Transport Security (TLS)

Google Cloud

Edge Support for TLS

- TLS
  - Public and private cloud

Edge Support for TLS

- TLS
  - Public and private cloud
- One-way TLS
  - Client verifies server

apigee

Edge Support for TLS

- TLS
  - Public and private cloud
- One-way TLS
  - Client verifies server
- Two-way TLS
  - Mutual auth between client and server

What data is encrypted via TLS?

- Encrypted data
  - URL
  - Headers
  - Query params
  - HTTP verb
  - Payload

- Destination server & payload size known

Sensitive Data and TLS

- Data encrypted in motion
  - Not at rest

apigee

Sensitive Data and TLS

- Data encrypted in motion
  - Not at rest
- No passwords in query parameters

apigee

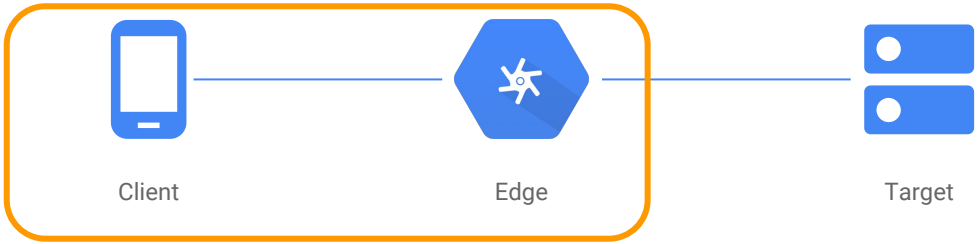
Sensitive Data and TLS

- Data encrypted in motion
  - Not at rest
- No passwords in query parameters
- Avoid sensitive info in URLs

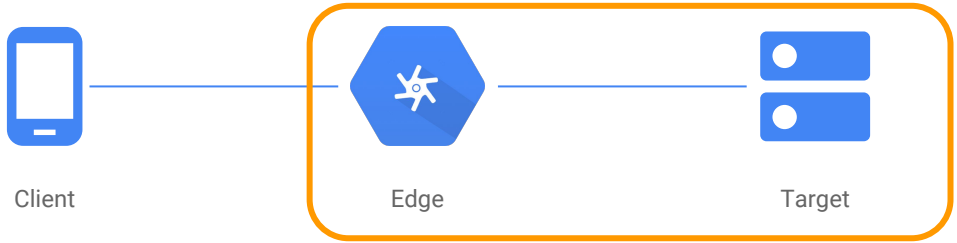
Sensitive Data and TLS

- Data encrypted in motion
  - Not at rest
- No passwords in query parameters
- Avoid sensitive info in URLs
- Use payload or headers

