Connecting a Customer System to SAP Cloud Platform Integration
SFTP-Based Communication .................................................... 95
Message-Level Security ....................................................... 97
Certificate Management ....................................................... 110
7.2 Security Elements .......................................................... 117
  Security Elements (Transport-Level Security) ......................... 117
  Security Elements (Message-Level Security) ......................... 120
  Security Elements Related to the Mail Adapter ....................... 123
  How Security Artifacts Are Related to Integration Flow Configuration .......................................................... 124

8 Setting up Message-Level Security Use Cases ....................... 127
  8.1 Inbound: Message-Level Security With PKCS#7, XML DigitalSignature ...................................................... 127
  8.2 Inbound: Message-Level Security with OpenPGP .......................................................... 129
  8.3 Outbound: Message-Level Security With PKCS#7, XML DigitalSignature ...................................................... 131
  8.4 Outbound: Message-Level Security with OpenPGP .......................................................... 134

9 Specific Use Cases ............................................................ 136
  9.1 Setting Up Principal Propagation (Example Scenario) ......... 136
  9.2 Technical Landscape for On Premise-On Demand Integration .......................................................... 137
1 Introduction

You can connect various kinds of remote systems to the cloud-based integration platform using protocols such as HTTP/S, SSH and SMTP/S. Each communication protocol comes with certain options to protect the message exchange (security options).

Kind of Systems to Connect to Cloud Integration

To give you an idea of which kinds of remote systems can be connected to the integration platform, here are some typical examples (this is not a complete list):

- On-premise systems, for example, SAP systems based on SAP NetWeaver
- SFTP servers
- Cloud applications, for example, SAP SuccessFactors or SAP Cloud for Customer
- Other systems such as e-mail servers or SOAP clients

Depending on the kind of system to connect, a certain communication protocol is to be considered, as will be explained below.

To support dedicated kinds of systems (through dedicated communication protocols), the integration platform provides certain adapters. An adapter allows you to configure the details of the technical communication channel between the remote system and the integration platform.

Supported Protocols

First task when setting up an integration scenario is to set up a secure transport channel between the remote system and Cloud Integration. The following protocols can be used: Hypertext Transfer Protocol Secure (HTTPS), SSH File Transfer Protocol (SFTP) and Simple Mail Transfer Protocol (SMTP), respectively SMTP secured with transport layer security (SMTPS).

Note

Note that HTTPS is based on the Transport Layer Security (TLS) protocol.

The following table provides more information on the different aspects to consider for each protocol.
### Table 1: Protocols

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Call Direction</th>
<th>On Premise (Mandatory)</th>
<th>On Premise (Recommended)</th>
<th>Further Aspects to Consider</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTP, HTTPS</td>
<td>Inbound</td>
<td>HTTP/S sender system (for example, SAP ERP Central Component)</td>
<td>HTTP/S proxy</td>
<td>Firewall to set up and configure</td>
</tr>
<tr>
<td>HTTP, HTTPS</td>
<td>Outbound</td>
<td>HTTP/S receiver system (for example, SAP ERP Central Component)</td>
<td>Web Dispatcher OR SAP Cloud Connector</td>
<td>Firewall to set up and configure</td>
</tr>
<tr>
<td>SSH</td>
<td>Outbound</td>
<td>SFTP server (to store files)</td>
<td>Tooling for ssh key management</td>
<td>Virus scanner on inbound directory</td>
</tr>
<tr>
<td>SMTP, SMTPS</td>
<td>Outbound</td>
<td>Mail server</td>
<td>SMTPS (SMTP over SSL/TLS) support of mail server</td>
<td>Virus scanner on inbound mail boxes</td>
</tr>
</tbody>
</table>

For each protocol, different authentication options are supported—ways how the connected systems prove their trustworthiness against each other during connection setup. Connection setup is performed differently, depending on whether inbound communication (when a remote system as a sender calls Cloud Integration) or outbound communication (when Cloud Integration calls a remote system which, in turn, is then considered as the receiver) is configured. The detailed procedure also depends on the chosen protocol and authentication option.

### Adapters

The following figure illustrates some options for kinds of systems to connect to Cloud Integration. Both communication directions are considered: systems sending messages to Cloud Integration and systems that receive messages from Cloud Integration. The figure also shows which communication protocols and the Cloud Integration adapters that are to be configured in order to enable Cloud Integration to connect to the respective kind of system. Note that the figure only shows some typical use cases and is not complete.
The following table lists the available adapters:

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
</table>
| **Ariba** | Connects an SAP Cloud Platform tenant to the Ariba network. Using this adapter, SAP and non-SAP cloud applications can send and receive business-specific documents in commerce eXtensible Markup Language (cXML) format to and from the Ariba network. The sender adapter allows you to define a schedule for polling data from Ariba. Supported authentication options:  
  - Shared key  
  - Client certificate |
| **AS2** | Exchanges business-specific documents with a partner through the Applicability Statement 2 (AS2) protocol. The AS2 sender adapter can return an electronic receipt to the sender of the AS2 message (in the form of a Message Disposition Notification (MDN)). Authorization options (sender adapter):  
  - Client Certificate: Sender authorization is checked on the tenant by evaluating the subject/issuer distinguished name (DN) of the certificate (sent together with the inbound request). Can be used with the following authentication option: Client-certificate authentication (without certificate-to-user mapping).  
  - User Role: Sender authorization is checked based on roles defined on the tenant for the user associated with the inbound request. Can be used with the following authentication options:  
    - Basic authentication (using the credentials of the user)  
      For the user, the authorizations are checked based on user-to-role assignments defined on the tenant.  
    - Client-certificate authentication and certificate-to-user mapping  
      For the user derived from the certificate-to-user mapping, the authorizations are checked based on user-to-role assignments defined on the tenant. |
<p>| <strong>Facebook</strong> | Accesses and extracts information from Facebook based on certain criteria such as keywords or user data. Using OAuth, the SAP Cloud Platform tenant can access resources on Facebook on behalf of a Facebook user. |</p>
<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>HTTP</strong></td>
<td>Establishes an HTTP connection to a receiver system.</td>
</tr>
<tr>
<td>Receiver adapter</td>
<td>The adapter supports HTTP 1.1 only. This means that the target system must support chunked transfer encoding and may not rely on the existence of the HTTP Content-Length header.</td>
</tr>
<tr>
<td>Authentication options:</td>
<td></td>
</tr>
<tr>
<td>● Basic authentication: Tenant authenticates itself against the receiver using user credentials.</td>
<td></td>
</tr>
<tr>
<td>● Client certificate authentication: Tenant authenticates itself against the receiver using a client certificate.</td>
<td></td>
</tr>
<tr>
<td>● Principal propagation: Tenant authenticates itself against the receiver by forwarding the principal of the inbound user to the cloud connector, and from there to the back end of the relevant on-premise system.</td>
<td></td>
</tr>
<tr>
<td>Supported methods: GET, POST, DELETE, HEAD, PUT, TRACE</td>
<td>The method can also dynamically be determined during runtime based on a message header or property.</td>
</tr>
<tr>
<td><strong>HTTPS</strong></td>
<td>Establishes an HTTPS connection to a sender system.</td>
</tr>
<tr>
<td>Sender adapter</td>
<td>Authorization options (sender adapter):</td>
</tr>
<tr>
<td>● Client certificate authorization: Tenant evaluates the subject/issuer distinguished name of the certificate (sent together with the inbound request).</td>
<td></td>
</tr>
<tr>
<td>● Role-based authorization: Authorization is checked based on roles defined on the tenant for the user associated with the inbound request.</td>
<td></td>
</tr>
<tr>
<td>To prevent Cross-Site Request Forgery (CSRF), you can an X-CSRF-Token in the HTTP header.</td>
<td></td>
</tr>
</tbody>
</table>

⚠️ Caution

URL parameters not supported:

- throwExceptionOnFailure
- bridgeEndpoint
- transferException
- client
- clientConfig
- binding
- sslContextParameters
- bufferSize
<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>IDoc</strong> Sender adapter</td>
<td>Allows an SAP Cloud Platform tenant to exchange Intermediate Document (IDoc) messages with other systems that support communication via SOAP web services.</td>
</tr>
</tbody>
</table>
| **IDoc** Receiver adapter | **Authorization options (sender adapter):**  
- Client certificate authorization: Tenant evaluates the subject/issuer distinguished name of the certificate (sent together with the inbound request).  
- Role-based authorization: Authorization is checked based on roles defined on the tenant for the user associated with the inbound request.  
**Authentication options (receiver adapter):**  
- Basic authentication: Tenant authenticates itself against the receiver using user credentials.  
- Client certificate authentication: Tenant authenticates itself against the receiver using a client certificate.  
- Principal propagation: Tenant authenticates itself against the receiver by forwarding the principal of the inbound user to the cloud connector, and from there to the back end of the relevant on-premise system. |
| **JMS** Sender adapter | Enables asynchronous messaging by using message queues. |
| **JMS** Receiver adapter | The sender adapter stores incoming message permanently and schedules them for processing in a queue. The messages are processed concurrently.  
The JMS adapter stores only simple data types, which includes in particular: exchange properties that do not start with Camel, as well as header (in so far they are primitive data types or strings).  
To prevent situations that the JMS adapter tries to process a failed (large) message again and again, you can store messages in a dead-letter queue if it cannot be processed after two retries.  
You can manage messages that that cannot be processed using the Monitoring Web UI.  

**Caution**  
Do not use this adapter type together with Data Store Operations steps, Gather (message aggregator) steps, or global variables, as this can cause issues related to transactional behavior.  
This adapter type cannot process ZIP files correctly. Therefore, don’t use this adapter type together with Encoder or Decoder process steps that deal with ZIP compression or decompression. |
<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Mail</strong></td>
<td><strong>Sender adapter</strong>&lt;br&gt;Enables an SAP Cloud Platform tenant to exchange e-mails with an e-mail server.&lt;br&gt;With the mail sender adapter you can download e-mails from mailboxes using Internet Message Access Protocol (IMAP) or Post Office Protocol 3 (POP3), access the content of an e-mail body, and access e-mail attachments.&lt;br&gt;<strong>Receiver adapter</strong>&lt;br&gt;With the mail receiver adapter you can send out encrypted messages by e-mail. You can protect outbound e-mails at the transport layer by STARTTLS or SMTPS.&lt;br&gt;Supported transport protocols: IMAP4/POP3 (sender adapter), SMTP (receiver adapter)&lt;br&gt;To authenticate against the e-mail server, as well as the Plain User Name/Password option (username and password sent in plain text), you can select CRAM-MD5 (Challenge-Response Authentication Mechanism, Message Digest 5), which generates a hash from the user name and password and sends this to the server.</td>
</tr>
</tbody>
</table>

**Note**<br>No authentication of e-mail sender possible<br>Unlike with other adapters, if you are using the sender mail adapter, the Cloud Integration system cannot authenticate the sender of an e-mail.<br>Therefore, if someone is sending you malware, for example, it is not possible to identify and block this sender in the Cloud Integration system.<br>To minimize this danger, you can use the authentication mechanism of your mailbox. Bear in mind, however, that this mechanism might not be sufficient to protect against such attacks.<br><br>There are three possible threats when processing e-mail content:<br>● Danger to a receiver system when forwarding e-mail content<br>  E-mails can contain malware, such as viruses or Trojan horses.<br>  These will not affect the Cloud Integration system, but they can cause damage to a receiver system if it doesn't have sufficient protection strategies.<br>● Danger to the Cloud Integration system<br>  E-mail content can be designed to affect the processing runtime of a system.<br>  Processing this content overloads the system and prevents requests from being fulfilled (denial of service).<br>  The Cloud Integration system is then unavailable until the problem is fixed.<br>● Reliability of data<br>  Sending e-mails is anonymous. It is not possible to verify whether the sender of an e-mail really is who they claim to be.<br>  Even if your mailbox has an authentication mechanism, this mechanism might not be sufficient.<br>  Therefore, data contained in an e-mail (for example, the amount of an order), is not reliable without further verification.
<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
</table>
| **OData**        | Connects an SAP Cloud Platform tenant to systems using Open Data Protocol (OData) protocol in either ATOM or JSON format. The OData sender adapter can receive incoming requests in either ATOM or JSON format, the OData receiver adapter sends the OData request in the format you choose (ATOM or JSON) to the OData service provider. Only synchronous communication is supported. Operations: GET, POST, PUT, DELETE By the GET or POST method, the sender adapter can also invoke operations that are not covered by the standard CRUD (Create, Retrieve, Update, and Delete) methods (function import). Authorization options (sender adapter):  
  • Client certificate authorization: Tenant evaluates the subject/issuer distinguished name of the certificate (sent together with the inbound request).  
  • Role-based authorization: Authorization is checked based on roles defined on the tenant for the user associated with the inbound request.  
  • Basic authentication: Tenant authenticates itself against the receiver using user credentials.  
  • Client certificate authentication: Tenant authenticates itself against the receiver using a client certificate.  
  • Principal propagation: Tenant authenticates itself against the receiver by forwarding the principal of the inbound user to the cloud connector, and from there to the back end of the relevant on-premise system. Authentication options (receiver adapter): |
<p>| <strong>ODC</strong>          | Connects an SAP Cloud Platform tenant to SAP Gateway OData Channel (through transport protocol HTTPS).                                                                                                                                                   |
| <strong>RFC</strong>          | Connects an SAP Cloud Platform tenant to a remote receiver system using Remote Function Call (RFC). RFC is the standard interface used for integrating On-premise ABAP systems to the systems hosted on cloud using SAP Cloud Connector. The adapter supports SAP NetWeaver, version 7.31 or higher. |</p>
<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SFTP</strong></td>
<td>Connects an SAP Cloud Platform tenant to a remote system using the SSH File Transfer protocol to read files from the system (sender adapter) or to write files to the system (receiver adapter). SSH File Transfer protocol is also referred to as Secure File Transfer protocol (or SFTP).</td>
</tr>
<tr>
<td>Sender adapter</td>
<td></td>
</tr>
<tr>
<td>Receiver adapter</td>
<td></td>
</tr>
<tr>
<td><strong>Caution</strong></td>
<td>This adapter does not support connections to FTP servers.</td>
</tr>
<tr>
<td></td>
<td>SSH uses a symmetric key length with at least 128 bits to protect FTP communication. The asymmetric key length used in SSH is typically 2048 bits, but at least 1024 bits.</td>
</tr>
<tr>
<td></td>
<td>Authentication options (sender and receiver adapter):</td>
</tr>
<tr>
<td></td>
<td>● User name/password authentication: SFTP server authenticates the calling component based on the user name and password.</td>
</tr>
<tr>
<td></td>
<td>● Public key authentication SFTP server authenticates the calling component based on a public key.</td>
</tr>
<tr>
<td><strong>SOAP 1.x</strong></td>
<td>Exchanges messages with another system that supports Simple Object Access Protocol (SOAP) 1.1.</td>
</tr>
<tr>
<td>Sender adapter</td>
<td>Authorization options (sender adapter): Exchanges messages with another system that supports SOAP 1.1 or SOAP 1.2 (transport protocol: HTTP).</td>
</tr>
<tr>
<td>Receiver adapter</td>
<td>● Client certificate authorization: Tenant evaluates the subject/issuer distinguished name of the certificate (sent together with the inbound request).</td>
</tr>
<tr>
<td></td>
<td>● Role-based authorization: Authorization is checked based on roles defined on the tenant for the user associated with the inbound request.</td>
</tr>
<tr>
<td></td>
<td>Authentication options (receiver adapter):</td>
</tr>
<tr>
<td></td>
<td>● Basic authentication: Tenant authenticates itself against the receiver using user credentials.</td>
</tr>
<tr>
<td></td>
<td>● Client certificate authentication: Tenant authenticates itself against the receiver using a client certificate.</td>
</tr>
<tr>
<td></td>
<td>● Principal propagation: Tenant authenticates itself against the receiver by forwarding the principal of the inbound user to the cloud connector, and from there to the back end of the relevant on-premise system.</td>
</tr>
<tr>
<td></td>
<td>You have the option to set SOAP headers using Groovy script.</td>
</tr>
<tr>
<td>Feature</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
</tr>
</tbody>
</table>
| **SOAP SAP RM**  
Sender adapter  
Receiver adapter | Exchanges messages with another system based on the SOAP communication protocol and SAP Reliable Messaging (SAP RM) as the message protocol. SAP RM is a simplified communication protocol for asynchronous Web service communication that does not require the use of Web Service Reliable Messaging standards.  
Authorization options (sender adapter):  
- Client certificate authorization: Tenant evaluates the subject/issuer distinguished name of the certificate (sent together with the inbound request).  
- Role-based authorization: Authorization is checked based on roles defined on the tenant for the user associated with the inbound request.  
Authentication options (receiver adapter):  
- Basic authentication: Tenant authenticates itself against the receiver using user credentials.  
- Client certificate authentication: Tenant authenticates itself against the receiver using a client certificate.  
- Principal propagation: Tenant authenticates itself against the receiver by forwarding the principal of the inbound user to the cloud connector, and from there to the back end of the relevant on-premise system.  
You have the option to set SOAP headers using Groovy script. |
| **SuccessFactors REST**  
Sender adapter  
Receiver adapter | Connects an SAP Cloud Platform tenant to a SuccessFactors system using the REST message protocol. |
| **SuccessFactors SOAP**  
Sender adapter  
Receiver adapter | Connects an SAP Cloud Platform tenant to SOAP-based web services of a SuccessFactors system. |
| **SuccessFactors OData V2**  
Receiver adapter | Connect an SAP Cloud Platform tenant to a SuccessFactors system using OData V2  
Supported features of OData version 2.0:  
- Operations: GET (get single entity as an entry document), PUT (update existing entry with an entry document), POST (create new entry from an entry document), MERGE (incremental update of an existing entry that does not replace all the contents of an entry), UPSERT (combination of Update OR Insert)  
- Query options: $expand, $skip, and $top  
- Servicer-side pagination  
- Client-side pagination  
- Pagination enhancement: Data retrieved in chunks and sent to Cloud Integration  
- Deep insert: Create a structure of related entities in one request  
- Authentication options: Basic authentication  
- Reference links: Link two entities using the &lt;link&gt; tag |
<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
</table>
| **SuccessFactors OData V4** Receiver adapter | Connect an SAP Cloud Platform tenant to a SuccessFactors system using OData V4  
Supported features of OData version 4.0:  
- Operations: GET, POST, PUT, DELETE  
- Navigation  
- Primitive types supported as per OData V4 specification  
- Structural types supported for create/update operations:  
  - Edm.ComplexType  
  - Edm:EnumType  
  - Collection(Edm.PrimitiveType) and Collection(Edm.ComplexType) |
| **Twitter** Receiver adapter | Accesses Twitter and either reads or posts tweets.  
Using OAuth, the SAP Cloud Platform tenant can access resources on Twitter on behalf of a Twitter user. |

As well as the transport-level security options, you can also secure the communication at message level. This protects the content of the exchanged messages by means of digital encryption and signatures. Various security standards are available to do this: PKCS#7, XML Digital Signature, OpenPGP, and WS-Security.
2 Basic Setup for HTTPS-Based Communication

To configure connections using HTTPS, you need a specific setup in which a load balancer component (a BIG-IP server from vendor F5 Networks) is interconnected for inbound calls between the remote sender system and the Cloud Integration tenant.

