This solution describes a robust, secure, and automated framework for external business partners (B2B clients) to obtain mTLS client certificates. It integrates Google Cloud's **Certificate Authority Service (CAS)** for secure certificate management, **Apigee API Gateway** for controlled external access, a custom **Spring Boot microservice** for orchestrating requests, and **Shared VPC** for private and secure internal network communication.

### 1. Overall Architecture Diagram

Code snippet

graph TD  
 subgraph GCP Organization  
 ORG[GCP Organization Policy Enforcement]  
 end  
  
 subgraph "Host Project (Shared VPC)"  
 HP\_VPC(Shared VPC Network)  
 HP\_FIREWALL(Shared Firewall Rules)  
 end  
  
 subgraph "Central CAS Service Project (Security/PKI Team)"  
 direction LR  
 CSP\_CAS\_POOL\_LABEL[GCP CAS: B2B CA Pool]  
 CSP\_CA\_INTERMEDIATE(Intermediate CA for B2B mTLS)  
 CSP\_CERT\_TEMPLATE\_B2B(Certificate Template: B2B\_MTLS\_TEMPLATE)  
 CSP\_CLOUD\_AUDIT\_LOGS(Cloud Audit Logs: CAS Activity)  
  
 CSP\_CAS\_POOL\_LABEL --- CSP\_CA\_INTERMEDIATE  
 CSP\_CAS\_POOL\_LABEL --- CSP\_CERT\_TEMPLATE\_B2B  
 end  
  
 subgraph "Your Service Project (Your Dev Team)"  
 direction LR  
 YSP\_SA(Service Account: my-app-cas-requester)  
 YSP\_SB\_MICROSERVICE[Spring Boot Microservice: Cert Request API]  
  
 YSP\_SB\_MICROSERVICE -- Deployed As --> YSP\_SA  
 end  
  
 subgraph "Apigee Project (API Management)"  
 direction LR  
 APIGEE\_GATEWAY[Apigee API Gateway]  
 APIGEE\_POLICIES(Apigee Policies: Auth, Quota, Spike Arrest)  
 APIGEE\_PROXY(API Proxy for Cert Request)  
  
 APIGEE\_GATEWAY -- Uses --> APIGEE\_POLICIES  
 APIGEE\_GATEWAY -- Routes Traffic Via --> APIGEE\_PROXY  
 end  
  
 EXTERNAL\_CLIENT[External Client / Partner App] -- 1. External Client Calls Apigee API (CSR) --> APIGEE\_GATEWAY  
  
 APIGEE\_GATEWAY -- 2. Applies Apigee Policies & Routes --> YSP\_SB\_MICROSERVICE  
  
 YSP\_SB\_MICROSERVICE -- 3. Authenticates to GCP (ADC) --> GCP\_IAM[GCP IAM]  
 GCP\_IAM -- 3a. Validates YSP\_SA Permissions (Cross-Project IAM) --> CSP\_CAS\_POOL\_LABEL  
 GCP\_IAM -- 3b. Validates YSP\_SA Permissions (Cross-Project IAM) --> CSP\_CERT\_TEMPLATE\_B2B  
  
 YSP\_SB\_MICROSERVICE -- 4. Calls CAS API (CreateCertificate) --> CSP\_CAS\_POOL\_LABEL  
  
 CSP\_CAS\_POOL\_LABEL -- 5. Issues Certificate (via CSP\_CA\_INTERMEDIATE, per CSP\_CERT\_TEMPLATE\_B2B) --> CSP\_CAS\_POOL\_LABEL  
 CSP\_CAS\_POOL\_LABEL -- 6. Returns Issued Cert & CA Chain --> YSP\_SB\_MICROSERVICE  
  
 YSP\_SB\_MICROSERVICE -- 7. Returns Issued Cert & CA Chain --> APIGEE\_GATEWAY  
  
 APIGEE\_GATEWAY -- 8. Returns Issued Cert & CA Chain --> EXTERNAL\_CLIENT  
  
 CSP\_CAS\_POOL\_LABEL -- 9. Logs all CAS operations --> CSP\_CLOUD\_AUDIT\_LOGS  
  
 EXTERNAL\_CLIENT -- 10. Client Configures mTLS (Keystore/Truststore) --> CLIENT\_KEYSTORE\_TRUSTSTORE(Client Local Keystore & Truststore)  
  
 %% Security Aspects (as separate nodes for clarity in Mermaid)  
 SEC\_ADC[Security: Application Default Credentials]  
 SEC\_CROSS\_IAM[Security: Cross-Project IAM Bindings]  
 SEC\_LEAST\_PRIV[Security: Principle of Least Privilege]  
 SEC\_CLIENT\_KEY[Security: Client Owns Private Key]  
 SEC\_CERT\_TEMPLATES[Security: CAS Certificate Templates]  
 SEC\_SHARED\_VPC[Security: Shared VPC for Private Comm.]  
 SEC\_AUDIT\_LOGS[Security: Comprehensive Audit Logging]  
 SEC\_APIGEE\_POLICY[Security: Apigee Policy Enforcement]  
 SEC\_APIGEE\_THREAT[Security: Apigee Threat Protection]  
  