TLS & Edge



TLS & Edge



Edge Keystores and Truststores

- Keystore
  - Server certificate

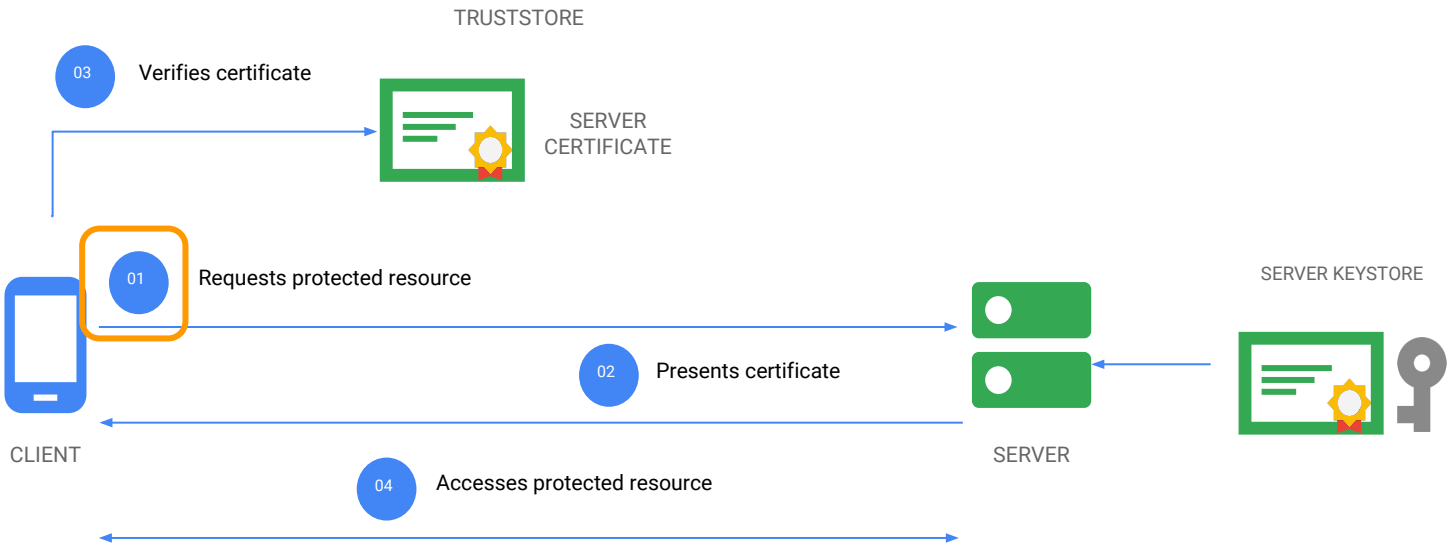
Edge Keystores and Truststores

- Keystore
  - Server certificate

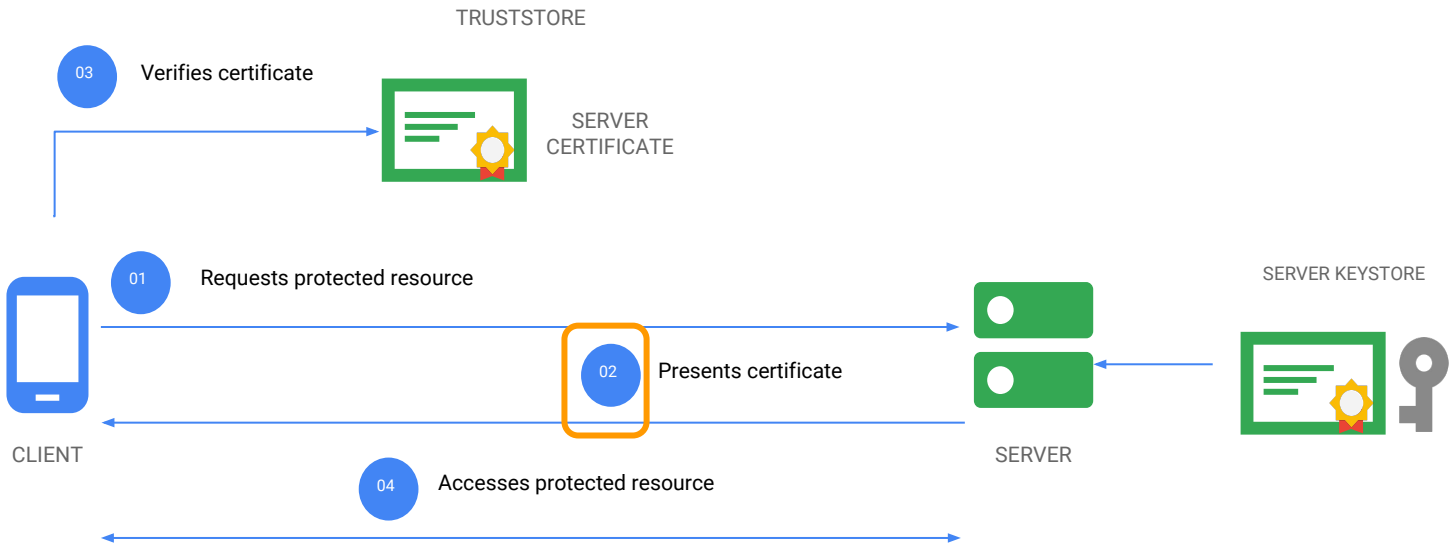
- Truststore
  - Valid client certificates