The BIG-IP load balancer terminates each inbound TLS (Transport Layer Security) request and re-establishes a new one for the connection to the tenant where the message will be processed. The following figure shows this setup:

**Note**

Throughout this documentation, the terms *inbound* and *outbound* reflect the perspective of the integration platform.

- **Inbound** refers to message processing from a customer system to Cloud Integration. Here, the Cloud Integration tenant is the server.
- **Outbound** refers to message processing from the integration platform to a customer system (where the integration platform is the client).
3 Connecting to an On-Premise Landscape (Example Setup)

To give you an idea of what the technical landscape behind a real-life integration scenario looks like, here is an example for the SAP Cloud for Customer (C4C)-to-SAP ERP integration scenario. In this scenario, SAP’s own cloud solution SAP Cloud for Customer (C4C) is connected with an on-premise SAP Enterprise Resource Planning (ERP) system through Cloud Integration.

The following figure shows a typical setup of components:

The left side of the figure covers the communication of Cloud Integration with the on-premise system in the customer landscape.

The setup contains components that all are connected by HTTPS communication. Typical adapters are the IDoc adapter for the connection between the on-premise system and Cloud Integration, and the SOAP adapter for the connection between SAP Cloud for Customer and Cloud Integration (within the SAP Cloud).

The upper path shows the connection from Cloud Integration to the on-premise system, which is located in the customer landscape. This is the outbound communication from the perspective of the integration platform, but is an inbound connection from the perspective of the customer landscape. Therefore, to protect the components in the customer landscape from remote calls from the Internet, a load balancer component is required – which is either a Web Dispatcher component or the SAP Cloud Connector.

The lower path shows the connection from the on-premise system to Cloud Integration. From the perspective of Cloud Integration, this is an inbound connection and, therefore, again a load balancer is required to protect the tenant that actually processes the message against remote calls. This is the BIG-IP load balancer, which is involved in all HTTPS inbound requests by default, and is not shown in the figure for the sake of simplicity. Also, this component is preconfigured by SAP and does not require any further configuration for such a scenario.
4 Configuring Inbound Communication

Inbound communication refers to message processing from a remote system (often located in the customer landscape) to Cloud Integration. Here, the integration platform is the server.

The following figure illustrates the basic setup for inbound communication:

![Diagram of inbound communication]

Configuring inbound communication means setting up the connection of a remote sender system with the integration platform.

Sender Systems You Can Connect to the Integration Platform

You can connect the following kinds of sender systems to the integration platform (examples):

- A cloud application, for example, SuccessFactors
- An on-premise application, for example, SAP ERP
- A SOAP client
- An SFTP server
  - In this case, the integration platform reads files from the SFTP server (polling).

Authentication and Authorization

For this communication direction, you can configure how sender systems should be authenticated against the Cloud Integration. Additionally, you can define authorizations for these entities with regard to access to the Cloud Integration components.

- Authentication
  - Verifies the identity of the calling entity.
Authorization
Checks what a user or other entity is authorized to do (for example, as defined by roles assigned to it). In other words, the authorization check evaluates the access rights of a user or other entity.

The detailed procedure depends on the security option that you want to configure for the communication path. See the relevant topic below.

Combinations of Authentication and Authorization (Inbound)

Authentication and authorization options can be combined in a specific way for inbound communication.

The following table shows which combinations of authentication and authorization options are supported for inbound calls.

Table 3: Combination of Authentication/Authorization Options

<table>
<thead>
<tr>
<th>Authentication Option ...</th>
<th>Can Be Used with the Following Authorization Option ...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic authentication</td>
<td>Role-based authorization</td>
</tr>
<tr>
<td>The sender (client) authentiﬁes itself against the server based on user credentials (SCN user name and password). The HTTP header of the inbound message (from the sender) contains the user name and password.</td>
<td>For this user, the authorizations are checked based on user-to-role assignments defined on the tenant. To authorize a sender system to process messages on a tenant, the role ESBMessaging.send has to be assigned to the associated user.</td>
</tr>
<tr>
<td>Client-certificate authentiﬁcation and certificate-to-user mapping</td>
<td>Role-based authorization</td>
</tr>
<tr>
<td>The sender (client) authentiﬁes itself against the server based on a digital client certificate. Furthermore, this certiﬁcate is mapped to a user (based on the information contained in a Certificate-to-User Mapping artifact deployed on the tenant).</td>
<td>For the user derived from the certificate-to-user mapping, the authorizations are checked based on user-to-role assignments defined on the tenant. To authorize a sender system to process messages on a tenant, the role ESBMessaging.send has to be assigned to the associated user.</td>
</tr>
<tr>
<td>Client-certificate authentiﬁcation (without certificate-to-user mapping)</td>
<td>Subject/Issuer DN authorization check of a certiﬁcate</td>
</tr>
<tr>
<td>The sender (client) authentiﬁes itself against the server based on a digital client certificate.</td>
<td>In a subsequent authorization check, the permissions of the sender are checked on the tenant by evaluating the distinguished name (DN) of the client certiﬁcate of the sender.</td>
</tr>
</tbody>
</table>

Note
You can map multiple certiﬁcates to the same user (n:1 certificate-to-user mappings possible).
### Authentication Option ...

<table>
<thead>
<tr>
<th>OAuth</th>
</tr>
</thead>
<tbody>
<tr>
<td>OAuth allows you to set up authentication scenarios without the need to share credentials.</td>
</tr>
<tr>
<td>More information on the concepts:</td>
</tr>
<tr>
<td>Protecting Applications with OAuth 2.0</td>
</tr>
<tr>
<td>OAuth 2.0 Specification</td>
</tr>
</tbody>
</table>

#### Note
This option is supported for the following sender adapter types: SOAP (SOAP 1.x), SOAP (SAP RM), HTTPS.

### Related Information

- Authentication and Authorization Options (Inbound) [page 78]
- Inbound/HTTPS/Basic Authentication [page 18]
- Inbound/HTTPS/Client Certificate Authentication (With Certificate-to-User Mapping) [page 21]
- Inbound/HTTPS/Client Certificate Authentication [page 23]
- Inbound: SFTP [page 29]

### 4.1 Inbound/HTTPS/Basic Authentication

#### Prerequisites

SAP has provided you or your organization with an account and tenant. Your tenant administrator has assigned you the integration developer permissions.

#### Context

This option is based on user credentials (user name and password).

We assume that you have the Developer Edition. If you are using another edition, tasks related to the management and deployment of the tenant keystore have to be performed by SAP.
Procedure

1. Configure the sender system. 

   This detailed procedure depends on the type of sender system and will not be covered here.
   
   However, make sure that the following steps are accomplished.
   
   a. Make sure that the sender keystore contains a root certificate of one of those certification authorities (CAs) that are supported by the load balancer.
      
      More information: Load Balancer Root Certificates Supported by SAP [page 94]
   b. In case you do not have a SAP Community Network (SCN) user yet, go to SCN (http://scn.sap.com) and request a user.
      
      To enable the sender for this authentication option, a communication user has to be created for the sender system. SCN users are used for this purpose. Use an SCN dialog user for the communication between the sender system and the tenant.
      
      Make sure that the message sent from the sender to the tenant contains this user in the message header.

2. Authorize the user to process messages on the tenant.

   The user has to be assigned the role ESBMessaging.send.
   
   You perform the user-to-role assignment in SAP Cloud Platform Cockpit.

   Table 4: SAP Cloud Platform Cockpit URLs

<table>
<thead>
<tr>
<th>Region</th>
<th>URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Europe (Rot)</td>
<td><a href="https://account.hana.ondemand.com/cockpit">https://account.hana.ondemand.com/cockpit</a></td>
</tr>
<tr>
<td>US East (Ashburn)</td>
<td><a href="https://account.us1.hana.ondemand.com/cockpit">https://account.us1.hana.ondemand.com/cockpit</a></td>
</tr>
<tr>
<td>Australia (Sydney)</td>
<td><a href="https://account.ap1.hana.ondemand.com/cockpit">https://account.ap1.hana.ondemand.com/cockpit</a></td>
</tr>
</tbody>
</table>

In SAP Cloud Platform Cockpit, choose Security >> Authorizations >> Assign. Enter your user ID, and choose Assign.

To authorize a sender system to call a tenant (using HTTPS/basic authentication) so that messages can be processed on it, you need to assign the specific role ESBmessaging.send to the associated technical user. For Application, choose the option that ends with iflmap (corresponding to a runtime node of the cluster that is responsible for processing the message).
3. Create and deploy a tenant keystore that contains a valid client certificate (which is accepted by the load balancer).

Although you configure inbound communication, a keystore with a valid client certificate is required for the tenant due to a connectivity test which is performed regularly for an active tenant cluster.

More information:
Basic Authentication [page 80]

4. Configure basic authentication for the related integration flow.

To open the design tool for integration flows, open a browser and enter the Web UI URL you have received from SAP in the mail that informs you that your tenant has been provided.

That way, you connect to your tenant and open the Web design tool for integration content. To create and design integration flows, go to the Design tab.

a. Open the integration flow with the integration designer and click the connection for the associated sender adapter.

b. As Authorization choose User Role and specify the role against which to check inbound authorization.

c. After you have finished configuring the integration flow (including the processing steps for your scenario), deploy the integration flow on the tenant.
4.2 Inbound/HTTPS/Client Certificate Authentication (With Certificate-to-User Mapping)

Prerequisites

SAP has provided you or your organization with an account and tenant. Your tenant administrator has assigned you the integration developer permissions.

Context

We assume that you have the Developer Edition. If you are using another edition, tasks related to the management and deployment of the tenant keystore have to be performed by SAP.

Procedure

1. Configure the sender system.
   a. Make sure that the sender keystore contains the root certificate of the load balancer server certificate.
   b. Make sure that the sender keystore contains a client certificate that is signed by one of the CAs supported by the load balancer.

More information: Load Balancer Root Certificates Supported by SAP [page 94]

2. Define a user and authorize this user to process messages on the tenant.
   As you use certificate-to-user mapping, this can be any user (say: myUser).
   The user has to be assigned the role ESBMessaging.send.
   You perform the user-to-role assignment in SAP Cloud Platform Cockpit.

Table 5: SAP Cloud Platform Cockpit URLs

<table>
<thead>
<tr>
<th>Region</th>
<th>URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Europe (Rot)</td>
<td><a href="https://account.hana.ondemand.com/cockpit">https://account.hana.ondemand.com/cockpit</a></td>
</tr>
<tr>
<td>US East (Ashburn)</td>
<td><a href="https://account.us1.hana.ondemand.com/cockpit">https://account.us1.hana.ondemand.com/cockpit</a></td>
</tr>
<tr>
<td>Australia (Sydney)</td>
<td><a href="https://account.ap1.hana.ondemand.com/cockpit">https://account.ap1.hana.ondemand.com/cockpit</a></td>
</tr>
</tbody>
</table>

In SAP Cloud Platform Cockpit, choose Authorizations. the following figure shows the page where you assign the role ESBMessaging.send to the user.
3. Create and deploy a tenant keystore that contains a valid client certificate (that is accepted by the load balancer).

A tenant keystore with this certificate is also required for inbound basic authentication as part of an automatic, regular connectivity tests. The basic technical connectivity of a cluster is checked on a regular basis, as soon as the cluster is active. For this purpose, every 30 seconds the tenant management node sends an HTTPS request to an assigned runtime node via the load balancer. This simulates an external call to the runtime node. To enable this communication, a keystore needs to be deployed on the tenant, containing a valid client certificate that is accepted by the load balancer as well as the root certificate of the same. If this keystore is not available or contains an invalid certificate, the cluster will raise an error.

4. Configure role-based authentication for the related integration flow.

To open the design tool for integration flows, open a browser and enter the Web UI URL you have received from SAP in the mail that informs you that your tenant has been provided.

That way, you connect to your tenant and open the Web design tool for integration content. To create and design integration flows, go to the Design tab.

a. Open the integration flow with the integration designer and click the connection for the associated sender adapter.

b. As Authorization choose User Role and specify the role against which to check inbound authorization.

c. After you have finished configuring the integration flow (including the processing steps for your scenario), deploy the integration flow on the tenant.

5. Define the certificate-to-user mapping.

a. Export the sender client certificate from the sender keystore to your local computer.

b. Create a Certificate-to-User Mapping artifact.

You perform this step using the Web-based Monitoring application.

Use the same URL like for the integration flow design tool and choose the Monitoring tab.

To create a new artifact or edit an existing one for the tenant, click the Certificate-to-User Mapping tile under Manage Security Material.

When specifying the properties of the Certificate-to-User Mapping artifact, select the sender client certificate from your hard disk and as user enter the user which is authorized to process messages on the tenant (user myUser from above).
4.3 Inbound/HTTPS/Client Certificate Authentication

Prerequisites

SAP has provided you or your organization with an account and tenant. Your tenant administrator has assigned you the integration developer permissions.

Context

We assume that you have the Developer Edition. If you are using another edition, tasks related to the management and deployment of the tenant keystore have to be performed by SAP.

Procedure

1. Configure the sender system.
   a. Make sure that the sender keystore contains the root certificate of the load balancer server certificate.
   b. Make sure that the sender keystore contains a client certificate that is signed by one of the CAs supported by the load balancer.

   More information: Load Balancer Root Certificates Supported by SAP [page 94]

2. Create and deploy a tenant keystore that contains a valid client certificate (that is accepted by the load balancer).

   A tenant keystore with this certificate is also required for inbound basic authentication as part of an automatic, regular connectivity tests. The basic technical connectivity of a cluster is checked on a regular basis, as soon as the cluster is active. For this purpose, every 30 seconds the tenant management node sends an HTTPS request to an assigned runtime node via the load balancer. This simulates an external call to the runtime node. To enable this communication, a keystore needs to be deployed on the tenant, containing a valid client certificate that is accepted by the load balancer as well as the root certificate of the same. If this keystore is not available or contains an invalid certificate, the cluster will raise an error.

3. Configure the security-related settings (for inbound communication) in the associated integration flow.
a. Create a new integration flow or open an existing integration flow with the Integration Designer.
b. Open the integration flow with the integration designer and click the connection for the associated sender adapter.
c. As Authorization choose Client Certificate and browse for the sender client certificate (for example <UserID>.crt) from your local file system or enter the Subject DN (information used to authorize the sender) and Issuer DN (information about the Certificate Authority who issues the certificate) manually.

This step is required to configure the authorization check in the integration flow.

Related Information

Client Certificate Authentication (Inbound) [page 86]
Load Balancer Root Certificates Supported by SAP [page 94]

4.4 Inbound/HTTPS/OAuth

Context

The following use cases can be implemented for inbound communication:

- Using the OAuth Client Credentials Grant scenario to support system-to-system communication
- Using an OAuth SAML bearer destination to implement principal propagation between accounts

More information on the concepts:

Protecting Applications with OAuth 2.0
OAuth 2.0 Specification

Note

This option is supported for the following sender adapter types: SOAP (SOAP 1.x), SOAP (SAP RM), HTTPS.

Procedure
4.4.1 OAuth Client Credentials Grant

You can configure OAuth authentication, particularly the Client Credentials Grant variant, for inbound calls from sender systems to the integration platform. This gives the sender (client) application access to the associated runtime node through OAuth authentication.

It works as follows: A client requests access to a protected virtual environment (for example, a runtime node that is to be used to process messages received by the client application). The initial request is sent to an OAuth authorization server (which is part of SAP Cloud).

After the client has been authenticated successfully by the OAuth authorization server, it will be provided with the access tokens that are required to process messages on the associated runtime node. In terms of OAuth, the client uses the access token to get access to the protected resources (represented by the virtual environment of a runtime node).

This process is executed without any manual interaction, and is therefore best suited to system-to-system communication.

More information on the concepts:

- Protecting Applications with OAuth 2.0
- OAuth 2.0 Specification

**Note**

This option is supported for the following sender adapter types: SOAP (SOAP 1.x), SOAP (SAP RM), HTTPS.

### Configuring OAuth with Client Credentials Grant

Using the SAP Cloud Platform cockpit, perform the following steps.

1. Register the client application as the OAuth client in the consumer account using the SAP Cloud Platform cockpit (in the **Security** ➤ **OAuth** section, go to the **Clients** tab).
   
   Also specify a subscription in order to restrict the authorizations associated with the access token on the particular runtime node.

   **Note**

   You can only subscribe to runtime nodes with node type iflmap or hcioem.

   Perform this step as described under **Registering an OAuth Client**.
To enable this security setting for the abovementioned scenario (client application sending messages to the cloud-based integration platform), specify the following information when registering the OAuth client:

- **As Subscription**, select the VM name of the runtime node (that ends with the node type, for example, ...iflmap).
  You can only register applications for node type iflmap or hcioem.
- **Enter a client ID.**
  You can either get a client ID from the client or you can choose one (you then have to forward this ID to the client).
- **As Authorization Grant**, choose **Client Credentials**.
- **Enter a secret (as assigned to the client application).**
- **Specify a Token Lifetime** to increase the security level.

2. Assign the user with name oauth_client_<client ID> to the role ESBMessaging.send in the subscription of the consumer account (for the iflmap/hcioem node).
   To do this, select the **Security Authorizations** section. Perform this step as described under .

3. On the client side, perform the following steps:
   1. **Perform a POST HTTPS call to** https://oauthasservices-<consumer-account>.<landscape host name>/oauth2/api/v1/token?grant_type=client_credentials.
      For the URL part https://oauthasservices-<consumer-account>.<landscape host name>/oauth2/api/v1/token, you can find the value that you need to enter in the receiver account, SAP Cloud Platform Cockpit, under **Security > OAuth**. On the **Branding** tab in section **OAuth URLs**, the URL is displayed under **Token Endpoint**.
      Use basic authentication where the client ID is the user and the secret is the password. This call returns the access token.
      Example:

      ```json
      { 
        "access_token": "8271a067126f0aa93b46c2fe07c68880",
        "token_type": "Bearer",
        "expires_in": 0,
        "scopes": []
      }
      ```

2. Perform an HTTPS call to the endpoint URI with the HTTP header with name “Authorization” and value “Bearer <access token>”.
   You can repeat the call several times until the access token is invalid. Then execute step a) again.

### 4.4.2 OAuth SAML Bearer Destination

You can enable principal propagation between SAP Cloud Platform accounts.

If you have chosen this option, the identity of the user associated with the sender (client) application is forwarded from the sender account to the receiver account. It is a prerequisite for this scenario that the authentication method OAuth 2.0 is used, in particular, the OAuth 2.0 SAML bearer assertion flow.

A Security Assertion Markup Language (SAML) 2.0 Bearer Assertion is used to authenticate the client as well as to request the OAuth 2.0 access token from an OAuth 2.0 authorization server (hosted in the SAP cloud).
To configure the scenario, an OAuth2SAMLBearerAssertion destination has to be specified on the sender account.