 YSP\_SB\_MICROSERVICE --- SEC\_ADC  
 YSP\_SA --- SEC\_CROSS\_IAM  
 YSP\_SA --- SEC\_LEAST\_PRIV  
 EXTERNAL\_CLIENT --- SEC\_CLIENT\_KEY  
 CSP\_CAS\_POOL\_LABEL --- SEC\_CERT\_TEMPLATES  
 HP\_VPC --- SEC\_SHARED\_VPC  
 CSP\_CLOUD\_AUDIT\_LOGS --- SEC\_AUDIT\_LOGS  
 APIGEE\_GATEWAY --- SEC\_APIGEE\_POLICY  
 APIGEE\_GATEWAY --- SEC\_APIGEE\_THREAT

### 2. Detailed Setup Steps

#### A. Central CAS Service Project (Managed by Security/PKI Team)

1. **Enable CAS API:** Ensure the Certificate Authority Service API is enabled in this project.
2. **Create B2B CA Pool:**
   * Create a dedicated CA Pool (e.g., b2b-mtls-ca-pool) in your preferred region (e.g., asia-south1).
   * **Recommendation:** Use the **Enterprise tier** for its advanced features crucial for a B2B context (higher QPS, certificate storage, robust revocation).
3. **Create Intermediate CA:**
   * Within the b2b-mtls-ca-pool, create an Intermediate Certificate Authority. This CA will be used for issuing client certificates. It should be signed by an organizational Root CA (which might reside offline or in another highly secured CAS Pool).
   * Ensure its configuration specifies it as a non-CA certificate and includes appropriate key usages for end-entity certificates.
4. **Define B2B Certificate Template:**
   * Create a Certificate Template (e.g., B2B\_MTLS\_TEMPLATE) in the same region as the CA Pool.
   * **Crucial Configurations for mTLS Client Certificates (aligned with your Spring Boot validation):**
     + **Extended Key Usage:** Must include clientAuth (OID: 1.3.6.1.5.5.7.3.2).
     + **Key Usage:** Typically digitalSignature and keyEncipherment.
     + **Subject Configuration:** Define required Common Name (CN) and Subject Alternative Names (SANs) constraints.
       - For CN: enforce length limits and the regex pattern (e.g., ^[a-z0-9.-]+\.([a-z0-9-]+)\.example\\.com$).
       - For SANs: Specify allowed types (DNS, URI, IP) and potentially limits on the number of SANs.
     + **Validity Period:** Set a maximum lifetime (e.g., 365 days) that your Spring Boot microservice will respect.
     + **is\_ca: false:** Ensures issued certificates cannot sign other certificates.
     + **Path:** Record the full resource path for this template: projects/CENTRAL\_CAS\_PROJECT\_ID/locations/asia-south1/certificateTemplates/B2B\_MTLS\_TEMPLATE. This will be used in your Spring Boot application configuration.
5. **Grant Cross-Project IAM Permissions:**
   * Grant **your** service account (my-app-cas-requester@your-project-id.iam.gserviceaccount.com, from *Your Service Project*) the necessary permissions on the CAS resources in this central project.
   * **Roles to Grant:**
     + roles/privateca.certificateRequester on the b2b-mtls-ca-pool.
     + roles/privateca.certificateTemplates.use on the B2B\_MTLS\_TEMPLATE.

#### B. Your Service Project (Managed by Your Development Team)

1. **Create Service Account:**
   * Create a dedicated service account in *your* project (e.g., my-app-cas-requester@your-project-id.iam.gserviceaccount.com). This service account will be used by your Spring Boot microservice.
   * This service account *does not need additional IAM roles within your project* for this specific use case, as its critical permissions are granted by the central project.
2. **Spring Boot Microservice Development:**
   * **Project Setup (pom.xml):**  
     XML  
     <?xml version="1.0" encoding="UTF-8"?>  
     <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
      xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">  
      <modelVersion>4.0.0</modelVersion>  
      <parent>  
      <groupId>org.springframework.boot</groupId>  
      <artifactId>spring-boot-starter-parent</artifactId>  
      <version>3.2.7</version> <relativePath/> </parent>  
      <groupId>com.example.cas</groupId>  
      <artifactId>certificate-issuance-service</artifactId>  
      <version>0.0.1-SNAPSHOT</version>  
      <name>certificate-issuance-service</name>  
      <description>Microservice for issuing mTLS certificates via GCP CAS</description>  
       
      <properties>  
      <java.version>17</java.version>  
      <google-cloud.version>0.246.0</google-cloud.version> <bouncycastle.version>1.78</bouncycastle.version> </properties>  
       
      <dependencies>  
      <dependency>  
      <groupId>org.springframework.boot</groupId>  
      <artifactId>spring-boot-starter-web</artifactId>  
      </dependency>  
      <dependency>  
      <groupId>org.springframework.boot</groupId>  
      <artifactId>spring-boot-starter-validation</artifactId>  
      </dependency>  
       