- Keystores and Truststores used for client and target communication

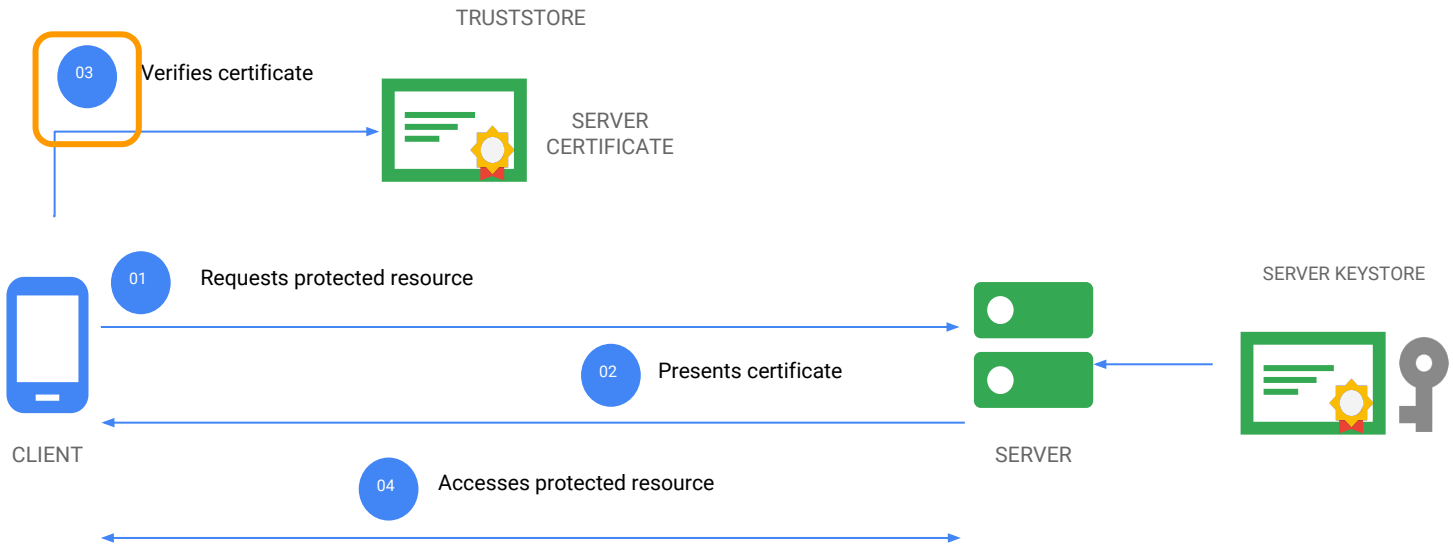
Transport Layer Security (one-way TLS)



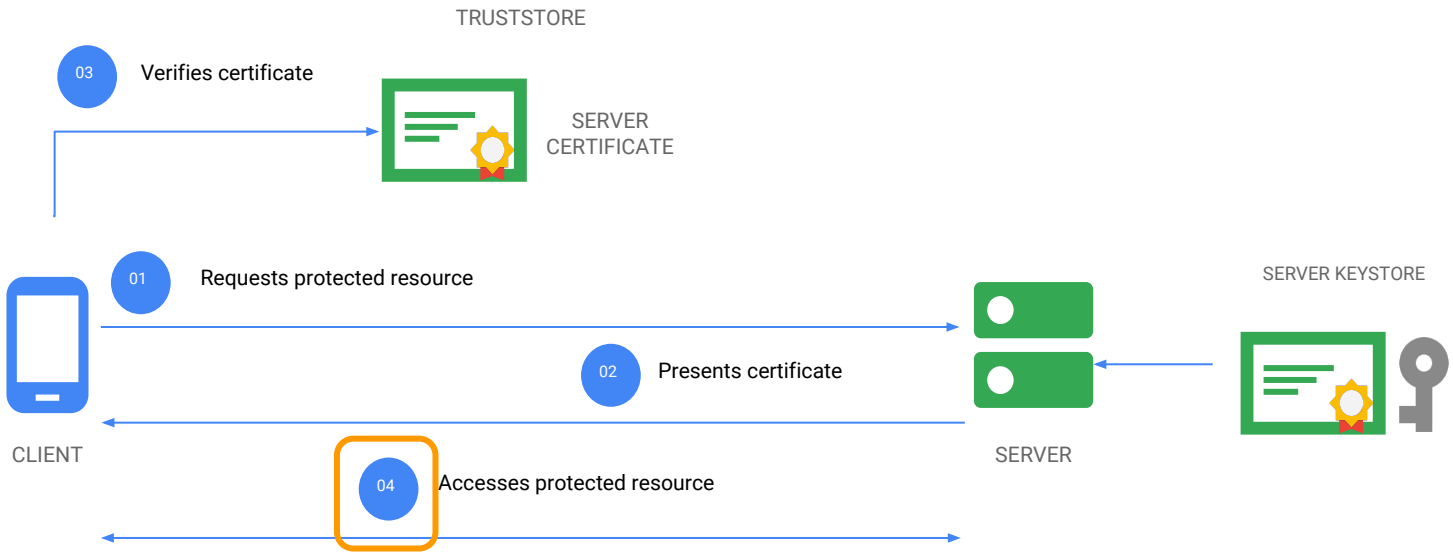
Transport Layer Security (one-way TLS)