More information:

- SAML Bearer Assertion Authentication
- Principal Propagation to OAuth-Protected Applications

Note

This option is supported for the following sender adapter types: SOAP (SOAP 1.x), SOAP (SAP RM), HTTPS.

Create Connection of Sender and Receiver Account with Trusted Identity Providers

Make sure that the settings for SAML communication between SAP Cloud Platform and a trusted identity provider are specified. This communication has to be established for both the sender and receiver account. In this way you establish a trust relationship between the sender and receiver account.

Note

Note the following remarks related to the identity providers of the sender and receiver account:

- You can assign different identity providers to sender and receiver accounts.
- Sender account: You must **not** assign the default SAP ID Service as the identity provider.
- Receiver account: You can assign the default SAP ID Service for testing purposes. This identity provider is configured by default and has a landscape-dependent *Local Service Provider* name.

Perform the following steps for both the sender and receiver account:

To configure the settings, go to SAP Cloud Platform Cockpit and choose **Security** ➤ **Trust**.

Proceed as described under *ID Federation with the Corporate Identity Provider*.

Configure OAuth in Receiver Account

Configure the OAuth settings for the receiver account. In this way, you register the client application as the OAuth client.

Go to SAP Cloud Platform Cockpit and choose **Security** ➤ **OAuth** ➤ *(Clients tab)*.

Proceed as described under *Configuring OAuth 2.0*.

Note the following specific settings:

- As **Subscription**, select the VM name of the runtime node (that ends with the node type, for example, ...iflmap).
  
  You can only register applications for node type *iflmap* or *hcioem*. 
● Enter a client ID. You can either get a client ID from the client or you can choose one (you then have to forward this ID to the client).
● As Authorization Grant, choose Client Credentials.
● Enter a secret (as assigned to the client application).
● Specify a Token Lifetime to increase the security level.

**Configure Trust to Sender Local Service Provider in the Receiver Account**

In the receiver account, configure a trust relationship to the sender’s local service provider.

Note that here the local service provider of the sender account takes the role of an additional trusted entity provider for the receiver account.

To configure the settings, go to the SAP Cloud Platform Cockpit and choose Security > Trust (Trusted Entity Provider tab).

Proceed as described under ID Federation with the Corporate Identity Provider (subsection Configure Trust to the SAML Identity Provider).

As Name, enter the Local Service Provider name from the sender account.

Enter the Signing Certificate as specified for the sender’s local service provider.

**Specify User Group in Receiver Account and Enable User Group to Process Message on Runtime Node**

In the receiver account, perform the following tasks:

1. Create a user group.
   To configure the settings, go to the SAP Cloud Platform cockpit and choose Security > Authorizations > On the Groups tab, create a new group.

2. Create a mapping of the user group to the local sender service provider.
   You have the following options:
   ○ Specify a default group, which means that all users logged in via the sender’s local service provider are assigned to this user group.
   To configure the settings for the default group, go to the SAP Cloud Platform Cockpit and choose Security > Trust > On the Trusted Identity Provider tab, go to the identity provider specified previously. On the Groups tab, choose Add Default Group and enter the name of the newly created user group.
   ○ Define mapping rules based on the user attributes (such as e-mail address).

3. Assign the user group to the ESBMessaging.send role.
   You perform this step to enable all users that are assigned to the user group created to execute integration flows on the runtime node application.
To configure the settings, go to the SAP Cloud Platform Cockpit and choose Security. On the Groups tab, select the group defined previously and choose Assign. Select the role ESBMessaging.send.

More information:

Create OAuth2SAMLBearerAssertion Destination in the Sender Account

To configure the settings, go to the SAP Cloud Platform Cockpit and choose Destinations.

- As Type select HTTP.
- As Proxy Type select Internet.
- As Authentication select OAuth2SAMLBearerAssertion.
- As Audience enter the Local Provider Name of the receiver account.
- As Client Key specify the key that identifies the consumer to the authorization server. This key must contain the ID of the client created above.
- As Token Service URL enter the OAuth token URL for the receiver account. You can find the value to be entered in the receiver account, SAP Cloud Platform Cockpit, under Security OAuth. On the Branding tab in section OAuth URLs, the URL is displayed under Token Endpoint.
- As Token Service User specify the user for basic authentication for the OAuth server (if required). This entry must contain the ID of the client created above.
- As Token Service Password specify the Password for Token Service User (if required). This entry must contain the secret of the confidential client.
- As Additional Property add the property authnContextClassRef with the following value:

Related Information

Setting Up Principal Propagation (Example Scenario) [page 136]

4.5 Inbound: SFTP

In a scenario using SFTP, an SFTP client connects to an SFTP server in order to perform one of the following tasks:

- SFTP client writes (pushes) a file to a file directory on an SFTP server.
- SFTP client reads (pulls) data from the SFTP server.
  Typically the SFTP client periodically reads files from the SFTP server.

In case the SFTP server is hosted by the customer: Provide SAP with the public key, public key algorithm and host name of the SFTP server (is used to configure the known hosts file for the tenant).
Configuring the SFTP Server (from which data is to be read)

Configure the authorized keys file on the SFTP server. It has to contain the public key of the SFTP client (tenant).

Who performs this task depends on whether the SFTP server is hosted by the customer or by SAP.

Configuring the Integration Flow

Configure the SFTP sender adapter to specify the technical details how the data is to be read from the SFTP server.

Related Information

How SFTP Works [page 95]
Creating SFTP Keys [page 55]
Inbound SFTP With Public Key Authentication [page 30]

4.5.1 Inbound SFTP With Public Key Authentication

For an SFTP client connected to an SFTP server using the Public Key authentication option, the following artifacts have to be generated and stored at the locations summarized in the following table. The table also
shows which artifacts need to be exchanged between the client and the server (during the onboarding process):

Table 6:

<table>
<thead>
<tr>
<th>SFTP Client Side</th>
<th>SFTP Server Side</th>
</tr>
</thead>
<tbody>
<tr>
<td>Public keys of all connected SFTP servers</td>
<td>Public keys of all connected SFTP clients (used in order to authenticate the SFTP clients on the SFTP server side)</td>
</tr>
<tr>
<td>A public key is used in order to authenticate the SFTP server (as known host) on the SFTP client side. Public keys of all connected SFTP servers are stored in a <code>&lt;known_hosts&gt;</code> file on the client side.</td>
<td>This file has to be stored in an <code>&lt;authorized_keys&gt;</code> file on the SFTP server.</td>
</tr>
</tbody>
</table>

**Note**

The `<known_hosts>` file contains the public keys and addresses of the trusted SFTP servers. The client checks if the server is a trusted participant by evaluating a `<known_hosts>` file on the client side: If the server’s public key is listed there, the identity of the server is confirmed.

**Note**

Generating this public key is the task of the expert that hosts the SFTP server.

<table>
<thead>
<tr>
<th>Private key of SFTP client (stored on client)</th>
<th>Private key of SFTP server (stored on server)</th>
</tr>
</thead>
</table>
| **Note**

The private key of the SFTP client can be either an RSA private key file or a DSA private key file. The private key (together with its associated public key) has to be stored in a keystore. | **Note**

Generating this public key is the task of the expert that hosts the SFTP server. |

**Note**

Generating this private key is the task of the expert that hosts the SFTP client.

A tenant can connect as an SFTP client to an SFTP server (the latter either hosted at SAP or in the customer landscape).

The following figure shows the basic setup of components used for SFTP for inbound communication (when the data flow is directed from an SFTP server to the tenant).
To specify the technical details of the message flow from the SFTP sender to the tenant (SFTP client), an SFTP sender adapter has to be configured for the related integration flow.
5 Configuring Outbound Communication

Outbound communication refers to message processing from the integration platform to a remote system (where the integration platform is the client).

The following figure illustrates the basic setup for inbound communication:

Configuring inbound communication means setting up the connection of a remote receiver system with the integration platform.

Receiver Systems You Can Connect to the Integration Platform

You can connect the following kinds of receiver systems to the integration platform (examples):

- A cloud application, for example, an SAP cloud application like SuccessFactors or SAP Cloud for Customer
- An on-premise application, for example SAP ERP
  You can connect on-premise systems (located in the customer system landscape) such as SAP systems. Typical use cases for this are hybrid integration scenarios, where an on-premise SAP application (for example, SAP ERP) is integrated with an SAP cloud application (for example, SAP Cloud for Customer or SAP SuccessFactors).
- An e-mail server
  In this case, the integration platform sends e-mails to the e-mail server (for an e-mail address specified in the related adapter).
- An SFTP server
  In this case, the integration platform writes files to the SFTP server.

The detailed procedure depends on the security option that you want to configure for the communication path. See the relevant topic below.
Outbound Connections to On-Premise Systems in the Customer Landscape

For connections like this (when Cloud Integration sends a message to the on-premise system) you have to make sure that the on-premise business systems connected to the cloud are not directly exposed to the Internet. Therefore, use a reference setup as outlined in the following figure.

As an example, we show the setup for integrating SAP Cloud for Customer with SAP ERP.

In the proposed reference setup, a further component is interconnected between the on-premise system and the integration platform in the SAP Cloud that protects the on-premise system against external calls (from the Internet).

There are two different options for this component:

- SAP Cloud Connector
- Reverse proxy (for example, SAP Web Dispatcher)

Related Information

Outbound/On-Premise: Reverse Proxy or SAP Cloud Connector [page 35]
Outbound/HTTPS/Basic Authentication [page 40]
Outbound/HTTPS/Client Certificate Authentication [page 41]
Outbound: SFTP [page 43]
Outbound: SAP Cloud Connector [page 38]
5.1 Outbound/On-Premise: Reverse Proxy or SAP Cloud Connector

When you connect an on-premise (receiver) system to the integration platform, you need to interconnect either a reverse proxy or an SAP Cloud Connector between the on-premise system and the integration platform in the SAP Cloud.

Overview

To decide which option is the best one for your use case, refer to the following table.

Table 7: Decision Matrix: Reverse Proxy versus SAP Cloud Connector

<table>
<thead>
<tr>
<th>Advantages When Using Reverse Proxy</th>
<th>Advantages When Using SAP Cloud Connector</th>
</tr>
</thead>
<tbody>
<tr>
<td>Existing reverse proxy and demilitarized zone (DMZ) infrastructure can be reused for cloud scenarios: No additional components need to be operated on-premise.</td>
<td>• No inbound ports pointing to the on-premise network need to be opened.</td>
</tr>
<tr>
<td>• Firewall and DMZ remain unchanged (for example, no additional IP white-listing).</td>
<td></td>
</tr>
<tr>
<td>• Direct attacks (for example, DDOS) from the Internet are not possible.</td>
<td></td>
</tr>
<tr>
<td>IT-based, centralized approach with more re-use potential, independent of SAP Cloud Platform or Cloud Integration infrastructure (Cloud Integration)</td>
<td>De-central, simple solution that can be administered by LOBs and subsidiaries</td>
</tr>
<tr>
<td>Usage for other cloud scenarios besides SAP Cloud Platform/Cloud Integration-connectivity of backends</td>
<td>Usage for other SAP Cloud Platform-related scenarios, for example, extension apps, possible</td>
</tr>
<tr>
<td>Additional capabilities might be provided by the reverse proxy (load balancing, application gateway, rules, and so forth, depending on the used product).</td>
<td>Synchronous native RFC-client call from SAP Cloud Platform supported in addition (that means, outside Cloud Integration)</td>
</tr>
<tr>
<td>Several reverse proxy instances per target landscape in one Cloud Integration tenant</td>
<td>Propagation of cloud user identity to on-premise system is supported.</td>
</tr>
<tr>
<td>Monitoring and control included in the IT processes, tools and concepts.</td>
<td>Monitoring and control native on SAP Cloud Platform (for example, SAP Cloud Platform Cockpit. User, Security)</td>
</tr>
<tr>
<td>Re-use of existing license, but separate license needed of a reverse proxy is not used yet.</td>
<td>License comes with Cloud Integration Standard/Professional Edition.</td>
</tr>
<tr>
<td>Third party support needed, except if the SAP Netweaver Web Dispatcher is used as reverse proxy.</td>
<td>SAP support in case of issues or feature requests.</td>
</tr>
</tbody>
</table>
Decision Graph

To decide whether to use a reverse proxy or SAP Cloud Connector, you can follow the decision graph as outlined in the following figure and described further below.
Want to give the control for introduction of SAP Cloud connectivity to the LOB?

Yes

Use SAP Cloud Connector

No

Need identity provider federation?

Yes

You need SAP Cloud Connector anyway. In this case it can be used as reverse proxy, too.

No

Have a reverse proxy in place connected to the required application systems?

Yes

Use reverse proxy, add SAP Cloud Platform IP range for white-listing, use SSL/TLS, apply your IT standards.

No

Have a reverse proxy in place, but not connected to the application systems needed yet?

Yes

Add application systems to reverse proxy and use it.

No

Want to start a general investment into a reverse proxy infrastructure?

Yes

Purchase a reverse proxy and introduce it by IT (might be a project with several months of execution time).

No

Use SAP Cloud Connector
1. Independent of your current IT infrastructure setup, you want to give the control for introduction of SAP cloud connectivity to the line of business (LOB) - for example, because of timing, or special solution in a subsidiary or segment.
   - Yes: Use SAP Cloud Connector.
   - No: Go to next question.

2. Need identity provider federation?
   For example, users are managed via on-premise ldap servers like Microsoft Active Directory.
   - You need SAP Cloud Connector anyway. In this case it can be used as reverse proxy, too.
   - No: Go to next question.

3. Have a reverse proxy in place connected to the required application systems?
   - Use reverse proxy, add SAP Cloud Platform IP range for white-listing, use SSL/TLS, apply your IT standards.
   - No: Go to next question.

4. Have a reverse proxy in place, but not connected to the application systems needed yet?
   - Add application systems to reverse proxy and use it.
   - No: Go to next question.

5. Want to start a general investment into a reverse proxy infrastructure?
   - Purchase a reverse proxy and introduce it by IT.
   - No: Use SAP Cloud Connector.

---

**Caution**

This might be a project with several months of execution time.

---

**More Information**

**SAP Cloud Connector**

**Outbound: SAP Cloud Connector [page 38]**

**SAP Cloud Platform Connector** (product documentation)

Technical connectivity between cloud and on-premise systems via the SAP Cloud Platform Connector (SCN article)

**SAP Web Dispatcher**

**SAP Web Dispatcher** (product documentation)

---

**5.1.1 Outbound: SAP Cloud Connector**

SAP Cloud Connector (SCC) runs as on premise agent in a secured network and acts as a reverse invoke proxy between the on premise network and SAP Cloud Platform Integration. Due to its reverse invoke support, you don’t need to configure the on premise firewall to allow external access from the cloud to internal systems.
You can configure an outbound connection from the tenant via SAP Cloud Connector (SCC). The following figure illustrates how the connection is set up and the basic components of the scenario.

![Diagram of HANA Cloud Platform and Customer (on-premise)](image)

You need to install and configure the SAP Cloud Connector on your on premise systems for this mode of outbound communication. For more information on installing and configuring SCC, you can refer to Using SAP Cloud Connector with Cloud Integration Adapters [page 39]

**More Information**

These documents describe step-by-step how to install SAP Cloud Connector for different scenarios:

- [http://scn.sap.com/docs/DOC-42533](http://scn.sap.com/docs/DOC-42533)

### 5.1.1.1 Using SAP Cloud Connector with Cloud Integration Adapters

You can use the SAP Cloud Connector (SCC) with Cloud Integration adapters to communicate with SAP on premise systems. You have to install cloud connector on the on premise system/s that you want to communicate with, and configure it. This is a one-time activity after which you can use cloud connector with Cloud Integration adapters.

Perform the following steps to use cloud connector with Cloud Integration adapters.

1. Install SAP Cloud Connector on your on premise system. For more information, see [Installing the Cloud Connector](#).
2. Set up mutual authentication between the cloud connector and a backend system. For more information, see [Initial Configuration](#) and [Initial Configuration (HTTP)](#).
3. Enable the web application to connect to access backend system on the intranet. For more information, see [Configuring Access Control (HTTP)](#).

You can now connect to on premise systems using Cloud Integration adapters by selecting *on-premise* value in *Proxy Type* field dropdown list.
Remember

Here are some important considerations while using SAP Cloud Connector with Cloud Integration adapters:

- Ensure that the receiver URL starts with `http://` while configuring the integration flow.
- Always use Basic Authentication as the `<Authentication Type>`.
- Ensure that you deploy the credentials that enables access to the backend system that you are trying to connect to.

Note

You need to enter your tenant ID as the account name while configuring SAP Cloud Connector. You can access the tenant ID by selecting the tenant in Node Explorer and accessing the Tasks View by choosing Window ➔ Show View ➔ Tasks View.

5.2 Outbound/HTTPS/Basic Authentication

Prerequisites

SAP has provided you or your organization with an account and tenant. Your tenant administrator has assigned you the integration developer permissions.

Context

We assume that you have the Developer Edition. If you are using another edition, tasks related to the management and deployment of the tenant keystore have to be performed by SAP.

This option was referred to as basic authentication in former releases. It is based on user credentials.

Procedure

1. Create and deploy a tenant keystore that contains the receiver server root certificate.
   
   This certificate is required to identify (authenticate) the receiver system as trusted server.
   
   More information:

2. Create and deploy the credentials on the tenant.
These are user name and password that are used to authenticate the tenant calling the receiver system.

You perform this step using the Web-based Monitoring application.

a. Use the same URL like for the integration flow design tool and choose the Monitoring tab.
b. Click the Security Material tile under Manage Security Material.
c. To create a new Credentials artifact or edit an existing one for the tenant, choose Add.
d. On the Add User Credentials page, enter the attributes (Credential Name, User and Password) and choose OK.

3. Configure the security-specific settings in the related integration flow.

a. To open the design tool for integration flows, open a browser and enter the Web UI URL you have received from SAP in the mail that informs you that your tenant has been provided. That way, you connect to your tenant and open the Web design tool for integration content.
b. To create and design integration flows, go to the Design tab.
c. Open the related receiver adapter (that is used to specify the connection of the tenant with the receiver system) and select Connect Using Basic Authentication; then enter the credential name.

This is the name of the User Credentials artifact that you have deployed on the tenant in a previous step.

4. Configure the receiver keystore.

The keystore needs to contain a certificate that is signed by a certification authority (CA) which is also contained in the tenant keystore.

More information:

Creating X.509 Keys [page 49]

Related Information

Basic Authentication [page 90]

5.3 Outbound/HTTPS/Client Certificate Authentication

Prerequisites

SAP has provided you or your organization with an account and tenant. Your tenant administrator has assigned you the integration developer permissions.
Context

We assume that you have the Developer Edition. If you are using another edition, tasks related to the management and deployment of the tenant keystore have to be performed by SAP.

Procedure

1. Create and deploy the tenant keystore.

   To enable the tenant to authenticate itself as client against the receiver, a keystore with a valid client certificate has to be deployed on the tenant.

   The keystore also has to contain a certificate of the certification authority (CA) that has signed the server certificate of the receiver system.

   a. Generate the tenant client keystore used for outbound communication.