      <dependency>  
      <groupId>com.google.cloud</groupId>  
      <artifactId>google-cloud-privateca</artifactId>  
      </dependency>  
       
      <dependency>  
      <groupId>org.bouncycastle</groupId>  
      <artifactId>bcpkix-jdk18on</artifactId>  
      <version>${bouncycastle.version}</version>  
      </dependency>  
      <dependency>  
      <groupId>org.bouncycastle</groupId>  
      <artifactId>bcprov-jdk18on</artifactId>  
      <version>${bouncycastle.version}</version>  
      </dependency>  
       
      <dependency>  
      <groupId>org.springframework.boot</groupId>  
      <artifactId>spring-boot-starter-actuator</artifactId>  
      </dependency>  
       
      <dependency>  
      <groupId>org.projectlombok</groupId>  
      <artifactId>lombok</artifactId>  
      <optional>true</optional>  
      </dependency>  
       
      <dependency>  
      <groupId>org.springframework.boot</groupId>  
      <artifactId>spring-boot-starter-test</artifactId>  
      <scope>test</scope>  
      </dependency>  
      </dependencies>  
       
      <dependencyManagement>  
      <dependencies>  
      <dependency>  
      <groupId>com.google.cloud</groupId>  
      <artifactId>google-cloud-bom</artifactId>  
      <version>${google-cloud.version}</version>  
      <type>pom</type>  
      <scope>import</scope>  
      </dependency>  
      </dependencies>  
      </dependencyManagement>  
       
      <build>  
      <plugins>  
      <plugin>  
      <groupId>org.springframework.boot</groupId>  
      <artifactId>spring-boot-maven-plugin</artifactId>  
      <configuration>  
      <excludes>  
      <exclude>  
      <groupId>org.projectlombok</groupId>  
      <artifactId>lombok</artifactId>  
      </exclude>  
      </excludes>  
      </configuration>  
      </plugin>  
      </plugins>  
      </build>  
       
     </project>
   * **Configuration (src/main/resources/application.yml):**  
     YAML  
     # Spring Boot application properties  
     spring:  
      application:  
      name: certificate-issuance-service  
       
     # Server port for the microservice  
     server:  
      port: 8080  
       
     # GCP CAS Configuration (provided by your central security/PKI team)  
     gcp:  
      cas:  
      project-id: "central-cas-project-id" # REPLACE WITH YOUR CENTRAL CAS PROJECT ID  
      location: "asia-south1" # REPLACE WITH YOUR CAS POOL REGION  
      ca-pool-id: "b2b-mtls-ca-pool" # REPLACE WITH YOUR B2B CA POOL ID  
      certificate-template-path: "projects/central-cas-project-id/locations/asia-south1/certificateTemplates/B2B\_MTLS\_TEMPLATE" # REPLACE WITH YOUR FULL TEMPLATE PATH  
       
     # Logging Configuration  
     logging:  
      level:  
      root: INFO  
      com.example.cas: DEBUG # Set to DEBUG for more detailed logging from your app  
      com.google.cloud.security.privateca: INFO # Set to DEBUG for more verbose CAS client logging
   * **Main Application Class (src/main/java/com/example/cas/CertificateIssuanceServiceApplication.java):**  
     Java  
     package com.example.cas;  
       
     import org.springframework.boot.SpringApplication;  
     import org.springframework.boot.autoconfigure.SpringBootApplication;  
     import org.springframework.boot.context.properties.ConfigurationPropertiesScan;  
       
     @SpringBootApplication  
     @ConfigurationPropertiesScan  
     public class CertificateIssuanceServiceApplication {  
       
      public static void main(String[] args) {  
      SpringApplication.run(CertificateIssuanceServiceApplication.class, args);  
      }  
       
     }
   * **External Contracts (DTOs for Apigee-facing API) (src/main/java/com/example/cas/api/ExternalCertificateRequest.java and ExternalCertificateResponse.java):**  
     Java  
     // ExternalCertificateRequest.java  
     package com.example.cas.api;  
       
     import jakarta.validation.constraints.Max;  
     import jakarta.validation.constraints.Min;  
     import jakarta.validation.constraints.NotBlank;  
     import lombok.Data;  
       
     @Data  
     public class ExternalCertificateRequest {  
      @NotBlank(message = "CSR cannot be blank")  
      private String csr;  
       
      @Min(value = 1, message = "Validity days must be at least 1")  
      @Max(value = 365, message = "Validity days cannot exceed 365 for security reasons") // API's own max limit  
      private Integer validityDays;  
     }  
       
     Java  
     // ExternalCertificateResponse.java  
     package com.example.cas.api;  
       
     import lombok.Data;  
     import java.util.List;  
       