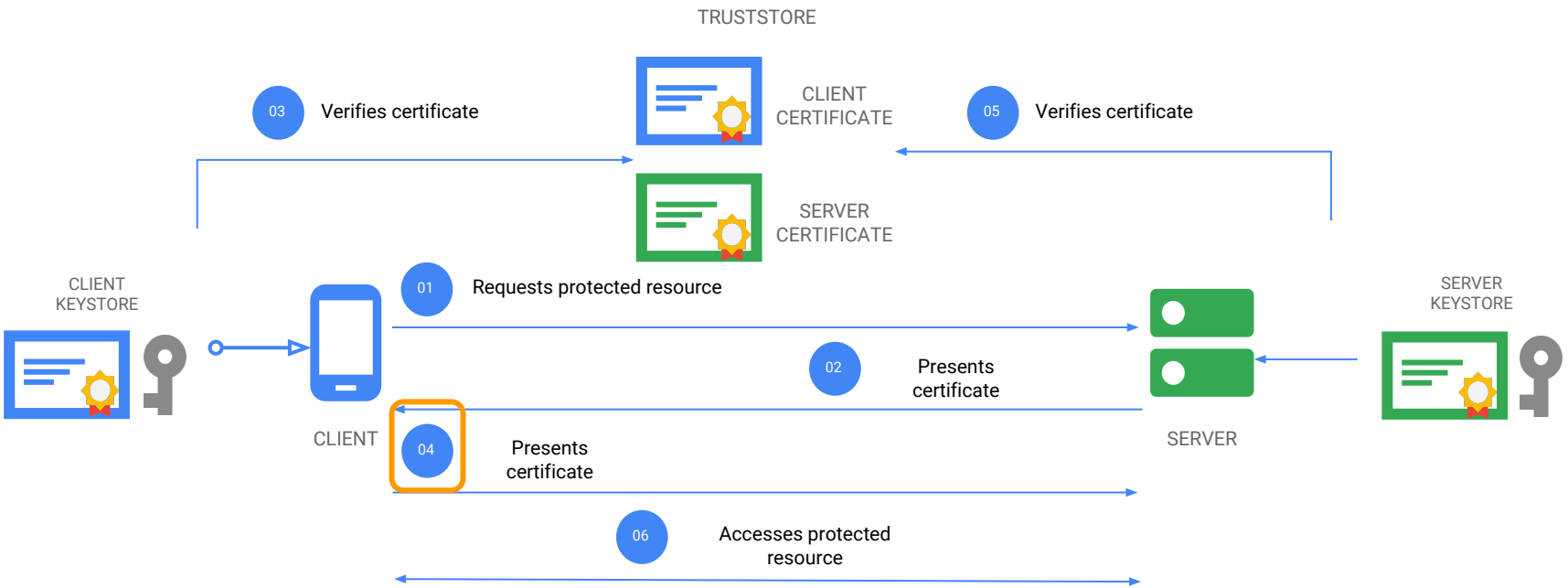
Transport Layer Security (one-way TLS)



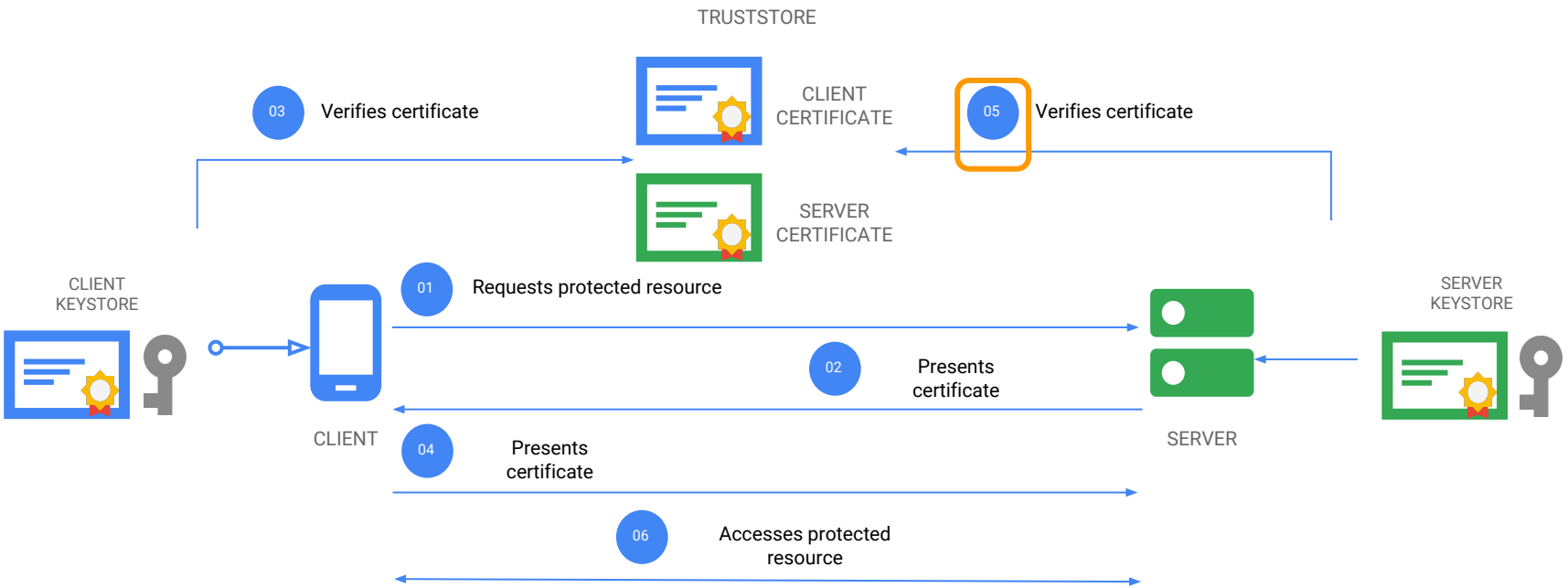
Transport Layer Security (one-way TLS)