      The tenant client keystore is physically one file. It contains the private key and the associated public key client certificate (as well as all intermediate certificates of the certificate chain) used to authenticate the client - the tenant in this case - when calling the server (the receiver).

   b. Initiate a certificate signing request (CSR).

      The client certificate initially created is self-signed (owner and issuer are identical) and therefore needs to be signed by a certification authority (CA). To initiate this step, create a certificate signing request (CSR). Generate the CSR to have the recommended CA sign the certificate.

   c. Request a signed certificate from the CA.

      Once you have performed this step, the CA sends back the signed certificate.

   d. Import the signed client certificate into the keystore (and the intermediate certificates of the certificate chain).

   e. Import the server root certificate from the receiver system into the tenant client keystore.

   f. Export the tenant client certificate from the keystore (to make it available for the administrator of the receiver system).

   g. Deploy the tenant client keystore on the tenant.

   More information:

2. Configure the receiver keystore.

   In the same way as for the tenant keystore, generate a public/private key pair, create a certificate signing request and get the certificate signed by a CA. Note that this must be the CA which root certificate is also obtained in the tenant keystore.

   More information:

   Creating X.509 Keys [page 49]

3. Open the Configure the related integration flow.

   To open the design tool for integration flows, open a browser and enter the Web UI URL you have received from SAP in the mail that informs you that your tenant has been provided. That way, you connect to your
tenant and open the Web design tool for integration content. To create and design integration flows, go to the Design tab.

In the related integration flow, open the receiver channel, select the option to connect to the receiver using certificate-based authentication, and enter the private key alias.

The private key alias that enables the system to fetch that private key from the keystore that is to be used for authentication.

Related Information

Client Certificate Authentication (Outbound) [page 91]

### 5.4 Outbound: SFTP

In a scenario using SFTP, an SFTP client connects to an SFTP server in order to perform one of the following tasks:

- SFTP client writes (pushes) a file to a file directory on an SFTP server.
- SFTP client reads (pulls) data from the SFTP server.
  
Typically the SFTP client periodically reads files from the SFTP server.

In case the SFTP server is hosted by the customer: Provide SAP with the public key, public key algorithm, and host name of the SFTP server (is used to configure the known hosts file for the tenant).

**Configuring the SFTP Server (To Which Data Is To Be Written)**

Configure the authorized keys file on the SFTP server. It has to contain the public key of the SFTP client (tenant).

Who performs this task depends on whether the SFTP server is hosted by the customer or by SAP.

**Configuring the Integration Flow**

Open the related integration flow and configure the SFTP receiver adapter to specify the technical details of how the data is to be written to the SFTP server.
Related Information

How SFTP Works [page 95]
Creating SFTP Keys [page 55]
Outbound SFTP With Public Key Authentication [page 44]

5.4.1 Outbound SFTP With Public Key Authentication

For an SFTP client connected to an SFTP server using the Public Key authentication option, the following artifacts have to be generated and stored at the locations summarized in the following table. The table also shows which artifacts need to be exchanged between the client and the server (during the onboarding process):

Table 8:

<table>
<thead>
<tr>
<th>SFTP Client Side</th>
<th>SFTP Server Side</th>
</tr>
</thead>
<tbody>
<tr>
<td>Public keys of all connected SFTP servers</td>
<td>Public keys of all connected SFTP clients (used in order to authenticate the SFTP clients on the SFTP server side)</td>
</tr>
<tr>
<td>A public key is used in order to authenticate the SFTP server (as known host) on the SFTP client side. Public keys of all connected SFTP servers are stored in a <code>&lt;known_hosts&gt;</code> file on the client side.</td>
<td>This file has to be stored in an <code>&lt;authorized_keys&gt;</code> file on the SFTP server.</td>
</tr>
</tbody>
</table>

**Note**
The `<known_hosts>` file contains the public keys and addresses of the trusted SFTP servers. The client checks if the server is a trusted participant by evaluating a `<known_hosts>` file on the client side: If the server’s public key is listed there, the identity of the server is confirmed.

**Note**
Generating the public key of the SFTP server is the task of the expert that hosts the SFTP server.
A tenant can connect as an SFTP client to an SFTP server (the latter either hosted at SAP or in the customer landscape).

The following figure shows the basic setup of components used for SFTP for outbound communication (when the data flow is directed from the tenant to an SFTP server).

To specify the technical details of the message flow from the tenant (SFTP client) to the SFTP server, an **SFTP receiver adapter** has to be configured for the related integration flow.
6 Detailed Steps

6.1 Setting Up the Tenant Client Keystore

A tenant client keystore is required for each tenant that sends messages to a receiver system (server). It is the storage location for the tenant client certificate. Additionally, the required server root certificates from the connected external systems have to be imported into the tenant client keystore.

Prerequisites

You have installed the KeyStore Explorer.
For the procedure described in this documentation, it is assumed that you are using the KeyStore Explorer. You can download this tool from http://keystore-explorer.sourceforge.net/.

Context

Procedure

1. Open the KeyStore Explorer and choose New.
2. For type of keystore select JCEKS.
JCEKS provides a stronger encryption than JKS.

3. Choose **Tools ➤ Generate Key Pair**.

4. As **Algorithm** select **RSA** or **DSA**, and for **Key Size** select **2048**.

5. Choose **OK**.

6. For **Version** select **Version 3**, and for **Signature Algorithm** select **SHA-256 with RSA**. Choose a **Validity Period** as required for your scenario.

7. Next to the **Name** field, click **Edit Name**.

8. In the next dialog, enter the name parts.

The information you need to enter depends on who is the owner of the tenant for which the keystore and the contained client certificate are being generated.

Enter the relevant information to identify your tenant as the owner of the certificate.

- **Password**
- **Common Name (CN)**
  Enter a meaningful common name of your choice.
- **Organizational Unit (OU)**
  Enter a short name for your (the customer’s) organizational unit.
- **Organization Name (O)**
  Enter a short name for your (the customer’s) organization.
- **Locality Name (L)**
  Enter the name of your (the customer’s) location.
- **State Name (ST)**
  Enter the name of your (the customer’s) state.
- **Country (C)**
  Enter the name of your (the customer’s) country.

You do not need to make an entry for the **Email (E)** field.

9. Choose **OK**.
10. Enter a key alias.

When creating SSH keys (for SFTP), enter one of the following key aliases.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id_rsa</td>
<td>When you have selected RSA as Algorithm.</td>
</tr>
<tr>
<td>id_dsa</td>
<td>When you have selected DSA as Algorithm.</td>
</tr>
</tbody>
</table>

11. Choose OK.
12. Enter (and repeat) the keystore password.

You need a password to protect the private key.

Note

There is the option to specify different passwords to protect a private key and to protect the keystore as a whole.

To set up a tenant client keystore, it is mandatory that identical passwords are used.

Note

When you specify the password, follow the password rules as described in a separate topic.

13. The key pair has been generated successfully.
14. When you save the keystore for the first time, you have to specify the keystore password.

Use the same password as for the protection of the private key.
15. When you save the keystore, enter .jks as the file extension.

Results

The client certificate created initially is self-signed (owner and issuer are identical) and therefore needs to be signed by a certification authority (CA). To initiate this step, create a certificate signing request (CSR). The CA sends back the signed certificate, and you can then update your keystore accordingly.

Next Steps

When you import an existing key pair into the keystore, and you have the choice among different files with different Key Pair Type, we recommend to choose the option PKCS #12 (in case one key pair file corresponding to this format is available). This format contains the certificate in addition to the private key. If you choose one of the other Key Pair Types, the certificate has to be specified separately.
6.2 Creating X.509 Keys

You need X.509 keys to configure communication with certificate-based authentication over HTTPS and if you want to configure digital encryption and signing of messages with security standards PKCS#7 and XML Digital Signature.

Related Information

Generating a Key Pair [page 49]
Creating a Certificate Signing Request [page 53]
Requesting a Signed Certificate from a Certification Authority [page 55]
Securely Exchanging Key Material [page 74]

6.2.1 Generating a Key Pair

Prerequisites

You have installed the KeyStore Explorer.

For the procedure described in this documentation, it is assumed that you are using the KeyStore Explorer. You can download this tool from http://keystore-explorer.sourceforge.net/.
Context

Procedure

1. Open the KeyStore Explorer and open a keystore or create a new one. When creating a keystore, for type of keystore select JCEKS.

2. Choose Tools > Generate Key Pair.

3. As Algorithm select RSA and for Key Size select 2048.

4. Choose OK.

5. For Version select Version 3, and for Signature Algorithm select SHA-256 with RSA. Choose a Validity Period as required for your scenario.
6. Next to the Name field, click Edit Name.
7. In the next dialog, enter the name parts.

![Image of Name dialogue box]

The information you need to enter depends on who is the owner of the tenant for which the keystore and the contained client certificate are being generated.

Enter the relevant information to identify your tenant as the owner of the certificate.

- **Password**
- **Common Name (CN)**
  Enter a meaningful common name of your choice.

⚠️ **Caution**
Do not use *.hana.ondemand.com as the Common Name (CN).

**Note**
Note the following with regard to the usage of wildcards in the CN entries (for example *.mycompany.com):

For inbound certificate-based client authentication (where the CA-signed certificate needs to be imported into the customer’s back-end systems), wildcards in the CN field are allowed.

For outbound certificate-based client authentication (where you have to import the CA-signed certificate into the tenant keystore), wildcards in the CN field are not allowed.

- **Organizational Unit (OU)**
  Enter a short name for your (the customer’s) organizational unit.
- **Organization Name (O)**
  Enter a short name for your (the customer’s) organization.
- **Locality Name (L)**
  Enter the name of your (the customer’s) location.
- State Name (ST)
  Enter the name of your (the customer’s) state.

- Country (C)
  Enter the name of your (the customer’s) country.

You do not need to make an entry for the Email (E) field.

8. Choose **OK**.

9. Enter a key alias.

   **When creating SSH keys** (for SFTP), enter one the following key aliases.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id_rsa</td>
<td>When you have selected <strong>RSA</strong> as Algorithm.</td>
</tr>
<tr>
<td>id_dsa</td>
<td>When you have selected <strong>DSA</strong> as Algorithm.</td>
</tr>
</tbody>
</table>

10. Choose **OK**.

11. Enter (and repeat) the keystore password.

   You need a password to protect the private key.

   **Note**

   There is the option to specify different passwords to protect a private key and to protect the keystore as a whole.

   To set up a tenant client keystore, it is mandatory that identical passwords are used.

   **Note**

   When you specify the password, follow the password rules as described in a separate topic.

12. The key pair has been generated successfully.

13. When you save the keystore for the first time, you have to specify the keystore password.

   Use the same password as for the protection of the private key.

14. When you save the keystore, enter `.jks` as the file extension.

**Results**

The client certificate created initially is self-signed (owner and issuer are identical) and therefore needs to be signed by a certification authority (CA). To initiate this step, create a certificate signing request (CSR). The CA sends back the signed certificate, and you can then update your keystore accordingly.

**Next Steps**

When you import an existing key pair into the keystore, and you have the choice among different files with different **Key Pair Type**, we recommend to choose the option **PKCS #12** (in case one key pair file
corresponding to this format is available). This format contains the certificate in addition to the private key. If you choose one of the other Key Pair Types, the certificate has to be specified separately.

![Import Key Pair Type](image)

**Related Information**

Requirements for Keystore Passwords [page 116]

### 6.2.2 Creating a Certificate Signing Request

When a certificate is originally created, it is self-signed and has to be signed by a certification authority (CA) prior to being used for productive scenarios. To get a certificate signed by a CA, you first need to create a certificate signing request.

**Context**

To generate a certificate signing request (CSR), perform the following steps:

**Procedure**

1. Open the keystore using the KeyStore Explorer.
2. To create a certificate signing request (CSR), select the certificate and, in the context menu, choose **Generate CSR**.
3. Specify the details for the CSR.
   - Specify the following mandatory settings:
### Generate CSR

**Format:** PKCS #10

**Signature Algorithm:** Select the recommended algorithm.

**CSR File:** Specify the name and location of the CSR file.

The other settings (Challenge, Optional Company Name, Add certificate extensions to request) are optional.

---

4. A file with file extension `.csr` is generated (by default, the file name corresponds to the Common Name of the certificate).

Additional background information:

The Base64-encoded `.csr` file content looks like this:

```plaintext
-----BEGIN NEW CERTIFICATE REQUEST-----
MIIC2zCCAkAwggZnCAYGAEgcjZzV2A3Jb5yNoVBAiTRKRFHvZwGWgYDQQIDBNCYnYLMbiX77+9cnR0ZIw1ZJn
MRb5yDVQQHEhXiXYXwzG9y23EDMA0GAlUEChMGU0FQIEFHRw5yGVYDQQET5TVQgVHYJ1c3Qg
Q29tbXVuc21zYwYIAYDVQQDE1pbmRhYXM6b2Vyc3QgVHJ1c3Qufferable
KcIhvcnNAQEBBQADQgEFADCCQAQCCqEBANFxc501j2bg+apnnF57fuZC30mEGY03qckGcBwFzFV1
kE+fx3b4gddSSmQeGgqzoozOESIp0kgf3I2NyEzk894PLKyOdHkB6T9yKhryWFu1ZaaywA
Fwom61dpJ45qDXzhw1Lj6OQ1b6B3J2Oe1x1c7X7s5yOsMk1JEU21AsqU04CZwvQv0D3AEyHUE
pEhuq3E3bAFINzvk777xK6AQ3ucNjBuZCBuUJSK2bR0DtyQ7mvrU5lrwCaA/hbdGdju3k+ooZ
H4QB1F5swvU5616C3nn86vMNWC2C3I12YHqFVH1bwOxyYIUcM9nSMJcewX8bU1kehL4RkJcAwPA
AmAA0F3g3k3IDQEBQUA43REwQCCQci0bUqc3VKPi/qe08x4yCGA4sTDyv1QDCsQq02A5sVd
z61t6yiSm/TbRQyUQZM7h0pLwnGB1mnHbF9v9xb481L76GiDUTX6enAWP16tsEYy9swi47pF12
z1nCprV/I1leyAsjvKvDvYhovcJ06D5X7wKWE/03x8M/31qTnpz0911K/11/1UW6Y8e1LGb3D/AL73sE4e
ZO/I4sJ7Y7H67FADQfK50a7kQeksOE06c22aSoL++1HYLRG7z2aszU0jxKBUzLhIFK31qMEXc12
3CUltuBzC9dW8Ujf+ruP4NHvVwK2by7B8n8v5+1VDejz+Voyj/qbj2bH/05ubb+KQW
-----END NEW CERTIFICATE REQUEST-----
```

To display the decoded content of the CSR, choose **Examine CSR**.
Results

You have created a certificate signing request (CSR) and saved it on your computer. The next step is to send the CSR to a certification authority to get a signed certificate.

6.2.3 Requesting a Signed Certificate from a Certification Authority

To enable the tenant to communicate as client with the customer system, you have to import a client certificate to the tenant client keystore. This certificate has to be signed by a certification authority (CA).

Prerequisites

You have created a certificate signing request (CSR). Using this CSR, you request a signed certificate from a certification authority (CA).

Each CA has its own processes how to perform these steps. Check out the website of the CA for more information.

Context

Note that usually only authorized people can directly order a signed certificate from a CA as costs are involved.

6.3 Creating SFTP Keys

You have the option to set up reliable file transfer based on SSH File Transfer Protocol (SFTP). SFTP is an enhancement of the Secure Shell (SSH) network protocol.

Context

This section covers all steps to generate the required security artifacts for a tenant to be connected as SFTP client to an SFTP server.

ℹ️ Note

In particular, the following versions are supported: SSH version 2 (as specified in [http://tools.ietf.org/html/rfc4251](http://tools.ietf.org/html/rfc4251)) SSH File Transfer Protocol (sFTP) version 3 or higher
Procedure

1. Install a tool for SSH key management (for example, Cygwin).
2. Generate an SSH key pair (consisting out of private and public key) as part of a keystore.
   Use a separate key pair for each connected SFTP server.
   As keystore type use a Java keystore as described in the section on HTTPS transport-level security.
3. Create or update the `<known_hosts>` file.
4. Deploy the keystore and the `<known_hosts>` file to the tenant.
   Use the Deployed Artifacts editor of the Integration Operations perspective.
   ○ Select the Keystore artifact type and start the Deploy Content wizard to deploy the keystore (that contains the SSH private key).
   ○ Select the Known Host (SSH) artifact type and start the Deploy Content wizard to deploy the `<known_hosts>` file.

Next Steps

Related Information

Generating a Key Pair [page 49]

6.4 Creating Keys for Message Level Security

Related Information

Creating OpenPGP Keys [page 56]
Creating Keys for the Usage of PKCS#7, XML Digital Signature and WS-Security [page 73]

6.4.1 Creating OpenPGP Keys

You use the tool gpg4win to create the required keys for the usage of OpenPGP.
This section covers the creation of OpenPGP keys for tenants managed by SAP.
This description does not apply to tenants managed by customers. Customers might have their own OpenPGP key management processes.

The OpenPGP keys are maintained on the Windows VM on which the keys of the X.509 certificates are also maintained.

The kind of keys required depends on the use case and the role of the tenant for which the keys are created. The following table lists the possible use cases and the required kinds of keys.

### Note
As soon as you start gpg4win, files are created for the PGP Public Keyring and PGP Secret Keyring.

<table>
<thead>
<tr>
<th>Role of Tenant</th>
<th>Chosen Kind of Message Protection</th>
<th>Required Keys</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sender (outbound commu-</td>
<td>Encrypts outbound payload</td>
<td>PGP Public Keyring (contains receiver's public key to encrypt payload)</td>
</tr>
<tr>
<td>unication)</td>
<td>Encrypts and signs outbound</td>
<td>PGP Public Keyring (contains receiver's public key to encrypt payload)</td>
</tr>
<tr>
<td></td>
<td>payload</td>
<td>PGP Secret Keyring (contains tenant's secret key to sign payload)</td>
</tr>
<tr>
<td>Receiver (inbound com-</td>
<td>Decrypts inbound payload</td>
<td>PGP Secret Keyring (contains tenant's secret key to decrypt payload)</td>
</tr>
<tr>
<td>unication)</td>
<td>Decrypts and verifies inbound</td>
<td>PGP Secret Keyring (contains tenant's secret key to decrypt payload)</td>
</tr>
<tr>
<td></td>
<td>payload</td>
<td>PGP Public Keyring (contains the sender's public key to verify payload) for verifying</td>
</tr>
</tbody>
</table>

### Related Information

- How OpenPGP Works [page 104]
- Creating PGP Keys for Encryption (Tenant Is Sender) [page 58]
- Creating PGP Keys for Encryption and Signing (Tenant Is Sender) [page 59]
- Creating PGP Keys for Decryption (Tenant Is Receiver) [page 61]
- Creating PGP Keys for Decryption and Verifying (Tenant Is Receiver) [page 62]
- Securely Exchanging Key Material [page 74]
6.4.1.1 Creating PGP Keys for Encryption (Tenant Is Sender)

Prerequisites

You have installed gpg4win, created the tenant-specific directory, and created a key pair.