     @Data  
     public class ExternalCertificateResponse {  
      private String issuedCertificatePem;  
      private List<String> caChainPem;  
      private String status;  
      private String message;  
       
      public ExternalCertificateResponse(String issuedCertificatePem, List<String> caChainPem, String status, String message) {  
      this.issuedCertificatePem = issuedCertificatePem;  
      this.caChainPem = caChainPem;  
      this.status = status;  
      this.message = message;  
      }  
     }
   * **Utility: CSR Parser (src/main/java/com/example/cas/util/CsrParser.java):**  
     Java  
     package com.example.cas.util;  
       
     import org.bouncycastle.asn1.x500.X500Name;  
     import org.bouncycastle.asn1.x500.style.BCStyle;  
     import org.bouncycastle.asn1.x509.Extension;  
     import org.bouncycastle.asn1.x509.Extensions;  
     import org.bouncycastle.asn1.x509.GeneralName;  
     import org.bouncycastle.asn1.x509.GeneralNames;  
     import org.bouncycastle.openssl.PEMParser;  
     import org.bouncycastle.pkcs.PKCS10CertificationRequest;  
     import org.bouncycastle.util.io.pem.PemObject;  
     import org.slf4j.Logger;  
     import org.slf4j.LoggerFactory;  
       
     import java.io.IOException;  
     import java.io.StringReader;  
     import java.util.ArrayList;  
     import java.util.Collections;  
     import java.util.List;  
       
     public class CsrParser {  
       
      private static final Logger logger = LoggerFactory.getLogger(CsrParser.class);  
       
      public static CsrDetails parseCsr(String pemCsr) throws IllegalArgumentException {  
      if (pemCsr == null || pemCsr.trim().isEmpty()) {  
      throw new IllegalArgumentException("CSR cannot be null or empty.");  
      }  
       
      try (StringReader stringReader = new StringReader(pemCsr);  
      PEMParser pemParser = new PEMParser(stringReader)) {  
       
      Object parsedObject = pemParser.readObject();  
      if (parsedObject == null) {  
      throw new IllegalArgumentException("Could not parse PEM object from CSR. Is it valid PEM?");  
      }  
       
      PKCS10CertificationRequest csr = null;  
      if (parsedObject instanceof PKCS10CertificationRequest) {  
      csr = (PKCS10CertificationRequest) parsedObject;  
      } else if (parsedObject instanceof PemObject) {  
      PemObject pemObject = (PemObject) parsedObject;  
      if ("CERTIFICATE REQUEST".equals(pemObject.getType())) {  
      csr = new PKCS10CertificationRequest(pemObject.getContent());  
      }  
      }  
       
      if (csr == null) {  
      throw new IllegalArgumentException("Parsed PEM object is not a Certificate Signing Request (PKCS10). Actual type: " + parsedObject.getClass().getName());  
      }  
       
      X500Name subject = csr.getSubject();  
      String commonName = getCommonNameFromX500Name(subject);  
      List<String> dnsNames = new ArrayList<>();  
      List<String> uriNames = new ArrayList<>();  
      List<String> ipAddresses = new ArrayList<>();  
       
      Extensions extensions = csr.getRequestedExtensions();  
       
      if (extensions != null) {  
      Extension sanExtension = extensions.getExtension(Extension.subjectAlternativeName);  
      if (sanExtension != null) {  
      GeneralNames generalNames = GeneralNames.getInstance(sanExtension.getParsedValue());  
      for (GeneralName generalName : generalNames.getNames()) {  
      switch (generalName.getTagNo()) {  
      case GeneralName.dNSName:  
      dnsNames.add(generalName.getName().toString());  
      break;  
      case GeneralName.iPAddress:  
      ipAddresses.add(generalName.getName().toString());  
      break;  
      case GeneralName.uniformResourceIdentifier:  
      uriNames.add(generalName.getName().toString());  
      break;  
      }  
      }  
      }  
      }  
       
      return new CsrDetails(commonName, dnsNames, uriNames, ipAddresses);  
       
      } catch (IOException e) {  
      logger.error("Error reading PEM CSR: {}", e.getMessage(), e);  
      throw new IllegalArgumentException("Error reading CSR: " + e.getMessage(), e);  
      } catch (Exception e) {  
      logger.error("Unexpected error during CSR parsing: {}", e.getMessage(), e);  
      throw new IllegalArgumentException("Failed to parse CSR: " + e.getMessage(), e);  
      }  
      }  
       
      private static String getCommonNameFromX500Name(X500Name x500Name) {  
      org.bouncycastle.asn1.x500.RDN[] rdns = x500Name.getRDNs(BCStyle.CN);  
      if (rdns.length > 0) {  
      return rdns[0].getFirst().getValue().toString();  
      }  
      return null;  
      }  
       