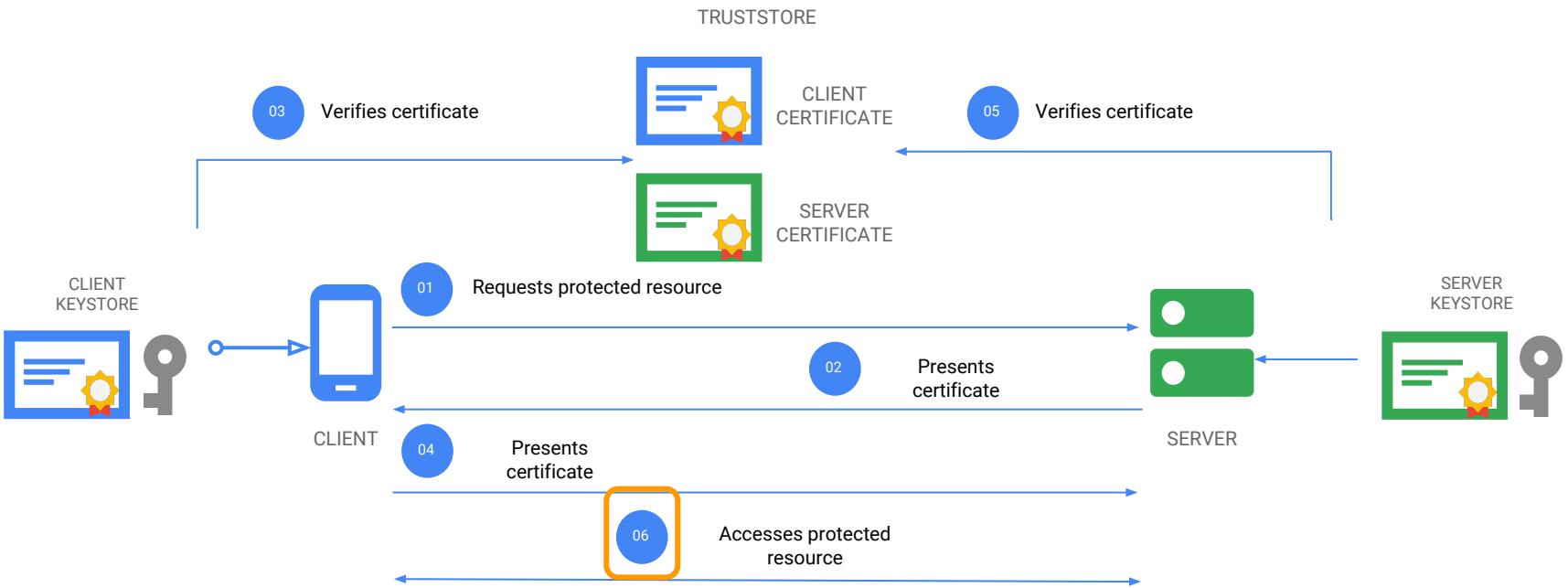
Transport Layer Security (two-way TLS)



Transport Layer Security (two-way TLS)



Transport Layer Security (two-way TLS)



Configuring 2-way TLS from client to Edge (Public Cloud)

01

Server keystore

Create a keystore and upload certificate and private key of the server

02

Truststore

If the client uses a self-signed certificate, or a certificate that is not signed by a trusted CA, create a truststore on Edge that contains the CA chain of the client certificate.