Context

For this use case, the following key artifact has to be deployed on the tenant:

A PGP Public Keyring that contains the receiver’s public key (required by the tenant to encrypt the payload)

The following figure shows the required entities to be configured for the tenant (on the left).

Procedure

1. Obtain the public key from the receiver.
   We recommend using a secure channel to ensure that the information originates from the correct source and that it has not been changed on its way. A signed email would be suitable, for example.
2. Import the receiver’s public key into the PGP Public Keyring.
3. If a secure channel has not been used to obtain the public key from the receiver, verify the fingerprint of the public key.
One option is to phone the owner of the public key and compare the fingerprint.

Next Steps

Using the Integration Operations feature, you deploy the PGP Public Keyring on the tenant.

Related Information

Installing gpg4win [page 64]
Creating Tenant-Specific File Directories [page 65]
Starting the GPA Tool [page 66]
Creating a Key Pair [page 68]
Importing a Public Key [page 71]
Securely Exchanging Key Material [page 74]

6.4.1.2 Creating PGP Keys for Encryption and Signing (Tenant Is Sender)

Prerequisites

You have installed gpg4win, created the tenant-specific directory, and created a key pair.

Context

For this use case, the following key artifacts have to be deployed on the tenant:

- A PGP Secret Keyring that contains the tenant’s private key (required by the tenant to sign the payload)
- A PGP Public Keyring that contains the receiver’s public key (required by the tenant to encrypt the payload)

The following figure shows the required entities to be configured for the tenant (on the left).
Procedure

1. Start the GPA tool and create a new key. This action creates a PGP Secret Keyring containing a private/public key pair.
2. Obtain the public key from the receiver. We recommend using a secure channel (for example, encrypted email) for this information exchange.
3. Import the receiver’s public key into the PGP Public Keyring.
4. If a secure channel was not used to obtain the public key from the receiver, verify the fingerprint of the public key.
5. Export the public key from the tenant’s PGP Public Keyring.
6. Provide the receiver with the public key (ideally through a secure channel). The receiver has to import the tenant’s public key into its PGP Public Keyring.

Next Steps

Using the Integration Operations feature, you deploy the PGP Public Keyring and the PGP Secret Keyring on the tenant.
Related Information

Installing gpg4win [page 64]
Creating Tenant-Specific File Directories [page 65]
Starting the GPA Tool [page 66]
Creating a Key Pair [page 68]
Importing a Public Key [page 71]
Exporting the Public Key [page 70]
Securely Exchanging Key Material [page 74]

6.4.1.3 Creating PGP Keys for Decryption (Tenant Is Receiver)

Prerequisites

You have installed gpg4win, created the tenant-specific directory, and created a key pair.

Context

For this use case, the following key artifact has to be deployed on the tenant:

A PGP Secret Keyring that contains the tenant's private key (required by the tenant to decrypt the payload)

The following figure shows the required entities to be configured for the tenant (on the right).

![Diagram showing the process of PGP keys for decryption](image-url)
Procedure

Start the GPA tool and create a new key.
This action creates a PGP Secret Keyring containing a private/public key pair.

Next Steps

Using the Integration Operations feature, you deploy the PGP Secret Keyring on the tenant.

Related Information

Installing gpg4win [page 64]
Creating Tenant-Specific File Directories [page 65]
Starting the GPA Tool [page 66]
Creating a Key Pair [page 68]
Securely Exchanging Key Material [page 74]

6.4.1.4 Creating PGP Keys for Decryption and Verifying (Tenant Is Receiver)

Prerequisites

You have installed gpg4win, created the tenant-specific directory, and created a key pair.

Context

For this use case, the following key artifacts have to be deployed on the tenant:

- A PGP Public Keyring that contains the sender’s public key (required by the tenant to verify the payload obtained from the sender)
- A PGP Secret Keyring that contains the tenant’s private key (required by the tenant to decrypt the payload obtained from the sender)

The following figure shows the required entities to be configured for the tenant (on the right).
Procedure

1. Start the GPA tool and create a new key.
   This action creates a PGP Secret Keyring containing a private/public key pair.
2. Obtain the public key from the sender.
   We recommend using a secure channel (for example, encrypted email) for this information exchange.
3. Import the sender’s public key into the PGP Public Keyring.
4. If a secure channel was not used to obtain the public key from the sender, verify the fingerprint of the public key.
5. Export the public key from the tenant’s PGP Public Keyring.
6. Provide the sender with the public key (ideally through a secure channel).
   The sender has to import the tenant’s public key into its PGP Public Keyring.

Next Steps

Using the Integration Operations feature, you deploy the **PGP Public Keyring** and the **PGP Secret Keyring** on the tenant.
6.4.1.5 Using gpg4win to Create PGP Keys

6.4.1.5.1 Installing gpg4win

We recommend that you use gpg4win to create OpenPGP key material.

Context

Procedure

2. Select the following components:
   - GnuPG (GNU Privacy Guard - a command line tool)
   - GPA (Gnu Privacy Assistant - provides a user interface to create and manage PGP keys)
   - Gpg4win Compendium (contains the documentation of the tool)
The following figure shows the screen for the installation procedure.

3. Finish the installation procedure.

6.4.1.5.2 Creating Tenant-Specific File Directories

A PGP Secret Keyring and a PGP Public Keyring have to be maintained for each tenant that uses OpenPGP. The GPA tool cannot maintain several PGP Secret or Public Keyrings at the same time. Therefore, you have to create a separate directory for each tenant, where you have to configure GPA and the launching of GPA separately (otherwise, keys from different tenants will be stored in the same keyring).

Context

The following procedure shows how you can achieve the described setup using Gnu Privacy Assistant. To facilitate the usage of the software, we provide a set of simple configuration files to download.

⚠️ Caution

The following description, together with the configuration files, show a possible way how to use Gnu Privacy Assistant. We cannot give any guarantee that the software (in combination with the configuration files) works in the desired way.

Start Gnu Privacy Assistant (separately for each tenant).
Procedure

1. For each tenant (using OpenPGP), create a separate file directory for maintaining the keyrings.
2. Copy the following three files into this file directory:
   - gpa.conf
   - gpg.conf
   - run_gpa.bat

   You can download the files [here](#).

   Download the .zip file and extract the content on your computer.

3. Adapt the file `run_gpa.bat` by entering the path to the tenant-specific directory.

   These files are required to configure the usage of the GPA tool.

   The file `run_gpa.bat` sets the shell variable GNUPGHOME to the tenant-specific directory.

   The files `gpa.conf` and `gpg.conf` contain configurations for GPA and GPG. The file `gpg.conf`, for example, determines the strength of the applied encryption. Read the comments in the configuration files for further details.

Next Steps

6.4.1.5.3   Starting the GPA Tool

Context

Procedure

Double-click the `run_gpa.bat` file in the relevant tenant-specific directory.

If you start GPA without executing `run_gpa.bat`, gpa will use the default GNUPGHOME directory.

The following figure shows the user interface of the tool.
Connecting a Customer System to SAP Cloud Platform Integration

Detailed Steps
Next Steps

As soon as you have started the GPA tool, the following files are created for the PGP Public Keyring and PGP Secret Keyring: `pubring.gpg` and `secring.gpg` (see the following screenshot of the tenant-specific directory after tool launch).

These files have to be deployed later on the tenant as PGP Public Keyring and PGP Secret Keyring.

6.4.1.5.4 Creating a Key Pair

Context

OpenPGP provides the option of defining two kinds of keys: primary keys and subkeys. There is no general recommendation for when to use which type.

Usually, a primary key is created for certification and signing, and a subkey is created for encryption for each tenant that uses OpenPGP, but this is just a recommendation.

Procedure

1. Start the GPA tool (by double-clicking `run_gpa.bat` in the tenant-specific directory).
2. In the menu, select \[ New \] \[ New Keys \] .
3. In the Generate Key dialog, keep the Algorithm and Key Size (RSA, 2048), and specify the following attributes.
For Name, enter a string according to the following naming convention:

<speaking tenant name> <tenant alias>.hci.sap.com

For <speaking tenant name>, you can use the name of the company, for example (like Citi).

Leave the Email and Comment fields empty.

Select Expires and chose a period of 2 years.

4. Choose OK.

5. Enter a password.

   Note that all private keys in the secret keyring must have the same password.

6. The key is generated.

   If you select the key entry, more details are displayed.
On the **Subkeys** tab, the usage of the related subkeys is displayed.

### 6.4.1.5.5 Exporting the Public Key

You can export a public key in order to make it available for your communication partner (sender or receiver).

#### Context

#### Procedure

1. Start the GPA tool and select the key that is to be exported.
2. Choose **Export**.
3. Select a location on your local disk and specify a file name (extension `.pub`).
4. Choose Save.

**Results**

When you open the public key file with a text editor, it looks like this (example):

```
-----BEGIN PGP PUBLIC KEY BLOCK-----
Version: GnuPG v2.0.22 (MingW32)
mQENBFN01NoBCAcn3GcPFGBJ1pLcVanYq2/oWiuwmd1c2ZWos+rQnyECl2tr3DN/dJK1lpMI4JwaE8ztsTRnkK30DPaDdHWLOeuVm61Utoa5tNdoa0tpdtXdx3dJ7SfIN21Mkd++ttrTwhKGWzY1uqF2P30eKmJVeBMCz/v55JH0THFHLKbdfKsGkprKm+d3TIwD9V2aX11xk6Pm1qyWSTb19eq078V2NOIDgPSRA2V9Qa+B0qdAQClHnbJn
9SLxkh94mk3SlkhayuM0MPPK17H/7al5uqOBiyjAczEGJxzpjr6sYVUfPoQREG
AK5+Wekkhgpa2n39IHUYTdhleTgpMxpsgrzABEBAAG0JVB1dGVyIEd1dHNjaGUg
PHBldGVyLmd1dHNjaGVAc2FwLmVybT6JATkEaECAMFA1N01NoCGw8HCwIBwMC
AQYVCA1JCgsEFgIDAQIEAQIIXAAARECzSBNhmXnBSeAVB/OTWgHEE74ZElgIIBW
vgOxZ2VPGxh8AN+2TimeBvXZ564DucU3RU0CCvdAAth76rj9eessAsVndIzW8k
7C10Tmrq0OmX4oMTJy94KbR3i1HwiD+yfynmL8E0QSTQwUBcHogTFjXK64qGe6UX
CscWfHClu1foK4N18KAJxFBo371d74ZmoJRljqcDsbHeeCB88Jv015pETBW
ns4abVj88sdVNSigmt7R64mkTMK0iaJ6NL958rfJ1Q21Eus8Z1WtcBoYLSa5JxSgB
9XgT=jEwk
-----END PGP PUBLIC KEY BLOCK-----
```

### 6.4.1.5.6 Importing a Public Key

You can import public keys provided by your communication partner.

**Context**

The administrators of the sender or receiver system provide the public keys that need to be imported into the tenant's PGP Public Keyring.

**Procedure**

1. You obtain the public key from the sender or receiver administrator (either by e-mail or by download from a key server).
2. Start the GPA tool and select the key.
3. Choose Import.
4. Browse for the key on your local disk and add it to your keyring.
5. After the import, verify the fingerprint of the imported key.
This is important because the key could have been tampered with during its transfer from the sender or receiver.

The fingerprint is displayed in the GPA tool on the Details tab.

One option to verify the correctness of the fingerprint is to contact the sender/receiver administrator by phone or signed e-mail and ask whether the fingerprint is correct.

### 6.4.1.5.7 Using the GNU Privacy Guard Command Line Tool

The GNU Privacy Guard command line tool provides additional functions for working with OpenPGP keys.

The CPA graphical tool only contains a subset of functions that might be relevant when configuring scenarios using OpenPGP. Some use cases might require you to remove a subkey or add a new subkey. This can only be done with the command line tool.

When using the command line tool, make sure that you always specify the tenant home directory in the commands, in order to make changes for a specific tenant.

Example:

```
gpg --homedir=C:/tenantMyCompany --edit-key MyCompany
```
This command edits the key in the tenant directory `C:/tenantCiti` that contains the string `Citi` in its user ID.

To consult the manual for further details, run the command: `gpg --help`.

### 6.4.2 Creating Keys for the Usage of PKCS#7, XML Digital Signature and WS-Security

To set up message level security scenarios based on PKCS#7, XML Digital Signature or WS-Security, the required keys are created in the same way as for transport level security HTTPS.

Setting up message level security based on PKCS#7, XML Digital Signature or WS-Security requires the generation of public-private key pairs of type `X.509` – the same standard as is used for transport level security SSL.

Therefore, technically, you can use the same public key pairs for message level and transport level security (HTTPS).

Depending on the scenario, however, separate key pairs might be required.

Keep in mind that you can set up message level security on top of another transport security (like, for example SFTP). In that case, you in any case have to generate key pairs based on X.509 standard.

To generate a new public-private key pair, proceed as described for transport level security SSL. In particular, proceed in the same way as described for the configuration of certificate-based outbound authentication (HTTPS).

Note the following in addition:

- If you have already generated a keystore file and a separate key pair should be used for message level security, you can use the same keystore file, import the certificates required for message level security, and re-deploy the keystore file on the relevant tenant.
- To implement digital signature, a certificate from the sender is needed (the public key of the sender is required to verify the signature – in other words, to decrypt the digest).
- To implement digital encryption, a certificate from the receiver is needed (the public key of the receiver is required to encrypt the symmetric encryption key).

### Related Information

- Message-Level Security [page 97]
- Creating X.509 Keys [page 49]
6.5 Securely Exchanging Key Material

In many cases, communication partners need to exchange public keys in order to establish a secure connection.

To establish a secure communication between software systems, communication partners use asymmetric (or public) key technology and work with private/public key pairs. In some cases, public keys have to be exchanged between the partners at certain points of the configuration process.

You need to apply certain measures when exchanging key material to ensure that you do not compromise the security of your scenario.

Public Keys

When exchanging public keys (for example, X.509 certificates), make sure that the keys cannot be manipulated by a third party during the transfer.

You have the following options:

- Use a secure communication channel for the key exchange. For example, you can use PGP-encrypted and -signed e-mail or a secure collaboration platform like SAP Jam.
- Verify the sender (for example, using a signature) and check whether the sender is authorized to provide this key material.
- Verify that the content was not manipulated (usually using a signature).

If you can’t use a secure communication channel, check the integrity of the keys by other means, such as the following:

- In the case of X.509 certificates, check that the certificate is valid and that it has been issued by a trusted certification authority (CA).
- Use a separate communication channel (for example, phone) to verify the fingerprint of the key with the sender.

Private Keys

Private keys are even more sensitive than public keys. Sharing your private key with others will allow them to read your encrypted messages and sign messages with your signature.

Whenever possible, avoid exchanging private key material.

In exceptional cases where you have to exchange private keys, apply one of the following measures:

- Use an encrypted container with a password (like PKCS#12 or Java Keystore).
- Transfer the password through a separate communication channel (for example, phone).
- Use secure communication channels. Never use plain e-mail or plain HTTP.
- SAP can provide you with a process for exchanging keys in a secure manner.
Example: Exchanging a Public Key

To configure a scenario that includes digital encryption of message content, the following main steps are required, including the exchange of a public key:

1. A potential receiver R of the message generates a public/private key pair (that contains the receiver’s public key PubKey_R and the associated private key PrivKey_R).
2. R provides a potential sender S of messages with the public key PubKey_R. To do this, R communicates with S using a private SAP Jam group that is only accessible for dedicated people associated with R and S. However, R does not disclose the private key.
3. S imports PubKey_R into the keystore of the software system that is involved in the scenario on the sender side.
4. S encrypts the message with public key PubKey_R and sends the encrypted message to the receiver.
5. R decrypts the message with the private key PrivKey_R.
7 Concepts of Secure Communication

There are several options to protect the message exchange. You can secure the communication on transport level by selecting the HTTPS or SFTP protocol and installing specific authentication methods. In addition to that, you can set up methods to encrypt and decrypt the content of the message and to digitally sign and verify the message.

Related Information

Basics [page 76]
Security Elements [page 117]

7.1 Basics

Related Information

HTTPS-Based Communication [page 76]
SFTP-Based Communication [page 95]
Message-Level Security [page 97]
Certificate Management [page 110]

7.1.1 HTTPS-Based Communication

Related Information

Technical Landscape [page 77]
Authentication and Authorization Options (Inbound) [page 78]
Authentication Options (Outbound) [page 90]
7.1.1.1 Technical Landscape

The following figures illustrate the general set up of components and communication paths.

Figure 1: Inbound Communication

For inbound SSL communication, a load balancer is interconnected between the sending customer back-end system and the tenant. Load balancer terminates inbound SSL connection (and starts a new one inside the VM landscape of SAP).

On the outbound side, a customer-specific tenant is connected to a customer back-end system. Therefore, all security settings for outbound message processing have to be configured tenant-specific.

The terms *inbound* and *outbound* reflect the perspective of SAP throughout this documentation.

- Inbound refers to message processing from a customer system to SAP (load balancer).
- Outbound refers to message processing from SAP (tenant) to a customer system.
7.1.1.2 Authentication and Authorization Options (Inbound)

When a client calls a server using a secure communication channel, two different kinds of checks are performed subsequently.

- **Authentication**
  - Verifies the identity of the calling entity.

- **Authorization**
  - Checks what a user or other entity is authorized to do (for example, as defined by roles assigned to it). In other words, the authorization check evaluates the access rights of a user or other entity.

When a client calls a server, it is first authenticated and, in a subsequent step, the authorization check is performed.

We use **inbound** to refer to the communication direction when a sender system sends a message to the integration platform.

**Combinations of Authentication and Authorization (Inbound)**

Authentication and authorization options can be combined in a specific way for inbound communication.

The following table shows which combinations of authentication and authorization options are supported for inbound calls.