      @lombok.Data  
      @lombok.AllArgsConstructor  
      public static class CsrDetails {  
      private String commonName;  
      private List<String> dnsNames;  
      private List<String> uriNames;  
      private List<String> ipAddresses;  
       
      public List<String> getAllSanNames() {  
      List<String> all = new ArrayList<>();  
      if (dnsNames != null) all.addAll(dnsNames);  
      if (uriNames != null) all.addAll(uriNames);  
      if (ipAddresses != null) all.addAll(ipAddresses);  
      return all;  
      }  
      }  
     }
   * **Custom Exception (src/main/java/com/example/cas/service/CertificateIssuanceException.java):**  
     Java  
     package com.example.cas.service;  
       
     public class CertificateIssuanceException extends Exception {  
      public CertificateIssuanceException(String message) {  
      super(message);  
      }  
       
      public CertificateIssuanceException(String message, Throwable cause) {  
      super(message, cause);  
      }  
     }
   * **Service Layer (src/main/java/com/example/cas/service/CertificateService.java):**  
     Java  
     package com.example.cas.service;  
       
     import com.google.cloud.security.privateca.v1.CaPoolName;  
     import com.google.cloud.security.privateca.v1.Certificate;  
     import com.google.cloud.security.privateca.v1.CertificateAuthorityServiceClient;  
     import com.google.protobuf.Duration;  
     import org.slf4j.Logger;  
     import org.slf4j.LoggerFactory;  
     import org.springframework.beans.factory.annotation.Value;  
     import org.springframework.stereotype.Service;  
       
     import java.io.IOException;  
       
     @Service  
     public class CertificateService {  
       
      private static final Logger logger = LoggerFactory.getLogger(CertificateService.class);  
       
      @Value("${gcp.cas.project-id}")  
      private String casProjectId;  
       
      @Value("${gcp.cas.location}")  
      private String casLocation;  
       
      @Value("${gcp.cas.ca-pool-id}")  
      private String caPoolId;  
       
      @Value("${gcp.cas.certificate-template-path}")  
      private String certificateTemplatePath;  
       
      public Certificate requestClientCertificate(String pemCsr, int validityDays) throws CertificateIssuanceException {  
      CertificateAuthorityServiceClient client = null;  
      try {  
      client = CertificateAuthorityServiceClient.create();  
      CaPoolName parent = CaPoolName.of(casProjectId, casLocation, caPoolId);  
      Duration lifetime = Duration.newBuilder()  
      .setSeconds((long) validityDays \* 24 \* 60 \* 60)  
      .build();  
       
      Certificate.Builder certificateBuilder = Certificate.newBuilder()  
      .setPemCsr(pemCsr)  
      .setLifetime(lifetime)  
      .setCertificateTemplate(certificateTemplatePath);  
       
      logger.info("Submitting certificate request to CAS Pool: {} using template: {}", parent.toString(), certificateTemplatePath);  
       
      Certificate issuedCertificate = client.createCertificate(parent, certificateBuilder.build());  
       
      logger.info("Successfully issued certificate with serial number: {}", issuedCertificate.getSerialNumber());  
      return issuedCertificate;  
       
      } catch (IOException e) {  
      logger.error("Failed to create CertificateAuthorityServiceClient: {}", e.getMessage(), e);  
      throw new CertificateIssuanceException("Could not initialize CAS client due to I/O error.", e);  
      } catch (com.google.api.gax.rpc.ApiException e) {  
      logger.error("GCP CAS API error during certificate request: {}. Details: {}", e.getStatusCode().getCode(), e.getMessage(), e);  
      throw new CertificateIssuanceException("GCP CAS API error: " + e.getMessage(), e);  
      } catch (Exception e) {  
      logger.error("An unexpected error occurred during certificate request: {}", e.getMessage(), e);  
      throw new CertificateIssuanceException("An unexpected error occurred: " + e.getMessage(), e);  
      } finally {  
      if (client != null) {  
      try {  
      client.close();  
      } catch (Exception e) {  
      logger.warn("Failed to close CertificateAuthorityServiceClient: {}", e.getMessage());  
      }  
      }  
      }  
      }  
     }
   * **REST Controller (src/main/java/com/example/cas/controller/CertificateController.java):**  
     Java  
     package com.example.cas.controller;  
       
     import com.example.cas.api.ExternalCertificateRequest;  
     import com.example.cas.api.ExternalCertificateResponse;  
     import com.example.cas.service.CertificateIssuanceException;  
     import com.example.cas.service.CertificateService;  
     import com.example.cas.util.CsrParser;  
     import com.example.cas.util.CsrParser.CsrDetails;  
     import com.google.cloud.security.privateca.v1.Certificate;  
     import jakarta.validation.Valid;  
     import org.slf4j.Logger;  
     import org.slf4j.LoggerFactory;  
     import org.springframework.http.HttpStatus;  
     import org.springframework.http.ResponseEntity;  
     import org.springframework.web.bind.annotation.PostMapping;  
     import org.springframework.web.bind.annotation.RequestBody;  
     import org.springframework.web.bind.annotation.RequestMapping;  
     import org.springframework.web.bind.annotation.RestController;  
       
     import java.util.Collections;  
     import java.util.List;  
     import java.util.regex.Pattern;  
       
     @RestController  
     @RequestMapping("/certificates")  
     public class CertificateController {  
       
      private static final Logger logger = LoggerFactory.getLogger(CertificateController.class);  
       