03

Virtualhost

Create a support ticket so the virtualhost is created with the suitable configuration.

```
<VirtualHost name="myTLSVHost">
  <HostAliases>
    <HostAlias>apiTLS.myCompany.com</HostAlias>
  </HostAliases>
  <Port>443</Port>
  <SSLInfo>
    <Enabled>true</Enabled>
    <ClientAuthEnabled>false</ClientAuthEnabled>
    <KeyStore>myTestKeystore</KeyStore>
    <KeyAlias>myKeyAlias</KeyAlias>
  </SSLInfo>
</VirtualHost>
```

apigee

Configuring 2-way TLS from client to Edge (Public Cloud)

01

Server keystore

Create a keystore and upload certificate and private key of the server

02

Truststore

If the client uses a self-signed certificate, or a certificate that is not signed by a trusted CA, create a truststore on Edge that contains the CA chain of the client certificate.

03

Virtualhost

Create a support ticket so the virtualhost is created with the suitable configuration.

```
<VirtualHost name="myTLSVHost">
  <HostAliases>
    <HostAlias>apiTLS.myCompany.com</HostAlias>
  </HostAliases>
  <Port>443</Port>
  <SSLInfo>
    <Enabled>true</Enabled>
    <ClientAuthEnabled>false</ClientAuthEnabled>
    <KeyStore>myTestKeystore</KeyStore>
    <KeyAlias>myKeyAlias</KeyAlias>
  </SSLInfo>
</VirtualHost>
```

apigee

Configuring 2-way TLS from client to Edge (Public Cloud)

01

Server keystore

Create a keystore and upload certificate and private key of the server

02

Truststore

If the client uses a self-signed certificate, or a certificate that is not signed by a trusted CA, create a truststore on Edge that contains the CA chain of the client certificate.

03

Virtualhost

Create a support ticket so the virtualhost is created with the suitable configuration.

```
<VirtualHost name="myTLSVHost">
  <HostAliases>
    <HostAlias>apiTLS.myCompany.com</HostAlias>
  </HostAliases>
  <Port>443</Port>
  <SSLInfo>
    <Enabled>true</Enabled>
    <ClientAuthEnabled>false</ClientAuthEnabled>
    <KeyStore>myTestKeystore</KeyStore>
    <KeyAlias>myKeyAlias</KeyAlias>
  </SSLInfo>
</VirtualHost>
```

apigee

Securing calls to the backend

- Generally the backend is locked down
  - You don't want apps to be able to call directly to backend

Securing calls to the backend

- Generally the backend is locked down
  - You don't want apps to be able to call directly to backend
- Options for securing the communication to the backend server
  - Credentials

apigee

Securing calls to the backend

- Generally the backend is locked down
  - You don't want apps to be able to call directly to backend
- Options for securing the communication to the backend server
  - Credentials
  - OAuth (adds significant complexity to backend calls)

apigee

Securing calls to the backend

- Generally the backend is locked down
  - You don't want apps to be able to call directly to backend
- Options for securing the communication to the backend server
  - Credentials
  - OAuth (adds significant complexity to backend calls)
  - Two-way TLS

apigee

Securing calls to the backend

- Generally the backend is locked down
  - You don't want apps to be able to call directly to backend
- Options for securing the communication to the backend server
  - Credentials
  - OAuth (adds significant complexity to backend calls)
  - Two-way TLS
  - IP Whitelisting (can be spoofed)

apigee

Securing backend communication with 2-Way TLS

- Obtain/generate client certificate for Edge

Securing backend communication with 2-Way TLS

- Obtain/generate client certificate for Edge
- Create and populate a keystore on Edge containing Edge's cert and private key

apigee

Securing backend communication with 2-Way TLS

- Obtain/generate client certificate for Edge
- Create and populate a keystore on Edge containing Edge's cert and private key
- Create and populate a truststore on Edge containing trusted certs

apigee

Securing backend communication with 2-Way TLS

- Obtain/generate client certificate for Edge
- Create and populate a keystore on Edge containing Edge's cert and private key
- Create and populate a truststore on Edge containing trusted certs
- Configure the TargetEndpoint or TargetServer

Configuring 2-Way TLS from Edge to target

```
<TargetServer name="target1">
  <SSLInfo>
    <Enabled>true</Enabled>
    <ClientAuthEnabled>true</ClientAuthEnabled>
    <KeyAlias>myKeystore</KeyAlias>
    <KeyStore>myKey</KeyStore>
    <TrustStore>myTrustStore</TrustStore>
  </SSLInfo>
</TargetServer>
```

Data Masking

- Edge's trace tool allows developers to capture runtime traffic
- Some of the data exposed by the trace tool may be sensitive information, such as passwords, credit card numbers, or personal health information