<table>
<thead>
<tr>
<th>Authentication Option ...</th>
<th>Can Be Used with the Following Authorization Option ...</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Basic authentication</strong></td>
<td>Role-based authorization</td>
</tr>
<tr>
<td>The sender (client) ...</td>
<td></td>
</tr>
<tr>
<td>The HTTP header of the</td>
<td></td>
</tr>
<tr>
<td>inbound message (from the</td>
<td></td>
</tr>
<tr>
<td>sender) contains the user</td>
<td></td>
</tr>
<tr>
<td>name and password.</td>
<td></td>
</tr>
<tr>
<td>For this user, the ...</td>
<td></td>
</tr>
<tr>
<td>authorizations are checked</td>
<td></td>
</tr>
<tr>
<td>based on user-to-role</td>
<td></td>
</tr>
<tr>
<td>assignments defined on the</td>
<td></td>
</tr>
<tr>
<td>tenant. To authorize a</td>
<td></td>
</tr>
<tr>
<td>sender system to process</td>
<td></td>
</tr>
<tr>
<td>messages on a tenant, the</td>
<td></td>
</tr>
<tr>
<td>role ESBMessaging.send</td>
<td></td>
</tr>
<tr>
<td>has to be assigned to the</td>
<td></td>
</tr>
<tr>
<td>associated user.</td>
<td></td>
</tr>
</tbody>
</table>
Authentication Option ...

| Client-certificate authentication and certificate-to-user mapping | Can Be Used with the Following Authorization Option ...
|---------------------------------------------------------------|-------------------------------------------------------------|
| The sender (client) authenticates itself against the server based on a digital client certificate. Furthermore, this certificate is mapped to a user (based on the information contained in a Certificate-to-User Mapping artifact deployed on the tenant). | Role-based authorization
For the user derived from the certificate-to-user mapping, the authorizations are checked based on user-to-role assignments defined on the tenant. To authorize a sender system to process messages on a tenant, the role ESBMessaging.send has to be assigned to the associated user. |

| Note |
| You can map multiple certificates to the same user (n:1 certificate-to-user mappings possible). |

| Client-certificate authentication (without certificate-to-user mapping) | Role-based authorization
Subject/Issuer DN authorization check of a certificate
In a subsequent authorization check, the permissions of the sender are checked on the tenant by evaluating the distinguished name (DN) of the client certificate of the sender. |

| OAuth |
| OAuth allows you to set up authentication scenarios without the need to share credentials. |
| More information on the concepts:
Protecting Applications with OAuth 2.0
OAuth 2.0 Specification |

| Note |
| This option is supported for the following sender adapter types: SOAP (SOAP 1.x), SOAP (SAP RM), HTTPS. |

Related Information

- Authentication Options (Inbound) [page 80]
- Authorization Options (Inbound) [page 89]
7.1.1.2.1 Authentication Options (Inbound)

For inbound communication, different ways are supported how the sender can authenticate itself against Cloud Integration.

We use **inbound** to refer to the communication direction when a sender system sends a message to the integration platform.

- **Basic authentication**
  The calling entity is authenticated based on credentials (user name and password)

- **Client-certificate authentication and certificate-to-user mapping**
  The calling entity is authenticated based on a certificate, and the certificate is mapped to a user (for which the authorization check is executed in a subsequent step).

- **Client-certificate authentication (without certificate-to-user mapping)**

- **OAuth 2.0**
  OAuth allows you to set up authentication scenarios without the need to share credentials.
  More information on the concepts:
  - Protecting Applications with OAuth 2.0
  - OAuth 2.0 Specification

**Related Information**

- Basic Authentication [page 80]
- Client Certificate Authentication and Certificate-to-User Mapping (Inbound) [page 83]
- Client Certificate Authentication (Inbound) [page 86]
- Inbound/HTTPS/OAuth [page 24]

7.1.1.2.1.1 Basic Authentication

Basic authentication allows a client to authenticate itself against the server based on user credentials (SCN user name and password).

⚠️ **Caution**

Consider that we do **not** recommend to use this option in productive scenarios because of the following security aspects:

Basic authentication has the risk that authentication credentials, for example, passwords, are sent in clear text. Using TLS (transport-layer security, also referred to as Secure Sockets Layer) as transport-level encryption method (when using HTTPS as protocol) makes sure that this information is nevertheless encrypted on the transport path. However, the authentication credentials might become visible to SAP-internal administrators at points in the network where the TLS connection is terminated, for example, load balancers. If logging is not done properly at such devices, the authentication credentials might become part of log files. Also network monitoring tools used at such devices might expose the authentication information to administrators. Furthermore, the person to whom the authentication credentials belong (in the example above, the password owner) needs to maintain the password in a secure place.
How it Works

The following figure shows the setup of components required for inbound basic authentication.

![Figure showing the setup of components required for inbound basic authentication.]

These are the steps at runtime:

The HTTP header of the inbound message (from the sender) contains user name and password. To protect these credentials during the communication step, the connection is secured using TLS (SSL).

This includes a step where the load balancer authenticates itself as server against the sender based on a certificate. To enable this security measure, the keystore of the load balancer contains a server certificate signed by a certification authority. To be more precise, the keystore of the load balancer contains a complete certificate chain from (including all intermediate certificates). On the other side of the communication, the keystore of the connected sender system must contain the load balancer server root certificate. That is the certificate that identifies the certification authority (CA) that signed the load balancer’s server certificate (on top of the certificate chain).

The other way round, the identity of the sender is checked by SAP evaluating the credentials (user and password) against the user stored in the SCN database.

It is also depicted in the figure that the authentication option needs to be activated for the corresponding integration flow.

Required Security Material

To enable the sender system to authenticate itself against the integration platform with basic authentication, a communication user has to be created for the sender. An SCN (dialog) user of the customer has to be re-used as the communication user.

The following figure provides an overview of the involved security artifacts and storage locations.
### Table 11: Certificates for Inbound Message Processing

<table>
<thead>
<tr>
<th>Keystore</th>
<th>Certificate</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sender keystore</td>
<td>Load balancer server root certificate (identifies CA that has signed the load balancer server certificate)</td>
<td>This certificate is required to identify the root CA at the top of the certificate chain that ultimately guarantees the trustability of the load balancer server certificate. In many cases, there is a multi-level setup of CAs so that a certificate is signed by an intermediate CA. The trustability of the intermediate CA is guaranteed by another intermediate CA one level higher, and so on, up to the root CA at the top of the certificate chain. In this case, it is best practice to assign the certificate chain to the certificate, to enable the connected component (which has imported only the root CA into its keystore) to evaluate the chain of trust.</td>
</tr>
<tr>
<td>Load balancer keystore</td>
<td>Load balancer server certificate</td>
<td>This certificate is required to identify the load balancer as a trusted server (to which clients like the sender system can connect). This certificate is required for certificate-based authentication where the sender acts as the client. On the tenant side, this certificate is required to configure the authorization check.</td>
</tr>
</tbody>
</table>
### Keystore Description

<table>
<thead>
<tr>
<th>Keystore</th>
<th>Certificate</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tenant keystore</td>
<td>Client certificate accepted by load balancer</td>
<td>A tenant keystore with this certificate is also required for inbound basic authentication as part of an automatic, regular connectivity tests. The basic technical connectivity of a cluster is checked on a regular basis, as soon as the cluster is active. For this purpose, every 30 seconds the tenant management node sends an HTTPS request to an assigned runtime node via the load balancer. This simulates an external call to the runtime node. To enable this communication, a keystore needs to be deployed on the tenant, containing a valid client certificate that is accepted by the load balancer as well as the root certificate of the same. If this keystore is not available or contains an invalid certificate, the cluster will raise an error.</td>
</tr>
</tbody>
</table>

### 7.1.1.2.1.2 Client Certificate Authentication and Certificate-to-User Mapping (Inbound)

This option includes an authentication step based on a digital client certificate and the mapping of the certificate to a user.

With a certificate-to-user mapping, a certificate is mapped to a user, and that way the user can be authenticated based on a certificate.

**Note**

Note that multiple certificates can be mapped to one user (n:1 certificate-to-user mappings possible).

Certificate-to-user mappings make sure that a user is always associated with the certificate as a whole, not only with one attribute of it (for example the common name (CN)). As different certificates can have the same CN, mapping only the CN to a user name bears the risk that different certificates can be mapped accidentally to the same user. Using certificate-to-user mappings circumvents this risk.

For the user defined that way, in a subsequent step, an authorization step is being executed.

### How it Works

The following figure shows the complete setup of components and security artifacts required for this option.
When you have configured this authentication option, the authentication of the user is performed in the following way at runtime:

The TLS connection of the sender system and the integration platform is terminated and newly established by the load balancer. This means, that first the load balancer authenticates itself against (as server) the sender based on the load balancer server certificate. Vice versa, the sender authenticates itself against the load balancer as client using the sender client certificate.

To enable the sender to communicate that way with the load balancer, the sender administrator has to make sure that the sender client certificate is signed by one of the certification authorities that are supported by the load balancer.

The load balancer sets the following message header fields:

- **SSL_CLIENT_CERT**
  Contains the Base64-encoded sender client certificate.
- **SSL_CLIENT_USER**

When the authentication is been executed successfully, the load balancer writes the sender client certificate (base 64-encoded) into the message header (field SSL_CLIENT_CERT). The tenant then maps the sender client certificate to a user based on the certificate-to-user mapping which is deployed on the tenant.

**Note**

In a subsequent step, the authorization check for this user is then executed based on the user-to-role assignments (permissions) as configured in the Central Authorization Service (CAS).
## Required Security Material

### Table 12: Certificates for Inbound Message Processing

<table>
<thead>
<tr>
<th>Keystore</th>
<th>Certificate</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sender keystore</td>
<td>Sender client certificate (public and private key; signed by CA with which the tenant has a trust relationship)</td>
<td></td>
</tr>
</tbody>
</table>
|                     | Load balancer server root certificate (identifies CA that has signed the load balancer server certificate) | This certificate is required to identify the root CA at the top of the certificate chain that ultimately guarantees the trustability of the load balancer server certificate.  
In many cases, there is a multi-level setup of CAs so that a certificate is signed by an intermediate CA. The trustability of the intermediate CA is guaranteed by another intermediate CA one level higher, and so on, up to the root CA at the top of the certificate chain. In this case, it is best practice to assign the certificate chain to the certificate, to enable the connected component (which has imported only the root CA into its keystore) to evaluate the chain of trust. |
| Load balancer keystore | Load balancer server certificate                  | This certificate is required to identify the load balancer as a trusted server (to which clients like the sender system can connect). This certificate is required for certificate-based authentication where the sender acts as the client. On the tenant side, this certificate is required to configure the authorization check. |
|                     | Sender client root certificate                    | This certificate is required to identify the root CA at the top of the certificate chain that ultimately guarantees the trustability of the sender client certificate. There is a list of CAs that are supported by the load balancer. |
### Tenant keystore

Client certificate accepted by load balancer

<table>
<thead>
<tr>
<th>Keystore</th>
<th>Certificate</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tenant keystore</td>
<td>Client certificate accepted by load balancer</td>
<td>A tenant keystore with this certificate is also required for inbound basic authentication as part of an automatic, regular connectivity tests. The basic technical connectivity of a cluster is checked on a regular basis, as soon as the cluster is active. For this purpose, every 30 seconds the tenant management node sends an HTTPS request to an assigned runtime node via the load balancer. This simulates an external call to the runtime node. To enable this communication, a keystore needs to be deployed on the tenant, containing a valid client certificate that is accepted by the load balancer as well as the root certificate of the same. If this keystore is not available or contains an invalid certificate, the cluster will raise an error.</td>
</tr>
</tbody>
</table>

In addition to that, a certificate-to-user mapping has to be defined for the tenant.

### 7.1.1.2.1.3 Client Certificate Authentication (Inbound)

This option includes an authentication step based on a digital client certificate.

#### How it Works

The following figure shows the complete setup of components and security artifacts required for this option.
When you have configured this authentication option, the authentication of the user is performed in the following way at runtime:

The TLS connection of the sender system and the integration platform is terminated and newly established by the load balancer. This means, that first the load balancer authenticates itself against (as server) the sender based on the load balancer server certificate. Vice versa, the sender authenticates itself against the load balancer as client using the sender client certificate.

To enable the sender to communicate that way with the load balancer, the sender administrator has to make sure that the sender client certificate is signed by one of the certification authorities that are supported by the load balancer.

The load balancer sets the following message header fields:

- **SSL_CLIENT_CERT**
  - Contains the Base64-encoded sender client certificate.
- **SSL_CLIENT_USER**

### Required Security Material

**Table 13: Certificates for Inbound Message Processing**

<table>
<thead>
<tr>
<th>Keystore</th>
<th>Certificate</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sender keystore</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Keystore</td>
<td>Certificate</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------</td>
<td>---------------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Load balancer server root certificate</td>
<td>This certificate is required to identify the root CA at the top of the certificate chain that ultimately guarantees the trustability of the load balancer server certificate. In many cases, there is a multi-level setup of CAs so that a certificate is signed by an intermediate CA. The trustability of the intermediate CA is guaranteed by another intermediate CA one level higher, and so on, up to the root CA at the top of the certificate chain. In this case, it is best practice to assign the certificate chain to the certificate, to enable the connected component (which has imported only the root CA into its keystore) to evaluate the chain of trust.</td>
<td></td>
</tr>
<tr>
<td>Load balancer server certificate</td>
<td>This certificate is required to identify the load balancer as a trusted server (to which clients like the sender system can connect). This certificate is required for certificate-based authentication where the sender acts as the client. On the tenant side, this certificate is required to configure the authorization check.</td>
<td></td>
</tr>
<tr>
<td>Sender client root certificate</td>
<td>This certificate is required to identify the root CA at the top of the certificate chain that ultimately guarantees the trustability of the sender client certificate. There is a list of CAs that are supported by the load balancer.</td>
<td></td>
</tr>
</tbody>
</table>

**Note**

In a subsequent authorization check, the permissions of the sender are checked on the tenant by evaluating the distinguished name (DN) of the client certificate of the sender. The client certificate of the sender is being passed through to the tenant by the load balancer (in the message header). To provide the tenant with the information on the correct client certificate to be expected from the sender, a corresponding setting has to be made in the related integration flow.
7.1.1.2.2 Authorization Options (Inbound)

For inbound HTTPS requests, two different ways to check the authorization of the caller can be configured.

We use inbound to refer to the communication direction when a sender system sends a message to the integration platform.

- **Role-based authorization**
  - The permissions of the calling entity (user) are checked based on a user-to-role assignments configured in the associated identity provider.
  - In the related sender adapter, you can assign the role based on which the inbound authorization is to be checked for the integration flow.

- **Subject/Issuer DN authorization check**
  - The distinguished name (DN) of a certificate (associated with the calling entity) is checked.
  - Subject/Issuer DN authorization check can be defined for individual integration flows.

Related Information

Role-Based Authorization [page 89]
Subject/Issuer DN authorization check [page 89]

7.1.1.2.2.1 Role-Based Authorization

This option allows you to define permissions for users in the connected identity provider (by default, SAP Identity Service) and to perform an authorization check based on these settings.

For HTTPS requests sent to Cloud Integration, it is checked if the role ESBMessaging.send is assigned to the user.

The permissions of the sending client are checked according to roles assigned to the user in the associated identity provider.

User management (which includes the assignment of permissions to users) is performed by the tenant administrator using the SAP Cloud Platform Cockpit.

7.1.1.2.2.2 Subject/Issuer DN authorization check

It is checked (for a specific integration flow) if the subject/issuer distinguished name (DN) of the assigned certificate matches the incoming certificate.

If yes, this specific integration flow can be processed. The authorization check is performed based on the distinguished name (DN) of the client certificate. The DN has to be specified when configuring the relevant integration flow.
7.1.1.3 Authentication Options (Outbound)

For outbound communication through HTTPS (when the tenant sends a message to a receiver), the following authentication options are supported.

- **Basic authentication**
  The calling entity (tenant) is authenticated based on credentials (user name and password)

- **Client-certificate authentication**
  The calling entity (tenant) is authenticated based on a certificate.

Related Information

- Basic Authentication [page 90]
- Client Certificate Authentication (Outbound) [page 91]

7.1.1.3.1 Basic Authentication

Basic authentication allows a the tenant to authenticate itself against the receiver through credentials (user name and password).

How it Works

The following figure shows the setup of components required for this authentication option.
Basic authentication for HTTPS-based outbound calls works the following way:

1. The tenant (client) sends a message to the customer back-end system.
   The HTTP header of the message contains user credentials (name and password).
   To protect the user credentials during the communication step, the connection is secured using SSL.
2. The customer back-end authenticates itself as server against the tenant using a certificate (the customer back-end identifies itself as trusted server).
   To support this, the keystore of the customer back-end system must contain a server certificate signed by a certification authority. To be more precise, the keystore must contain the complete certificate chain. On the other side of the communication, the keystore of the connected tenant must contain the customer back-end server root certificate.
3. The tenant is authenticated by the customer back-end by evaluating the credentials against the user stored in a related data base connected to the customer back-end.

**Required Security Material**

Table 14: Certificates for Outbound Message Processing

<table>
<thead>
<tr>
<th>Keystore</th>
<th>Security Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Keystore (tenant-specific)</td>
<td>Receiver server root certificate</td>
<td>This certificate is required to identify the root CA that is at the top of the certificate chain that ultimately guarantees the trustability of the receiver server certificate.</td>
</tr>
<tr>
<td>Receiver keystore</td>
<td>Receiver server certificate (signed by CA with which the tenant has a trust relationship)</td>
<td>This certificate is required to identify the receiver (to which the tenant connects as the client) as a trusted server.</td>
</tr>
<tr>
<td>User credentials artifact</td>
<td>User and password</td>
<td>With these credentials the tenant authenticates itself as client at the receiver system.</td>
</tr>
</tbody>
</table>

**7.1.1.3.2 Client Certificate Authentication (Outbound)**

The following figure shows the setup of components required for this authentication option.
How it Works

The tenant authenticates itself against the receiver based on a certificate.

This authentication option works the following way:

1. The tenant sends a message to the receiver.
2. The receiver authenticates itself (as trusted server) against the tenant when the connection is being set up.
   In this case, the receiver acts as server and the authentication is based on certificates.
3. Authentication of the tenant: The identity of the tenant is checked by the receiver by evaluating the client certificate chain of the tenant.
   As prerequisite for this authentication process, the client root certificate of the tenant has to be imported into the receiver keystore (prior to the connection set up).
   As CA who provides the root certificate, Cyber trust Public Sure Server SV CA is used.
   Steps 2 and 3 are referred to as mutual SSL handshake.
4. Authorization check: The permissions of the client (tenant) are checked in a subsequent step by the receiver.
**Required Security Artifacts**

Table 15: Certificates for Outbound Message Processing

<table>
<thead>
<tr>
<th>Keystore</th>
<th>Certificate</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Keystore (tenant-specific)</td>
<td>Tenant client certificate</td>
<td>This certificate is required to authenticate the tenant when calling the receiver system as the client. The certificate contains the public and private keys and has the certificate chain assigned to it.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Note</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>In many cases, there is a multi-level setup of CAs so that a certificate is signed by an intermediate CA. The trustability of the intermediate CA is guaranteed by another intermediate CA one level higher, and so on, up to the root CA at the top of the certificate chain. In this case, it is best practice to assign the certificate chain to the certificate, to enable the connected component (which has imported only the root CA into its keystore) to evaluate the chain of trust.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The keystore is deployed on the tenant.</td>
</tr>
<tr>
<td>Receiver server root certificate</td>
<td></td>
<td>This certificate is required to identify the root CA that is at the top of the certificate chain that ultimately guarantees the trustability of the receiver server certificate.</td>
</tr>
<tr>
<td>Receiver keystore</td>
<td>Receiver server certificate (signed by CA with which the tenant has a trust relationship)</td>
<td>This certificate is required to identify the receiver (to which the tenant connects as the client) as a trusted server.</td>
</tr>
<tr>
<td></td>
<td>Tenant client root certificate (identifies CA that has signed the tenant client certificate)</td>
<td>This certificate is required to identify the root CA that is at the top of the certificate chain that ultimately guarantees the trustability of the tenant client certificate.</td>
</tr>
</tbody>
</table>
7.1.1.4 Load Balancer Root Certificates Supported by SAP

The load balancer supports a certain list of root certificates.

A system sending a message to the Cloud-based integration platform using HTTPS as secure transport channel is not directly connected to the tenant. Instead of this, a load balancer component is interconnected that terminates all inbound HTTPS requests, and re-establishes a new secure connection.