      // --- Configuration for Template-Based Validation (Should reflect CAS Template) ---  
      // These constants define the validation rules \*enforced by your API\* based on shared knowledge  
      // with the central CAS team's template configuration.  
      // Adjust these patterns and limits as per your actual CAS template definition.  
      private static final int DEFAULT\_VALIDITY\_DAYS = 90;  
      private static final int MAX\_REQUESTED\_VALIDITY\_DAYS = 365; // Max validity your API accepts (e.g., 1 year)  
       
      private static final Pattern COMMON\_NAME\_PATTERN = Pattern.compile("^[a-z0-9](?:[a-z0-9-]{0,61}[a-z0-9])?\\.(?:[a-z0-9](?:[a-z0-9-]{0,61}[a-z0-9])?\\.)+[a-z0-9][a-z0-9-]{0,61}[a-z0-9]$");  
      private static final int MIN\_CN\_LENGTH = 5;  
      private static final int MAX\_CN\_LENGTH = 100;  
       
      private static final boolean ALLOW\_DNS\_SANS = true;  
      private static final boolean ALLOW\_URI\_SANS = false;  
      private static final boolean ALLOW\_IP\_SANS = false;  
       
      private static final int MAX\_SANS\_ALLOWED = 5;  
       
      private static final List<String> ALLOWED\_DNS\_SAN\_DOMAINS = List.of(".yourorg.com", ".partnerorg.com"); // REPLACE with your actual allowed domains  
       
       
      private final CertificateService certificateService;  
       
      public CertificateController(CertificateService certificateService) {  
      this.certificateService = certificateService;  
      }  
       
      @PostMapping("/request")  
      public ResponseEntity<ExternalCertificateResponse> requestCertificate(  
      @Valid @RequestBody ExternalCertificateRequest externalRequest) {  
       
      logger.info("Received certificate request from external client. Requested validity: {} days. CSR length: {}",  
      externalRequest.getValidityDays(), externalRequest.getCSR().length());  
       
      try {  
      // 1. Validate Input & Parse CSR (from external contract)  
      CsrDetails csrDetails = CsrParser.parseCsr(externalRequest.getCSR());  
      logger.debug("Parsed CSR: CN={}, DNS SANs={}, URI SANs={}, IP SANs={}",  
      csrDetails.getCommonName(), csrDetails.getDnsNames(),  
      csrDetails.getUriNames(), csrDetails.getIpAddresses());  
       
      // 2. Microservice-level Validation against Template Rules  
      validateCsrAgainstTemplateRules(csrDetails);  
       
      // 3. Validate Requested Validity (API's local limit)  
      int validity = externalRequest.getValidityDays() != null ?  
      Math.min(externalRequest.getValidityDays(), MAX\_REQUESTED\_VALIDITY\_DAYS) :  
      DEFAULT\_VALIDITY\_DAYS;  
      if (validity < 1 || validity > MAX\_REQUESTED\_VALIDITY\_DAYS) {  
      throw new IllegalArgumentException("Requested validity days (" + externalRequest.getValidityDays() +  
      ") must be between 1 and " + MAX\_REQUESTED\_VALIDITY\_DAYS + ".");  
      }  
       
      // 4. Call the Internal Service Layer  
      Certificate issuedCert = certificateService.requestClientCertificate(externalRequest.getCSR(), validity);  
       
      // 5. Prepare External Response (mapping from internal CAS object to external contract)  
      List<String> pemCertificateChain = issuedCert.getPemCertificateChainList();  
      List<String> caChain = Collections.emptyList();  
      String issuedClientCertPem = null;  
       
      if (pemCertificateChain != null && !pemCertificateChain.isEmpty()) {  
      issuedClientCertPem = pemCertificateChain.get(0);  
      if (pemCertificateChain.size() > 1) {  
      caChain = pemCertificateChain.subList(1, pemCertificateChain.size());  
      }  
      } else {  
      logger.warn("Issued certificate or chain is unexpectedly empty for serial number: {}", issuedCert.getSerialNumber());  
      }  
       
      ExternalCertificateResponse response = new ExternalCertificateResponse(  
      issuedClientCertPem,  
      caChain,  
      "success",  
      "Certificate issued successfully with serial: " + issuedCert.getSerialNumber());  
       
      logger.info("Certificate issued successfully for request. Serial: {}", issuedCert.getSerialNumber());  
      return ResponseEntity.ok(response);  
       
      } catch (IllegalArgumentException e) {  
      logger.error("Invalid request payload or CSR content validation failed: {}", e.getMessage());  
      return ResponseEntity.badRequest()  
      .body(new ExternalCertificateResponse(null, null, "error", "Invalid request: " + e.getMessage()));  
      } catch (CertificateIssuanceException e) {  
      logger.error("Certificate issuance failed at CAS layer: {}", e.getMessage());  
      return ResponseEntity.status(HttpStatus.INTERNAL\_SERVER\_ERROR)  
      .body(new ExternalCertificateResponse(null, null, "error", "Certificate issuance failed: " + e.getMessage()));  
      } catch (Exception e) {  
      logger.error("An unexpected error occurred during certificate request: {}", e.getMessage(), e);  
      return ResponseEntity.status(HttpStatus.INTERNAL\_SERVER\_ERROR)  
      .body(new ExternalCertificateResponse(null, null, "error", "An unexpected internal error occurred. Please try again later."));  
      }  
      }  
       