Data Masking

- Edge's trace tool allows developers to capture runtime traffic
- Some of the data exposed by the trace tool may be sensitive information, such as passwords, credit card numbers, or personal health information
- To filter this data out of the captured trace information, Edge provides data masking
    - block values in XML payloads, JSON payloads, and variables

apigee

Data Masking

- Edge's trace tool allows developers to capture runtime traffic
- Some of the data exposed by the trace tool may be sensitive information, such as passwords, credit card numbers, or personal health information
- To filter this data out of the captured trace information, Edge provides data masking
  - block values in XML payloads, JSON payloads, and variables
- Data masking configurations can be set
  - globally for an organization
    - POST /v1/o/{org}/maskconfigs
  - or on specific apis
    - POST /v1/o/{org}/apis/{api}/maskconfigs

apigee

Using mask configurations

```
<MaskDataConfiguration name="default">
  <XPathsRequest>
     <XPathRequest>/apigee:Greeting/apigee:User</XPathRequest>
  </XPathsRequest>
  <XPathsResponse>

<XPathResponse>/apigee:Greeting/apigee:User</XPathResponse>
  </XPathsResponse>
  <JSONPathsRequest>
    <JSONPathRequest>$.store.book[*].author</JSONPathRequest>
  </JSONPathsRequest>
  <JSONPathsResponse>
<JSONPathResponse>$.store.book[*].author</JSONPathResponse>
  </JSONPathsResponse>
  <XPathsFault>
     <XPathFault>/apigee:Greeting/apigee:User</XPathFault>
  </XPathsFault>
  <JSONPathsFault>
     <JSONPathFault>$.store.book[*].author</JSONPathFault>
  </JSONPathsFault>
  <Variables>
     <Variable>request.header.user-agent</Variable>
    <Variable>request.formparam.password</Variable>
  </Variables>
</MaskDataConfiguration>
```

Using mask configurations

```
<MaskDataConfiguration name="default">
  <XPathsRequest>
     <XPathRequest>/apigee:Greeting/apigee:User</XPathRequest>
  </XPathsRequest>
  <XPathsResponse>

<XPathResponse>/apigee:Greeting/apigee:User</XPathResponse>
  </XPathsResponse>
  <JSONPathsRequest>
    <JSONPathRequest>$.store.book[*].author</JSONPathRequest>
  </JSONPathsRequest>
  <JSONPathsResponse>
<JSONPathResponse>$.store.book[*].author</JSONPathResponse>
  </JSONPathsResponse>
  <XPathsFault>
     <XPathFault>/apigee:Greeting/apigee:User</XPathFault>
  </XPathsFault>
  <JSONPathsFault>
     <JSONPathFault>$.store.book[*].author</JSONPathFault>
  </JSONPathsFault>
  <Variables>
     <Variable>request.header.user-agent</Variable>
    <Variable>request.formparam.password</Variable>
  </Variables>
</MaskDataConfiguration>
```

Using mask configurations

```
<MaskDataConfiguration name="default">
  <XPathsRequest>
     <XPathRequest>/apigee:Greeting/apigee:User</XPathRequest>
  </XPathsRequest>
  <XPathsResponse>

<XPathResponse>/apigee:Greeting/apigee:User</XPathResponse>
  </XPathsResponse>
  <JSONPathsRequest>
    <JSONPathRequest>$.store.book[*].author</JSONPathRequest>
  </JSONPathsRequest>
  <JSONPathsResponse>
<JSONPathResponse>$.store.book[*].author</JSONPathResponse>
  </JSONPathsResponse>
  <XPathsFault>
     <XPathFault>/apigee:Greeting/apigee:User</XPathFault>
  </XPathsFault>
  <JSONPathsFault>
     <JSONPathFault>$.store.book[*].author</JSONPathFault>
  </JSONPathsFault>
  <Variables>
     <Variable>request.header.user-agent</Variable>
    <Variable>request.formparam.password</Variable>
  </Variables>
</MaskDataConfiguration>
```

Thank You

Google Cloud

Using mask configurations

- XML payloads: Using XPath, you identify XML elements to be filtered from request or response message payloads.
- JSON payloads: Using JSONPath, you identify JSON properties to be filtered from request or response message payloads.
- Flow variables: You can specify a list of variables that should be masked in debug output. When you specify the request.content, response.content, or message.content flow variables, the request/response body is also masked.

```
<MaskDataConfiguration name="default">
  <XPathsRequest>
     <XPathRequest>/apigee:Greeting/apigee:User</XPathRequest>
  </XPathsRequest>
  <XPathsResponse>

<XPathResponse>/apigee:Greeting/apigee:User</XPathResponse>
  </XPathsResponse>
  <JSONPathsRequest>
    <JSONPathRequest>$.store.book[*].author</JSONPathRequest>
  </JSONPathsRequest>
  <JSONPathsResponse>
<JSONPathResponse>$.store.book[*].author</JSONPathResponse>
  </JSONPathsResponse>
  <XPathsFault>
     <XPathFault>/apigee:Greeting/apigee:User</XPathFault>
  </XPathsFault>
  <JSONPathsFault>
     <JSONPathFault>$.store.book[*].author</JSONPathFault>
  </JSONPathsFault>
  <Variables>
     <Variable>request.header.user-agent</Variable>
    <Variable>request.formparam.password</Variable>
  </Variables>
</MaskDataConfiguration>
```