To set up a secure connection between a sender system and the integration platform, you therefore need to make sure that the sender system’s keystore contains a client certificate that is signed by one of those certification authorities (CAs) that are trusted by the load balancer component of SAP.

The following list summarizes the root certificates that are supported by the load balancer.

Note
A specific certificate that identifies a certification authority (CA) is referred to as root certificate. Such a certificate is typically not signed by any other authority, as it is at the root of a certificate chain.

The load balancer component is owned by SAP, and you, the customer, don’t need to care how it is configured. However, you need to make sure that the client certificate in your sender keystore is signed by one CA that is listed below.

- AddTrust External CA Root
- Baltimore CyberTrust Root
- Certum CA
- DigiCert High Assurance EV Root CA
- DigiCert Global Root CA
- GeoTrust Global CA
7.1.2 SFTP-Based Communication

7.1.2.1 How SFTP Works

A tenant can connect as SFTP client to an SFTP server (the latter either hosted at SAP or in the customer landscape).

In a scenario using SFTP, an SFTP client connects to an SFTP server in order to perform one of the following tasks:

- SFTP client writes (pushes) a file to a file directory on an SFTP server.
- SFTP client reads (pulls) data from the SFTP server.

Typically the SFTP client periodically reads files from the SFTP server.

Depending on the direction of data flow (whether the tenant reads data from the SFTP server or writes data to it), either an SFTP sender adapter or SFTP receiver adapter is involved.

Files are stored on the SFTP server in specific directories referred to as mailboxes. For each mailbox, a user is specified in order to control access to the data.

In certain cases, you have the option to choose between the following authentication options for SFTP connectivity in the SFTP (sender or receiver) adapter:
• User Name/Password
• Public Key

User Name/Password Authentication

The tenant connects to the server with a user and authenticates itself against the SFTP server with a password.

The user credentials (user name and password) are stored in a User Credentials artifact which has been deployed on the tenant prior to connection set up.

Public Key Authentication

In order to set up secure connection between the SFTP client and SFTP server, a combination of symmetric and asymmetric keys is applied.

• Symmetric (session) keys are used in order to encrypt and decrypt data within a data transfer session.
• Asymmetric key pairs (on client and server side) are used in order to encrypt and decrypt the session keys.

Symmetric and asymmetric keys are used by a client and a server exchanging data via SFTP in the following way:

1. The client connects to the server.
2. The server sends his public key to the client.
3. The client checks if the server is a trusted participant by evaluating a known_hosts file at client’s side: if the server’s public key is listed there-in, the identity of the server is confirmed.
4. The client generates a session key (to be used for one data transfer session).
5. The client encrypts the session key with the public key of the server.
6. The client sends the encrypted session key to the server. As public and private key of one party are mathematical correlated with each other, the server can decrypt the session key using its private key.
7. The session can now be continued in an encrypted way.
8. As part of the secure data transfer (using the session key exchanged by the step before), the client sends its public key to the server.
9. The server checks if the public key of the client is known to him (evaluating an authorized_keys file on the server side).
10. The server encrypts a random number with the client’s public key and sends it to the client.
11. The client decrypts the random number with its private key and sends the unencrypted random number back to the server. That way, the client authenticates itself on server side.
7.1.3 Message-Level Security

Several standards are supported to protect the message content (message-level security).

Message-level security features allow you to digitally encrypt/decrypt or sign/verify a message (or both). The following standards and algorithms are supported.

Table 16: Message-Level Security Options and Algorithms

<table>
<thead>
<tr>
<th>Security Standard/Option</th>
<th>Security Feature</th>
<th>Supported Algorithms</th>
</tr>
</thead>
<tbody>
<tr>
<td>PKCS#7/CMS Enveloped Data and Signed Data</td>
<td>Encryption/decryption of message content</td>
<td>The following algorithms for content encryption (by the symmetric key) are supported (format Cipher/Operation Mode/Padding Scheme): DESede/CBC/PKCS5Padding, DES/CBC/PKCS5Padding, AES/CBC/PKCS5Padding, ARCFour/ECB/NoPadding, Camellia/CBC/PKCS5Padding, RC2/CBC/PKCS5Padding, CAST5/CBC/PKCS5Padding.</td>
</tr>
<tr>
<td>PKCS#7/CMS Enveloped Data and Signed Data</td>
<td>Encryption/decryption and signing/verifying payload</td>
<td>The following algorithms for content encryption (by the symmetric key) are supported (format Cipher/Operation Mode/Padding Scheme): DESede/CBC/PKCS5Padding, DES/CBC/PKCS5Padding, AES/CBC/PKCS5Padding, ARCFour/ECB/NoPadding, Camellia/CBC/PKCS5Padding, RC2/CBC/PKCS5Padding, CAST5/CBC/PKCS5Padding. The signature algorithms: SHA512/RSA, SHA384/RSA, SHA256/RSA, SHA224/RSA, SHA/RSA, RIPEMD128/RSA, RIPEMD160/RSA, RIPEMD256/RSA, MD5/RSA. This is a subset of the algorithms that are supported for PKCS#7/CMS Enveloped Data and Signed Data. The generated signature does not conform to the CAdES-BES (CMS Advanced Electronic Signatures) signature standard.</td>
</tr>
<tr>
<td>Security Standard/Option</td>
<td>Security Feature</td>
<td>Supported Algorithms</td>
</tr>
<tr>
<td>------------------------------------------</td>
<td>---------------------------------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Basic Digital Signature Option (Simple Signer)</td>
<td>Signing/verifying payload</td>
<td>The following algorithms for content signing are supported (digest and encryption algorithm): SHA512/RSA, SHA384/RSA, SHA256/RSA, SHA224/RSA, SHA/DSA, RIPEMD128/RSA, RIPEMD160/RSA, RIPEMD256/RSA, MD5/RSA, MD2/RSA, RIPEMD160andMGF1/RSA-ISO9796-2-2-3, SHAandMGF1/RSA-ISO9796-2-2-3, SHA256withDSA, SHA224withDSA, SHA/DSA.</td>
</tr>
</tbody>
</table>
| Open Pretty Good Privacy (PGP)           | Encryption/decryption of message content | The following symmetric key algorithms for content encryption (symmetric key algorithms) are supported:  

- TripleDES (168bit key derived from 192), CAST5 (128 bit key, as per [RFC2144]), Blowfish (128 bit key, 16 rounds), AES with 128, 192, and 256-bit key, Twofish with 256-bit key  

DES is not supported. |
|                                         | Encryption/decryption and signing/verifying the message | The following hash algorithms are also supported for PGP signing:  

For DSA key: SHA-1, SHA224, SHA256, SHA384, SHA512  
For key: MD2, MD5, SHA-1, RIPE-MD/160 "RIPEMD160", SHA256, SHA384, SHA512 |
| XML Signature                           | Signing/verifying payload             | The following signature algorithms are supported when applying XML Signature: DSA/SHA1, RSA/SHA1, RSA/SHA256, RSA/SHA384, RSA/SHA512 |
| XML Advanced Electronic Signature (XAdES) | Signing payload                       | The same signature algorithms like for XML Signature are supported. |
| Supported XAdES forms: XAdES Basic Electronic Signature and XAdES Explicit Policy based Electronic Signature |                                     |                                                                 |
| WS-Security                             | Signing/verifying SOAP body          | The default signature algorithm is set by the data in the certificate, that is, one of the following: http://www.w3.org/2000/09/xmldsig#rsa-sha1 or http://www.w3.org/2000/09/xmldsig#dsa-sha1.  

The default signature digest algorithm is: http://www.w3.org/2000/09/xmldsig#sha1  

Strong encryption is supported for the following algorithms:  
- AES/CBC/PKCS5Padding  
- Camellia/CBC/PKCS5Padding  

For these algorithms, the key lengths 192 and 256 are possible.
Recommendations

Some algorithms (like MD2, MD5, DES or RC4) are still supported for legacy reasons, but they are not considered secure any more. We recommend that you check the official recommendations from National Institute of Standards and Technology (NIST) or European Union Agency for Network and Information Security (ENISA) for advice on algorithms and key strengths (for example, at: https://www.enisa.europa.eu/activities/identity-and-trust/library/deliverables/algorithms-key-sizes-and-parameters-report).

Related Information

How PKCS#7 Works [page 99]
How XML Signature Works [page 102]
How WS-Security Works [page 103]
How OpenPGP Works [page 104]

7.1.3.1 How PKCS#7 Works

You have the option to digitally sign and encrypt message payloads based on PKCS#7/CMS Enveloped Data and Signed Data (PKCS stands for Public Key Cryptography Standards).

Signing and Verifying a Message

A digital signature ensures the authenticity of a message that way that it guarantees the identity of the signer and that the message was not altered after signing.

Digitally signing a message works the following way:

1. The sender calculates out of the message content a digest (or hash value) using a digest algorithm.
2. The sender encrypts the digest using a private key (type RSA or DSA). This is actually the signing step.
   
   **Note**


3. The sender sends the encrypted digest (which corresponds to the signature) together with the message content to the receiver.
4. The receiver decrypts the digest with the public key (which is related to the senders’ private key). The public key has the type RSA or DSA.
5. The receiver calculates the digest out of the content of the message (which has been sent to it by the sender).
The receiver uses the same digest algorithm which the sender had used.

Note
PKCS#7 ensures that the digest algorithm is transferred together with the signature of the message and therefore available for the receiver.

This calculation is based on the message content. In case the message content has been transferred encrypted, a decryption step is needed before this step.

6. The receiver compares the decrypted digest (from the sender) with the one calculated at receiver side. In case both values (digests) are identical, the signature is verified.

The following figure illustrates the process of digitally signing and verifying a message.

### Encrypting and Decrypting the Content of a Message

Digital encryption allows you to encode the content of a message in such a way that only authorized parties can read it.

Digital encryption works two-stage based on symmetric and asymmetric key technology:

1. The sender encrypts the content of the message using a symmetric key.

Note
The following algorithms for content encryption (by the symmetric key) are supported (format Cipher/Operation Mode/Padding Scheme): DESede/CBC/PKCS5Padding, DES/CBC/PKCS5Padding.
AES/CBC/PKCS5Padding, ARCFOUR/ECB/NoPadding, Camellia/CBC/PKCS5Padding, RC2/CBC/PKCS5Padding, CAST5/CBC/PKCS5Padding.

2. The sender encrypts the symmetric key using a public key.

**Note**

To encrypt the symmetric key, a public key of type RSA (with the cipher – or algorithm – RSA/ECB/PKCS1Padding) is used for each recipient.

3. The sender sends the encrypted message and the encrypted symmetric key to the receiver.
4. The receiver decrypts the symmetric key using a private key (which is related to the public key used by the sender).

**Note**

For this decryption step a private key of type RSA is needed.

5. The receiver decrypts the content of the message using the decrypted symmetric key.

**Note**

Strong encryption is supported for the following algorithms:
- AES/CBC/PKCS5Padding
- Camellia/CBC/PKCS5Padding

For these algorithms also the key lengths 192 and 256 are possible.

The following figure illustrates the process of digitally encrypting and decrypting the content of a message.
7.1.3.2 How XML Signature Works

A digital signature ensures the authenticity of a message in such a way that it guarantees the identity of the signer and that the message was not altered after signing. You have the option to digitally sign and validate a message based on the XML Signature standard (issued by the W3C consortium). Applying this standard means that the digital signature of a document itself is stored as an XML element.

XML Signature can be applied to any XML document.

The following options for XML Signature are supported:

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enveloped Signature</td>
<td>Digital signature/validation is applied to XML element that contains the signature as an element (the Signature element). Using this option, the digital signature is part of the XML document to be signed/validated.</td>
</tr>
<tr>
<td>Enveloping Signature</td>
<td>Digital signature/validation is applied to content within an Object element which is part of the Signature element. That way, the Signature elements acts as an envelope for the signed content. Using this option, the digital signature is part of the XML document to be signed/validated.</td>
</tr>
</tbody>
</table>

**Note**

You configure the usage of XML Signature in the related integration flow.

When applying XML Signature the following signature algorithms are supported: DSA/SHA1, RSA/SHA1, RSA/SHA256, RSA/SHA384, RSA/SHA512

When applying XML Signature the following canonicalization methods are supported: C14N, C14NwithComments, exc-C14N, exc-C14NwithComments

**Background Information**

In a simplified view, digitally signing a message based on XML Signature implies the following main steps:

1. The sender of the message canonicalizes the XML message content that is to be signed. Canonicalization transforms the XML document to a standardized (reference) format. This step is required because an XML document can have more than one valid representations. Calculating a digest out of two different representations of the same document (according to step 2) results in different digests (or hash values). This would make the whole signing/validating process invalid.
2. Out of the canonicalized XML document, a digest is calculated using a digest algorithm.
3. The sender builds up a SignedInfo element that contains the digest.
4. The sender canonicalizes the SignedInfo element.
5. The sender builds a second digest for the `SignedInfo` element which contains the first digest.
6. The sender encrypts the digest using its private key.
7. The sender builds up the `SignatureValue` element which contains the encrypted digest from step 5 (the signature).
8. The message is sent to the receiver.

Digitally verifying (validating) a message based on XML Signature works the following way:

1. The receiver decrypts the encrypted digest (which is part of the `SignatureValue` element of the received message) using the sender’s public key.
2. The receiver calculates the digest out of the `SignedInfo` element of the message.
3. The receiver compares the two digests that result out of steps 1 and 2. That way it is the authenticity of the sender is checked.
4. The receiver canonicalizes the XML message content.
5. The receiver calculates the digest out of the XML message content.
6. The receiver compares the digest that results from the canonicalized message content with that one contained in the `SignedInfo` element of the message. That way, it is made sure that the content of the message has not been altered during message processing.

### 7.1.3.3 How WS-Security Works

Messages can be protected according to the WS-Security standard.

There are the following options:

- Digitally sign a message (and the other way round to verify a signed message)
- Digitally sign a message and to encrypt the message content (and the other way round to verify a message and to decrypt the message content)

### Signing a Message

Signing a message (SOAP body) based on the WS-Security is an additional feature with regard to signing/verifying on payload level based on the following standards: PKCS#7, XML DigitalSignature (see figure below).

---

**Note**

7.1.3.4 How OpenPGP Works

You can use Open Pretty Good Privacy (Open PGP) to digitally sign and encrypt messages.

OpenPGP gives you the following options to protect communication at message level:

- You can encrypt a payload.
- You can sign and encrypt a payload.

OpenPGP does not support signing without encryption or just verifying without decryption. The tenant expects either an encrypted payload or a signed and encrypted payload.

During runtime, the encryptor/signer processor signs and encrypts the body of the inbound message and returns the resulting OpenPGP message in the body of the outbound message.

The required keys are stored in OpenPGP keyrings. The following types of keyrings exist:

Table 18: PGP Keyrings

<table>
<thead>
<tr>
<th>Type of Keyring</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PGP secret keyring</td>
<td>Contains the public/private key pairs of the sender. It can contain multiple key pairs, each identified by a user ID. The private key is protected with a passphrase. For PGP secret keyrings deployed on tenants, the same passphrase has to be used to access all private keys of the PGP secret keyring.</td>
</tr>
<tr>
<td>Type of Keyring</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>PGP public keyring</td>
<td>Contains the public keys (related to the private keys that are stored in the PGP secret keyring of the communication partner).</td>
</tr>
</tbody>
</table>

**OpenPGP Signing/Verifying**

A digital signature ensures the authenticity of a message by guaranteeing the identity of the signer and that the message has not been altered since signing.

A message is digitally signed and verified as follows:

1. The sender calculates a digest (or hash value) from the message content using a digest algorithm. The following hash algorithms are supported for PGP signing:
   - For DSA key: SHA-1, SHA224, SHA256, SHA384, SHA512
   - For RSA key: MD5, SHA-1, RIPE-MD/160, SH256, SHA384, SHA512, SHA224
2. The sender encrypts the digest using a private key (type RSA or DSA). This is the actual signing step. The private key is looked up in the sender’s PGP secret keyring.
3. The encrypted hash value, together with the hash algorithm that has been used, is written to the signature element that is sent to the receiver together with the payload (as PGP signature format). The key ID of the signer of the private key is also written to the PGP signature format.
4. The receiver obtains the PGP signature format.
5. The receiver selects the key ID from the signature and uses the key ID to look up the right public key in the receiver’s PGP public keyring. This is the public key that corresponds to the private key used to sign the payload. In addition, the receiver checks whether the user ID (associated with the key ID) corresponds to an allowed user.
6. The receiver decrypts the hash value (and verifies the payload) using the public key.

The following figure illustrates the concept.
OpenPGP Encrypting/Decrypting

Digital encryption allows you to encode the content of a message in such a way that only authorized parties can read it.

A message is digitally encrypted and decrypted as follows:

1. The sender generates a symmetric key.
2. The sender encrypts the payload with the symmetric key. The following symmetric key algorithms for content encryption (symmetric key algorithms) are supported:
   - TripleDES (168bit key derived from 192), CAST5 (128 bit key, as per [RFC2144]), Blowfish (128 bit key, 16 rounds), AES with 128, 192, and 256-bit key. Twofish with 256-bit key DES is not supported.
3. The sender looks up a public PGP key in the PGP public keyring.
4. The sender encrypts the symmetric key using the public PGP key (from the PGP public keyring). You can use the following key types to encrypt the symmetric key: RSA and Elgamal (DAS is not supported).
5. The sender writes the encrypted symmetric key and the key ID into the Encryption Info element of the message. The key ID is used to identify the public key used for encryption (as the PGP public keyring can contain more than one public key). The Encryption Info element is sent to the receiver, together with the encrypted payload.
6. The receiver obtains the message and, based on the key ID (in the Encryption Info element), looks up the correct private key (associated with the public key used to encrypt the payload) in the PGP secret keyring. A passphrase is required to access the private key.
7. The receiver decrypts the symmetric key with the private key from the PGP secret keyring.
8. The receiver decrypts the payload with the symmetric key.

There is an option to compress data before the encryption step. The following compression algorithms are supported: ZIP [RFC1951], ZLIB [RFC1950], BZip2.

The following figure illustrates the concept.
The runtime supports the following features:

- Signing with several private keys (the resulting OpenPGP message then contains several signatures).
- Encryption with several public keys.
  More precisely, the symmetric encryption key can be encrypted by several public keys (the resulting OpenPGP message then contains several Public Key Encrypted Session Key packets).
- Optional: OpenPGP compression and base 64 output or input.
- OpenPGP allows you to apply two different kinds of keys: primary keys and subkeys. (For simplicity, these are not differentiated in the figures above.)

When you generate OpenPGP keys, a primary key with at least one subkey is created. Only the primary key can be used for certification, that is, to certify the trustworthiness of other keys. In addition, the primary key is also typically used to sign payloads. The subkey is used to encrypt payloads.