      private void validateCsrAgainstTemplateRules(CsrDetails csrDetails) throws IllegalArgumentException {  
      // 1. Validate Common Name (CN)  
      if (csrDetails.getCommonName() == null || csrDetails.getCommonName().isEmpty()) {  
      throw new IllegalArgumentException("CSR must contain a Common Name (CN).");  
      }  
      if (csrDetails.getCommonName().length() < MIN\_CN\_LENGTH || csrDetails.getCommonName().length() > MAX\_CN\_LENGTH) {  
      throw new IllegalArgumentException(String.format("Common Name length (%d) must be between %d and %d characters.",  
      csrDetails.getCommonName().length(), MIN\_CN\_LENGTH, MAX\_CN\_LENGTH));  
      }  
      if (!COMMON\_NAME\_PATTERN.matcher(csrDetails.getCommonName()).matches()) {  
      throw new IllegalArgumentException("Common Name '" + csrDetails.getCommonName() + "' does not match the required pattern.");  
      }  
       
      // 2. Validate Subject Alternative Names (SANs) presence, types, and count  
      int totalSans = csrDetails.getDnsNames().size() + csrDetails.getUriNames().size() + csrDetails.getIpAddresses().size();  
      if (totalSans > MAX\_SANS\_ALLOWED) {  
      throw new IllegalArgumentException(String.format("CSR contains too many SANs (%d). Maximum allowed is %d.", totalSans, MAX\_SANS\_ALLOWED));  
      }  
       
      if (!ALLOW\_DNS\_SANS && !csrDetails.getDnsNames().isEmpty()) {  
      throw new IllegalArgumentException("DNS Subject Alternative Names are not allowed per policy.");  
      }  
      if (!ALLOW\_URI\_SANS && !csrDetails.getUriNames().isEmpty()) {  
      throw new IllegalArgumentException("URI Subject Alternative Names are not allowed per policy.");  
      }  
      if (!ALLOW\_IP\_SANS && !csrDetails.getIpAddresses().isEmpty()) {  
      throw new IllegalArgumentException("IP Address Subject Alternative Names are not allowed per policy.");  
      }  
       
      // 3. Specific validation for allowed DNS SAN domains (if DNS SANs are allowed)  
      if (ALLOW\_DNS\_SANS) {  
      for (String dns : csrDetails.getDnsNames()) {  
      boolean domainAllowed = false;  
      for (String allowedDomain : ALLOWED\_DNS\_SAN\_DOMAINS) {  
      if (dns.endsWith(allowedDomain)) {  
      domainAllowed = true;  
      break;  
      }  
      }  
      if (!domainAllowed) {  
      throw new IllegalArgumentException("DNS SAN '" + dns + "' is not from an allowed domain. Allowed domains: " + ALLOWED\_DNS\_SAN\_DOMAINS);  
      }  
      }  
      }  
      logger.debug("CSR details successfully validated against template rules.");  
      }  
     }

### 3. Deployment and Execution

1. **Build your Spring Boot Microservice:**  
   Bash  
   ./mvnw clean install
2. **Containerize and Push Image:**  
   Bash  
   gcloud builds submit --tag gcr.io/your-project-id/certificate-issuance-service:latest
3. **Deploy to GCP (Example: Cloud Run):**  
   Bash  
   gcloud run deploy certificate-issuance-service \  
    --image gcr.io/your-project-id/certificate-issuance-service:latest \  
    --platform managed \  
    --region asia-south1 \  
    --allow-unauthenticated \  
    --service-account my-app-cas-requester@your-project-id.iam.gserviceaccount.com \  
    --vpc-connector projects/YOUR\_HOST\_PROJECT\_ID/ID\_OF\_VPC\_CONNECTOR \  
    --ingress internal-and-cloud-load-balancing  
   * **--service-account**: Essential for Application Default Credentials (ADC) to pick up your service account's permissions.
   * **--vpc-connector**: Required for private communication to the CAS API in the central project (via Shared VPC). Create this connector in your service project, linking it to a subnet in your Shared VPC host project.
   * **--allow-unauthenticated**: This is because Apigee will handle the initial authentication. If Apigee accesses Cloud Run via internal ingress, you might restrict this further.
4. **Apigee Configuration:**
   * **Create API Proxy:** Define a new API Proxy in Apigee (e.g., cert-request-proxy) that will serve as the public-facing endpoint for your certificate issuance API.
   * **Target Endpoint:** Configure the proxy's target endpoint to point to the *internal URL* of your deployed Spring Boot microservice (e.g., the Cloud Run service URL). Ensure Apigee has network connectivity to this internal endpoint (via VPC Peering for Apigee X, or Serverless VPC Access for Apigee Hybrid/Cloud Run).
   * **Attach Policies:**
     + **Authentication (Northbound):** Add policies like **Verify API Key**, **OAuthV2**, or configure **mTLS** client authentication on the Apigee Virtual Host.
     + **Traffic Management:** Implement **Quota** and **Spike Arrest** policies to protect your backend from excessive requests.
     + **Threat Protection:** Apply **JSON Threat Protection** to validate the incoming CSR payload.
     + **CORS Policy:** If your clients are web-based.
     + **Message Logging/Analytics:** Configure logging to capture API traffic details.
     + **Service Callout (Optional but Recommended):** Apigee can enrich the request before sending it to your Spring Boot service (e.g., inject partner ID, internal API key/token for your microservice to validate that the call originated from Apigee).

