Mutual TLS authentication

The network traffic initiated by Dialogflow for webhook requests is sent on a public network. To ensure that traffic is both secure and trusted in both directions, Dialogflow optionally supports Mutual TLS authentication (mTLS) (https://en.wikipedia.org/wiki/Mutual_authentication). During Dialogflow's standard TLS handshake (https://hpbn.co/transport-layer-security-tls/#tls-handshake), your webhook server presents a certificate that can be validated by Dialogflow, either by following the Certificate Authority chain (https://hpbn.co/transport-layer-security-tls/#chain-of-trust-and-certificate-authorities) or by comparing the certificate to a Custom CA certificate (/dialogflow/cx/docs/concept/custom-ca). By enabling mTLS on your webhook server, it will be able to authenticate the Google certificate (https://pki.goog/roots.pem) presented by Dialogflow to your webhook server for validation, completing the establishment of mutual trust.

Requesting mTLS

To request mTLS:

1. Prepare your webhook HTTPS server to request the client certificate during the TLS handshake.
2. Your webhook server should verify the client certificate upon receiving it.
3. Install a certificate chain for your webhook server, which can be mutually trusted by both client and server. Applications connecting to Google services should trust all the Certificate Authorities listed by Google Trust Services (https://pki.goog/faq/#faq-27). You can download root certs from: https://pki.goog/ (https://pki.goog/).

Sample call to a webhook server using mTLS

This example uses the agent shown in the quickstart with a webhook (/dialogflow/cx/docs/concept/webhook) server running `openssl` (https://www.openssl.org/docs/manmaster/man1/openssl.html).

1. Sample setup

a. A Dialogflow ES agent that greets the end user and queries a webhook pointing to a standalone web server.

b. A private key for TLS communication in a file named **`key.pem`**.

c. A certificate chain signed by a publicly-trusted CA (Certificate Authority) (/load-balancing/docs/ssl-certificates/self-managed-certs#use_a_publicly-trusted_ca) in a file named **`fullchain.pem`**.

2. Execute the **`openssl s_server`** (https://www.openssl.org/docs/manmaster/man1/openssl-s_server.html) program in the server machine.

```
sudo openssl s_server -key key.pem -cert fullchain.pem -accept 443 -verify
```

3. A request is sent to the agent from a client machine. For this example, the request is "Hi". This request can be sent using the Dialogflow Console, or through an API call.

4. Output of **`openssl s_server`** (https://www.openssl.org/docs/manmaster/man1/openssl-s_server.html) in the server machine.

```
verify depth is 1
Using default temp DH parameters
ACCEPT
depth=2 C = US, O = Google Trust Services LLC, CN = GTS Root R1
verify return:1
depth=1 C = US, O = Google Trust Services LLC, CN = GTS CA 1D4
verify return:1
depth=0 CN = *.dialogflow.com
verify return:1
-----BEGIN SSL SESSION PARAMETERS-----
MII...
-----END SSL SESSION PARAMETERS-----
Client certificate
-----BEGIN CERTIFICATE-----
MII...
-----END CERTIFICATE-----
subject=CN = *.dialogflow.com
```

```
issuer=C = US, O = Google Trust Services LLC, CN = GTS CA 1D4

Shared ciphers:TLS_AES_128_GCM_SHA256:...
Signature Algorithms: ECDSA+SHA256:...
Shared Signature Algorithms: ECDSA+SHA256:...
Peer signing digest: SHA256
Peer signature type: RSA-PSS
Supported Elliptic Groups: 0x6A6A:...
Shared Elliptic groups: X25519:...
CIPHER is TLS_AES_128_GCM_SHA256
Secure Renegotiation IS NOT supported
POST /dialogflowFulfillment HTTP/1.1
authorization: Bearer ey...
content-type: application/json
Host: www.example.com
Content-Length: 1011
Connection: keep-alive
Accept: */*
User-Agent: Google-Dialogflow
Accept-Encoding: gzip, deflate, br

{
  "responseId": "96c0029a-149d-4f5d-b225-0b0bb0f0c8d9-afbcf665",
  "queryResult": {
    "queryText": "Hi",
    "action": "input.welcome",
    "parameters": {
    },
    "allRequiredParamsPresent": true,
    "outputContexts": [{
      "name": "projects/PROJECT-ID/agent/sessions/58ab33f3-b57a-aae9-fb23-8
      "parameters": {
        "no-input": 0.0,
        "no-match": 0.0
      }
    }],
    "intent": {
      "name": "projects/PROJECT-ID/agent/intents/399277d6-2ed7-4329-840d-8b
      "displayName": "Default Welcome Intent"
    },
    "intentDetectionConfidence": 1.0,
    "languageCode": "en",
    "sentimentAnalysisResult": {
      "queryTextSentiment": {
        "score": 0.2,
```

```
        "magnitude": 0.2
      }
    }
  },
  "originalDetectIntentRequest": {
    "source": "DIALOGFLOW_CONSOLE",
    "payload": {
    }
  },
  "session": "projects/PROJECT-ID/agent/sessions/58ab33f3-b57a-aae9-fb23-83
}ERROR
shutting down SSL
CONNECTION CLOSED
```

Best Practice

To make sure that webhook requests are initiated from your own Dialogflow agents, you should verify the Bearer service identity token (/dialogflow/cx/docs/concept/webhook#id-token) from the request's Authorization header. Alternatively, you can verify a session parameter provided previously by an authentication server on your side.

Errors

If the client certificate validation fails (for example, the webhook server does not trust the client certificate), the TLS handshake fails and the session terminates.

Common error messages:

Error message	Explanation
Failed to verify client's certificate: x509: certificate signed by unknown authority	Dialogflow sends its client certificate to the external webhook, but the external webhook cannot verify it. This may be because the external webhook didn't install the CA chain correctly. All root CAs from Google should be trusted.

Previous
← Webhook for slot filling (/dialogflow/es/docs/fulfillment-webhook-slot-filling)

Next
Speech models (/dialogflow/es/docs/speech-models) →

Last updated 2025-04-17 UTC.