monitoring services. It also demonstrates a remediation workflow for security findings once detected.



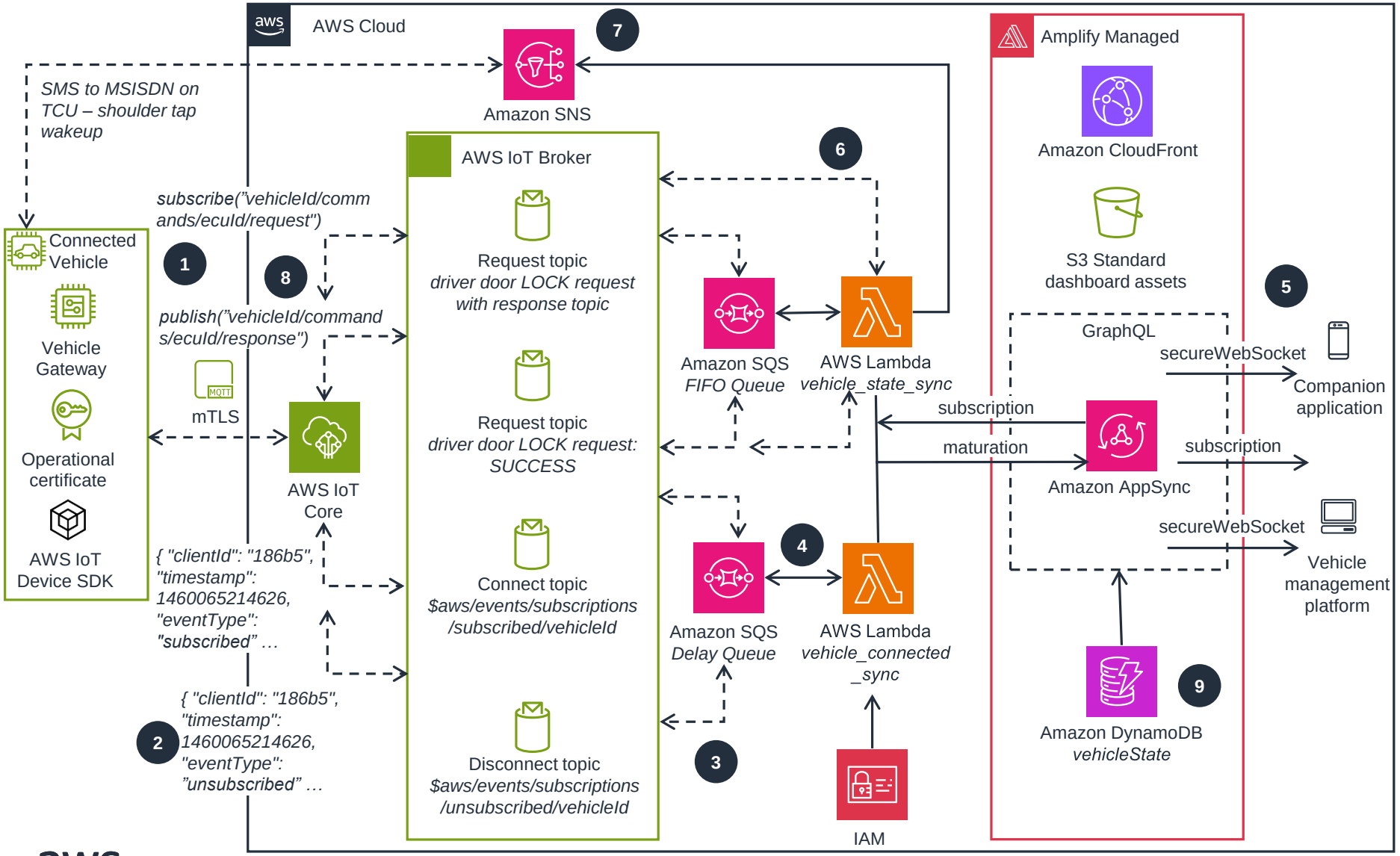
1. An ECU with a unique identity principal (X509 operational certificate) publishes telemetry by using MQTT to **AWS IoT Core**. The ECU can either run a generic HTTP or MQTT stack or accelerate development using the **AWS IoT Device SDK**.
2. During the TLS handshake, **AWS IoT Core** validates client certificate expiry and checks that the certificate is registered and active. **AWS IoT Core** retrieves policies attached to the certificate and thing groups for the thing attached to the certificate to authorize ECU operations.
3. **AWS Key Management Service (AWS KMS)** lets you create, manage, and control cryptographic keys. AWS services can be encrypted at rest on the server side.
4. You can encrypt highly sensitive payload data on the client side before sending it to **AWS IoT Core** in addition to encryption-in-transit using mTLS. The vehicle can use the **AWS Encryption SDK** and **AWS KMS** keys for client-side encryption. The ECU can get temporary API credentials from the **AWS IoT Core** credential provider to call **AWS KMS** APIs.
5. Sensitive payload data flows through intermediate systems as opaque ciphertext until it arrives at a **Lambda** function that has an execution role with permissions to invoke **AWS KMS**. The **Lambda** function code uses the **AWS Encryption SDK** to decrypt data.
6. Users can send authenticated and authorized remote commands to applications through **API Gateway**. The request **Lambda** function with execution role permissions can use the **AWS Encryption SDK** and **AWS KMS** keys to encrypt client-side command payloads before sending to the ECU.
7. **AWS IoT Device Defender** monitors devices connected to **AWS IoT Core** to detect abnormal behavior using rules and by building machine learning (ML) models.
8. **AWS IoT Device Defender** sends findings to **Security Hub** for aggregation and normalization.
9. **Security Hub** sends findings to a vehicle security operations center or **EventBridge**, which routes them to a **Step Functions** remediation workflow**.**
10. The **Step Functions** remediation workflow orchestrates steps, such as modifying the IoT policy for the certificate and changing the certificate status to inactive to disconnect the ECU.