### 4. External Client Usage

1. **Generate Private Key and CSR:**
   * The external client generates its own strong private key locally (e.g., RSA 2048/4096, ECC).
   * It then creates a CSR using this private key, ensuring the Subject (Common Name, Organization, etc.) and Subject Alternative Names (DNS, URI, IP) strictly adhere to the patterns and constraints defined in your B2B\_MTLS\_TEMPLATE and the validation rules in your Spring Boot microservice. <!-- end list -->

Bash  
# Example using OpenSSL  
openssl genrsa -out client.key 2048  
  
# Create a CSR - IMPORTANT: Adjust CN and SANs to match YOUR TEMPLATE and validation rules!  
openssl req -new -key client.key -out client.csr \  
 -subj "/CN=my-partner-app.prod.yourorg.com/O=PartnerCompany/OU=IT/C=IN" \  
 -addext "subjectAltName=DNS:webapp.partner.yourorg.com,DNS:api.partner.yourorg.com"

1. **Call the Apigee API:**
   * The client makes an HTTP POST request to the Apigee API Gateway's public endpoint.
   * It sends the PEM-encoded client.csr within the JSON request body, along with the optional validityDays.
   * It includes any required Apigee authentication headers (e.g., x-api-key, Authorization: Bearer ...). <!-- end list -->

Bash  
CSR\_CONTENT=$(cat client.csr)  
APIGEE\_AUTH\_HEADER="x-api-key: YOUR\_APIGEE\_API\_KEY" # Or "Authorization: Bearer YOUR\_OAUTH\_TOKEN"  
  
curl -X POST "https://api.yourcompany.com/v1/certificates/request" \  
 -H "Content-Type: application/json" \  
 -H "${APIGEE\_AUTH\_HEADER}" \  
 -d "{  
 \"csr\": \"${CSR\_CONTENT}\",  
 \"validityDays\": 90  
 }"

1. **Process Response:** The client receives a JSON response from Apigee containing the issuedCertificatePem (their new certificate) and the caChainPem (the chain of trust for their truststore).
2. **Configure mTLS:**
   * The client combines its local client.key (private key) and the received issuedCertificatePem into a PKCS#12 (.p12) or JKS (.jks) keystore. This keystore represents the client's identity.
   * The client imports the caChainPem into a separate truststore. This truststore allows the client to verify the certificates of servers it wants to establish mTLS with (e.g., your APIs protected by mTLS).
   * Finally, the client configures its application's HTTP client or networking stack to use this keystore and truststore for outgoing mTLS connections.

### 5. Security Best Practices Summary

* **Principle of Least Privilege:**
  + my-app-cas-requester service account is strictly limited to requesting certificates from the specific CA Pool and using the specific template. It has no administrative access to CAS.
  + Apigee is configured with only necessary permissions and policies.
* **Private Key Ownership:** Clients always generate and retain control of their private keys. Your Spring Boot service and CAS never handle private keys.
* **Application Default Credentials (ADC):** Secures and simplifies authentication for your Spring Boot microservice to GCP APIs, eliminating the need for sensitive key files in deployments.
* **Certificate Templates:** Central governance mechanism to enforce consistent and secure certificate policies (key usages, validity, subject attributes, is\_ca: false).
* **Microservice-level Validation:** Provides early feedback to clients, reduces unnecessary CAS calls, and adds an extra layer of security and robustness by validating CSRs against template rules.
* **Shared VPC:** Ensures all internal GCP communications (Spring Boot to CAS, Apigee to Spring Boot) occur over Google's secure, private network, minimizing public exposure.
* **Apigee API Gateway:**
  + Acts as a robust security perimeter, protecting your backend microservice.
  + Handles external authentication (API Keys, OAuth, mTLS), traffic management (rate limiting), and threat protection (JSON threat, content validation).
  + Enables standardized API exposure and management.
* **Comprehensive Audit Logging:** All CAS operations are logged in Cloud Audit Logs in the Central CAS Project, providing an immutable audit trail. Comprehensive logging should also be enabled for your Spring Boot application and Apigee.
* **Certificate Rotation & Revocation:** Plan for automated certificate renewal processes for clients. Implement API endpoints/processes for timely certificate revocation through CAS when a certificate is compromised or no longer needed.

This complete solution provides a highly secure, automated, and scalable framework for managing mTLS client certificates for your B2B integrations within an enterprise GCP environment.

**Sources**

1. <https://cloud.google.com/certificate-authority-service/docs/requesting-certificates?hl=id>