Request Content

Body	{"logonPassword":"**********","lastName":"Smith","firstName":"Bob"}

One way vs Two way TLS

- One-way TLS (server validation)
  - Server presents certificate, client does not
  - Client optionally validates the server certificate
  - Server must validate client via other means (HTTP message traffic)
  - Basic Auth, OAuth, etc.
  - This is standard web https
- Two-way TLS (mutual authentication)
  - Both client and server present a certificate
  - Client and server each validate the other's certificate
- One-way TLS is much more commonly used
  - Common to use two-way TLS for machine-to-machine connections, including Edge to backend target

apigee

About TLS/SSL

- SSL is the predecessor to TLS
- TLS encrypts requests/responses
- Client initiates with HTTPS

HTTP persistent connections

- HTTP 1.0 connections are not persistent
  - Use Connection: Keep-Alive header
- HTTP 1.1 connections are persistent by default
  - Client or server can send Connection: close header to tear down a connection
- Connection establishment and teardown are relatively expensive
- For TLS, we want to use a persistent connection if more traffic is likely to come from the client
  - For Edge to backend, we almost always want a persistent connection
- Traffic from all clients generally flow to the same few targets, so connection is likely to be reused quickly
  - For client to Edge, balance the cost of caching connections with the likelihood of reusing connections

Network level security using Access Control policy

IP Whitelisting / Blacklisting using AccessControl policy

```
<AccessControl name="ACL">
  <IPRules noRuleMatchAction="ALLOW">
    <MatchRule action="DENY">
      <SourceAddress mask="32">10.10.10.10</SourceAddress>
    </MatchRule>
  </IPRules>
</AccessControl>
```

Denies all client request from 10.10.10.10 and allows others. More info and config documented here

Virtual host SSL configuration

- Used to differentiate incoming traffic
- Configured on Edge
- Only for client requests, not target communication
- For public cloud, virtual hosts can only be configured by Edge Support

```
GET
https://api.enterprise.apigee.com/v1/o/{org}/e/{env}
/virtualhosts/secure

 {
   "hostAliases" : [ "myorg-prod.apigee.net" ],
   "interfaces" : [],
   "name" : "secure",
   "port" : "443",
   "sSLInfo" : {
     "ciphers" : [],
     "clientAuthEnabled" : true,
     "enabled" : true,
     "ignoreValidationErrors" : false,

     "keyAlias" : "myKey",
     "keyStore" : "myKeystore",

     "protocols" : [],
     "trustStore" : "myTruststore"
   }
 }
```

Keystore

```
curl -H "Content-Type: text/xml" \
https://api.enterprise.apigee.com/v1/o/{org_name}/en
vironments/{env_name}/keystores \
-d '<KeyStore name="myKeystore"/>' -u email:password
```

Keystore

```
curl -H "Content-Type: text/xml" \
https://api.enterprise.apigee.com/v1/o/{org_name}/en
vironments/{env_name}/keystores \
-d '<KeyStore name="myKeystore"/>' -u email:password

curl -X POST -H "Content-Type: multipart/form-data"
\
-F file="@myKeystore.jar" \
"https://api.enterprise.apigee.com/v1/o/{org_name}/e
nvironments/{env_name}/keystores/{myKeystore}/keys?a
lias={key_alias}&password={key_pass}" \
-u email:password
```

Configuring 2-way TLS from Edge to target

01

Client keystore

Create a keystore and upload certificate and private key (These are typically supplied by the target system) using management API.

02

Truststore

If the backend server uses a self-signed certificate, or a certificate that is not signed by a trusted CA, create a truststore on Edge that contains the CA chain that you received from the backend server using management API.

03

Target server

Create the target server in the Edge UI with the suitable configuration using management API.

```
<TargetServer name="target1">
  <SSLInfo>
    <Enabled>true</Enabled>
    <ClientAuthEnabled>true</ClientAuthEnabled>
    <KeyAlias>myKeystore</KeyAlias>
    <KeyStore>myKey</KeyStore>
    <TrustStore>myTrustStore</TrustStore>
  </SSLInfo>
</TargetServer>
```

apigee