Reviewed for technical accuracy November 13, 2023

AWS Reference Architecture

Guidance for Connected Vehicles on AWS

Companion Application: This architecture diagram demonstrates how to build a connected vehicle companion application to control a vehicle with AWS IoT Core and AWS AppSync.



1. The vehicle establishes an MQTT connection to the **AWS IoT Core** endpoint then subscribes to the control plane request topics to receive any cloud-side request commands. The vehicle will publish automatically to the **AWS IoT Core lifecycle event** topic, indicating that connectivity is established.
2. Upon connection, **AWS IoT Core** publishes the vehicle's connected state to the lifecycle events topic: *$aws/events/subscriptions/subscribed/vehicleId.*
3. When the topic receives a lifecycle event, enqueue a message to **Amazon Simple Queue Service (Amazon SQS)** FIFO Queue to maintain order of messages. When that message becomes available and is processed by **Lambda**, you can check if the device is still offline before taking further action.
4. The connected state is sent as a mutation to **AWS AppSync** which uses a custom resolver to persist the state to a **DynamoDB** table. This connected state is then used for logic when a remote command is sent from a companion application.
5. Using **Amplify**-managed native applications and web applications, a remote command is sent through a secure WebSocket as a mutation to **AWS AppSync**. That mutation is persisted to **DynamoDB**, and **Lambda** then processes a subscription.
6. The vehicle state **Lambda** checks the connected state (if connected) then publishes the command payload to the request topic with a unique requestId and a response topic.
7. If the device is in a disconnected state, the vehicle state **Lambda** then sends a command to **Amazon Simple Notification Service (Amazon SNS)** which will send an SMS to the disabled Mobile Station International Subscriber Directory Number (MSISDN) on the SIM on the Telematics Control Unit (TCU) to wake up and subscribe to command topics.
8. Upon receipt of the command payload in the request topic, the logic is managed by a client application on the device, and the result is published back to the response topic as success or failure. The response payload is then sent to an **SQS** FIFO queue for downstream processing by the vehicle state **Lambda**.
9. **AWS AppSync** processes the request as a mutation, which is persisted to **DynamoDB.** **AWS AppSync** alerts the companion applications and vehicle management platforms with the updated vehicle state.

Reviewed for technical accuracy November 13, 2023

AWS Reference Architecture