OpenPGP Message Format Specification


<table>
<thead>
<tr>
<th>Packet Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Public Key Encrypted Session Key</td>
<td>Session key encrypted with a public key, key ID of the public key, and public-key algorithm</td>
</tr>
<tr>
<td>Signature</td>
<td>Binding between a public key and some data.</td>
</tr>
<tr>
<td></td>
<td>There are several types of signature packets:</td>
</tr>
<tr>
<td></td>
<td>The certification, direct key, and subkey binding signature can be self-signed. The version 4 signature packet may also contain meta-information about the signature such as creation time, issuer, or key expiration time. The version 3 signature is deprecated.</td>
</tr>
<tr>
<td>Symmetric Key Encrypted Session Key</td>
<td>A symmetric key (also called session key) encrypted with a symmetric key; a symmetric algorithm is used. This packet is not supported.</td>
</tr>
<tr>
<td>Packet Type</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>One-Pass Signature</td>
<td>Placed at the beginning of the message before the data. It contains sufficient information to allow the system to start calculating the signature before the actual signature packet (which is after the data) is reached. There can be several such packets. One packet contains the public key algorithm, the hashing algorithm, the key ID of the signing key, and an indicator whether the signatures should be nested or not. A zero value indicates that the next packet is another One-Pass Signature packet that describes another signature to be applied to the same message data.</td>
</tr>
<tr>
<td></td>
<td><strong>Note</strong></td>
</tr>
<tr>
<td></td>
<td>Nested signatures are not supported. However, several signatures over the same data in one PGP message are supported.</td>
</tr>
<tr>
<td>Public Key</td>
<td></td>
</tr>
<tr>
<td>Public Subkey</td>
<td>Contains similar information to a public key package, but it denotes a subkey.</td>
</tr>
<tr>
<td>Secret Key</td>
<td>Contains all the information that is found in a public key packet, but also includes the secret key (encrypted private key).</td>
</tr>
<tr>
<td>Secret Subkey</td>
<td>Contains similar information to a secret key package, but it denotes a subkey.</td>
</tr>
<tr>
<td>Compressed Data</td>
<td>Typically, this packet contains the contents of an encrypted packet, or follows a Signature or One-Pass Signature packet, and it contains a literal data packet.</td>
</tr>
<tr>
<td>Symmetrically Encrypted Data</td>
<td>Data encrypted with a symmetric key (using a symmetric key algorithm). The symmetric cipher used may be specified in a Public-Key or Symmetric-Key Encrypted Session Key packet that precedes the Symmetrically Encrypted Data packet. This packet uses a variant of the cipher feedback mode (CFB) (as defined at <a href="http://tools.ietf.org/html/rfc4880">http://tools.ietf.org/html/rfc4880</a>).</td>
</tr>
<tr>
<td>Literal Data</td>
<td>Contains plain data (binary or text).</td>
</tr>
<tr>
<td>User ID</td>
<td>Indicates the holder of a key. The package contains the user name, e-mail address, and comment of the keyholder.</td>
</tr>
<tr>
<td>Packet Type</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>User Attribute</td>
<td>Variant of the User ID packet, which can contain more information about the user. It is only used together with key material. This packet is not supported.</td>
</tr>
<tr>
<td>Sym. Encrypted and Integrity Protected Data</td>
<td>Variant of the Symmetrically Encrypted Data packet. It contains data that is encrypted with a symmetric key algorithm (using a symmetric key algorithm) and is protected against modification by the SHA-1 hash algorithm (less strong than a signature, but stronger than bare CFB encryption). It does not use Open PGP CFB mode but pure CFB mode.</td>
</tr>
</tbody>
</table>

**Restrictions for the Input Message Structure (for Decryptor/Verifier)**

The input payload must have the following packet sequence:

**Public Key Encrypted Session Key ..., Sym. Encrypted and Integrity Protected Data | Sym. Encrypted Data, (Compressed Data,) (One Pass Signature ....,)Literal Data, (Signature ....,)**

Entries in brackets are optional, ellipses indicate repetition, commas represent sequential composition, and '|' separates alternatives.

For example, the Compressed Data packet is optional.

**Restrictions for the Output Message Structure (for Encryptor/Signer)**

The output PGP message is restricted to the following packet sequence:

**Public Key Encrypted Session Key ..., Sym. Encrypted and Integrity Protected Data | Sym. Encrypted Data, Compressed Data, (One Pass Signature ....,) Literal Data, (Signature ....,)**

Entries in brackets are optional, ellipses indicate repetition, commas represent sequential composition, and '|' separates alternatives.

This does mean the following:

- A symmetric key cannot be encrypted with another symmetric key. The symmetric key that encrypts the payload cannot be encrypted by another symmetric key (which is, for example, generated from a password). OpenPGP allows this (see Symmetric Key Encrypted Session Key packet).
- Compression cannot be switched off. The Compressed Data packet is always mandatory. However, it is possible to choose the UNCOMPRESSED algorithm. In this case, the Compressed Data packet is still there, but contains the Literal Data uncompressed.
- Encryption is always mandatory. It is not possible to only sign data.
- Only one password for all private keys in the keyring can be used. This simplifies password maintenance.
- Nested signatures are not supported: If there are multiple signatures in the PGP message, they all contain the same hash value built over the original payload. OpenPGP does allow nested signatures where the enclosing signature is a signature of the enclosed PGP message including the enclosed signatures.
- DSA keys can only be combined with certain hash algorithms.

7.1.4 Certificate Management

Depending on the applied transport- and message-level security option, different types of security artifacts need to be managed and deployed on the tenant.

- X.509 certificates
  Used for transport-level security TLS and for message-level security using PKCS#7, WS-Security, and XML Digital Signature.
  They are stored in a Java keystore.
- PGP keys
  Used for message-level security using Open PGP.
- Known hosts files
  Required for transport-level security SFTP.
  SFTP keys are also stored in a Java keystore.

Related Information

X.509 Certificates [page 110]
PGP Keys [page 116]
Known Hosts File [page 117]

7.1.4.1 X.509 Certificates

X.509 certificates (that comply with the X.509 standard) are used for transport-level security TLS and for message-level security using PKCS#7, WS-Security, and XML Digital Signature.

Elements of X.509 Certificates

This topic does not explain the standard in detail, but points out the following important elements of an X.509 certificate.

A digital certificate provides a public key that is signed by a certification authority (CA).
<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Issuer</td>
<td>Specifies the CA (that issued and signed the certificate).</td>
</tr>
<tr>
<td>Subject</td>
<td>Specifies the entity associated with the public key of the certificate.</td>
</tr>
<tr>
<td>Distinguished Name (DN)</td>
<td>Comprises the issuer, the subject, and other attributes.  A DN is a unique identifier of the certificate.</td>
</tr>
</tbody>
</table>

When you specify a certificate, you have to define additional attributes such as a company name, a country identification, and so on.

### Related Information

- [Keystores](#)
- [Requirements for Keystore Passwords](#)
- [Certificate Chains](#)

### 7.1.4.1.1 Keystores

X.509 certificates and key pairs are stored in one keystore per tenant.

On each tenant, exactly one keystore needs to be deployed. Therefore, this keystore is also referred to as the tenant keystore.

**Note**

To use X.509 certificates and key pairs, you need to use keystores with the JKS or JCEKS format. We recommend using the JCEKS format because it uses a stronger encryption algorithm.

### Keystore Usage

A tenant keystore is used to secure message exchange both at transport level and at message level.

- **Transport-level security**
  - Keystores are used in HTTPS outbound connections from the SAP Cloud Platform Integration tenant to a remote system (for example, when using a SOAP or HTTP receiver adapter).
  - In such calls, the tenant plays the role of the client and the remote system the role of the server. The trusted certificate entries together with the certificates of the key-pair entries (without the CA certificates of the chains) are used to validate the certificate chain of the server during the TLS handshake. Therefore,
you can control the possible connections to servers with the trusted certificates. You should only add those CA certificates to the keystore that are needed to verify the chains of the servers you want to connect to. It is best practice that the server certificate chain contains the server certificate and the intermediate certificate. That way, the keystore only needs to contain the self-signed root CA certificate as the Certificate entry. The server certificate chain must be verified.

Key-pair entries with their certificate chain are relevant for HTTPS outbound connections with client certificate authentication. During the TLS handshake, one of the key pairs whose certificate chain is trusted by the server is selected for the TLS communication. If the server does not have the appropriate CA certificate in its trust store, the communication fails because the server cannot authenticate the client. If the server trusts several key pairs, one key pair is chosen at random for the connection. If you want to avoid random selection, you can specify an alias of a key-pair entry in the related receiver adapter, so that only this specific key pair can be used in the TLS communication. If the keystore contains only one key pair or the server only trusts one key pair, this measure is not necessary. In some cases it is necessary to adapt the chain of the key pair. For example, if the chain of the key pair contains only the public certificate and the server contains only the root CA certificate, then you need to add the intermediate certificate to the chain of the key pair.

The keystore can also contain additional private keys with the aliases `id_rsa` or `id_dsa` (depending on the key type: RSA or DSA), which are used for the SFTP adapter.

- **Message-level security**

  The keystore also contains the public and private keys used for message-level security (signing and encryption). Public keys are used in the signature verification steps (XML Signature, PKCS#7/CMS Signature Verification, WebService Security) and in the encryption steps (PKCS#7/CMS, WebService Security) of integration flows. Private keys are used in the signature creation steps (XML Signature, PKCS#7/CMS Signature, WebService Security) and decryption steps (PKCS#7/CMS, WebService Security) of integration flows. In these steps, the relevant keystore entries are referenced by their aliases. We recommend using different keys for message- and transport-level security. Keep in mind that the expiration date of the certificates is not checked in the encryption/decryption steps and in the signing steps.

### Keystore Content

Keystore entries are identified by an alias. There are the following entry types:

- **Key Pair** entry

  Consists of a private key and its X.509 certificate chain.

  All private keys of a keystore are encrypted with the same password. This password is also used as the keystore password (for checking the integrity of the keystore). The keystore is never stored in the same database as the encrypted/signed application data. The password is stored in a separate database. The certificate chain typically consists of the public key certificate and the intermediate certification authority (CA) certificate with which the signature of the public key certificate can be verified.

- **Certificate** entry

  In many cases this is an X.509 root certificate.
Keystore Management

A tenant keystore contains both entries owned by the tenant administrator (tenant owner) and entries owned by SAP. SAP-owned entries cannot be changed or deleted by the tenant administrator and entries owned by the tenant administrator cannot be changed or deleted by SAP.

- Keystore management by the tenant administrator
  
  A Keystore Monitor allows tenant administrators to manage their keystore entries and display the SAP-owned entries. The tenant administrator cannot change or delete SAP-owned entries. Likewise, the SAP employee responsible for managing the SAP-owned entries cannot change those owned by the tenant administrator.

  **Note**
  
  This feature is only available for node assembly version 2.29.* and higher.

  Upload, download, and deletion of the keystore entries are restricted by special user roles, which are assigned by default only to the tenant administrator.

- Keystore management by SAP
  
  If SAP manages the keystore entry, it takes special precautions for its maintenance.
  
  When SAP generates a key pair (consisting of a public key and the corresponding private key) during the setup of a new customer tenant, and subsequently issues a certificate signing request, this all happens within a dedicated secure environment only used for this purpose. Only certain operators at SAP have permission to perform these tasks.

  **Note**
  
  There is a dedicated naming convention for keystore aliases to indicate the owner of the keystore entry:

  Alias names of SAP-owned entries start with sap_ or are hcicertificate, hcicertificate1, hcimsgcertificate.

  Exceptions can occur in partner-owned tenants: Here, hcicertificate, hcicertificate1, and hcimsgcertificate can belong to the tenant administrator if they have already been used in a node assembly version prior to 2.29.*.

SAP Cloud Platform Integration does not verify the signatures of the certificates during the upload. Therefore, the user who uploads the certificates is responsible for ensuring that the signatures of the certificates are verified before the upload. Note that root certificates in particular must always be verified manually in any case.

  **Note**
  
  If you are using a node assembly version lower than 2.29.*, be aware of the following implications:

  In this case, one party (SAP or the customer) can always download, upload, and delete the keystore entries owned by the other parties. Therefore, strict governance processes must be employed to ensure that only the party that is in charge of managing the keystore entry can use the upload and delete option. Otherwise, one party could overwrite the keystore entry of the other party, which could have severe implications for the message processing and communication setup.
7.1.4.1.2 Certificate Chains

The trust relationship between a client and a server using TLS authentication is usually based on chain certificates.

When using the X.509 standard, a key pair used for the TLS handshake is usually signed by a certification authority (CA). This means that the server can assume that the public key (included in the certificate) provided by the client originates from a trusted source.

The X.509 standard allows you to build up hierarchical trust models. In such a model (also referred to as a certificate chain), many certification authorities (CAs) are involved on different hierarchy levels. This means that the certificate that identifies the CA as a trusted participant can itself be signed by a CA at a higher level in the hierarchy. This means that a number of (intermediate) CAs can be arranged above the actual client certificate. The highest level CA is called the root CA.
The following figure shows a certificate chain with two intermediate CAs:

[Diagram showing a certificate chain with Root CA, Intermediate 1 CA, Intermediate 2 CA, and Client Certificate.]

We assume that the tenant is connected as a client to an external component (which can be referred to as the server or receiver system).

To establish SSL connectivity, the server is provided with the root CA certificate and nothing else. To make sure that a trust relationship between client and server can be established nevertheless, the client certificate...
(of the tenant) used for the SSL handshake has to contain the whole certificate chain. In other words, the client certificate has to include all intermediate CAs (excluding the root CA). This enables the server to evaluate and calculate the whole chain of trust.

Therefore, during connection setup (onboarding), the tenant key pair (client certificate) has to be assigned the whole certificate chain.

**Tip**
To find out the certificate chain of the server, you can use the TLS Outbound Connection Test (accessible in the Monitoring application). This test also helps you to find out whether you have the correct CA certificate in the keystore to validate the server certificate chain (see option Validate Server Certificate of the Outbound Connection Test).

### 7.1.4.1.3 Requirements for Keystore Passwords

To protect a keystore, you have to specify a password when creating the keystore.

You have to apply the following rules when specifying passwords for keystores:

- The password must have a minimum length of 8 characters.
- The password must contain characters of **at least three** of the following groups:
  - Lower-case Latin characters (a-z)
  - Upper-case Latin characters (A-Z)
  - Base 10 digits (0-9)
  - Non-alphabetic characters (!@#$%...)
- The password must **not** contain any characters from outside the standard ASCI table like, for example, German umlaut characters (<ü>).

**Note**
Example for password compliant with the above rule:

<xB+gku!kjhz>

### 7.1.4.2 PGP Keys

PGP public and secret keys (the latter containing a private key) can be uploaded to the tenant via separate keyrings. The PGP Public Keyring contains Transferable Public Keys as defined in section 11.1 of the Open PGP specification ([https://tools.ietf.org/html/rfc4880](https://tools.ietf.org/html/rfc4880)) and the secret keyring contains Transferable Secret Keys as defined in section 11.2.

PGP keys are used in the PGP Encryptor and Decryptor step. You should only add PGP Public keys to the PGP Public Keyring if you trust this key. Typically you check the fingerprint of the public key. The same security measures must be taken for the secret keys which you use in the secret keyring. The encryption and signing steps do also work with expired certificates.
For the PGP Secret Keyring the same precautions as for the X.509 keystore must be taken because it contains private keys.

### 7.1.4.3 Known Hosts File

Known hosts files are relevant for SFTP communication. The known hosts file contains the host names and the public keys of the trusted SFTP servers. You should only have entries for those servers in the file which are used by the integration flows of the tenant and which you trust.

### 7.2 Security Elements

To set up the secure communication between a tenant and a sender/receiver system, certain security elements have to be created and - in some cases - exchanged between the involved components (the tenant on the one side and the sender/receiver system on the other side of the communication).

For example, to set up SSL communication using certificate-based authentication between a tenant and a receiver system, X.509 certificates are required. Those private keys owned by the tenant are to be part of a Java keystore that is to be deployed on the tenant, whereas the private keys owned by the receiver are to be part of the receiver system keystore. To complete the security setup, each keystore also has to contain the public key of the connected partner. In our example, the Java keystore of the tenant has to contain the receiver public key, and the receiver keystore has to contain the tenant public key.

This section provides a summary for each security option of how the required security elements have to be distributed among the involved components (tenant and sender/receiver systems).

### Related Information

- Security Elements (Transport-Level Security) [page 117]
- Security Elements (Message-Level Security) [page 120]
- How Security Artifacts Are Related to Integration Flow Configuration [page 124]

### 7.2.1 Security Elements (Transport-Level Security)

Each transport-level security option requires a specific set of security elements.

The following tables provide a summary of how the required security elements (in **bold letters**) have to be distributed among the involved components (tenant and sender/receiver systems).
## Table 21: Transport-Level Security

<table>
<thead>
<tr>
<th>Security Option</th>
<th>Direction</th>
<th>Required by tenant administrator</th>
<th>... to do the following</th>
<th>Required by sender/receiver administrator</th>
<th>... to do the following</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTPS – basic authentication</td>
<td>Inbound (sender calls tenant)</td>
<td><strong>User name</strong> of SAP Cloud Platform (to be provided by sender administrator). This is the user under which the customer system is to call the integration platform of SAP Cloud Platform.</td>
<td>Grant the required authorizations to enable this user to call the tenant.</td>
<td><strong>Load balancer root certificate</strong> (to be provided by tenant administrator) Is required for the SSL communication step (can be obtained via the URL of the runtime node provided in the tenant mail by SAP).</td>
<td>Import into the keystore of the sender system.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td><strong>User name and password</strong> (to be provided by tenant administrator)</td>
<td>Enable the sender to support basic authentication.</td>
</tr>
<tr>
<td>Outbound (tenant calls receiver)</td>
<td><strong>Receiver server root certificate</strong> (to be provided by receiver administrator) Is required to enable HTTPS communication with the receiver system (server).</td>
<td>Import into the tenant keystore (and deploy the keystore on the tenant).</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Security Option</td>
<td>Direction</td>
<td>Required by tenant administrator</td>
<td>... to do the following</td>
<td>Required by sender/receiver administrator</td>
<td>... to do the following</td>
</tr>
<tr>
<td>---------------------------</td>
<td>----------------------</td>
<td>----------------------------------</td>
<td>-------------------------</td>
<td>--------------------------------------------</td>
<td>-------------------------</td>
</tr>
<tr>
<td>HTTPS – certificate-based</td>
<td>Inbound (sender calls tenant)</td>
<td>Sender client root certificate (to be provided by sender administrator)</td>
<td>Check whether the CA the customer system used to get its client certificate signed is already part of the load balancer (server) keystore.</td>
<td>Load balancer server root certificate (to be provided by tenant administrator)</td>
<td>Import into client PSE of the sender system.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Sender client certificate (to be provided by sender administrator)</td>
<td></td>
<td>List of trusted root certificates supported by load balancer (to be provided by tenant administrator)</td>
<td>Select a certification authority from the list for the certificate signing request for the client certificate.</td>
</tr>
<tr>
<td>Outbound (tenant calls receiver)</td>
<td>Receiver server root certificate (to be provided by receiver administrator)</td>
<td>Import into tenant keystore (if not already there).</td>
<td>Tenant client root certificate (to be provided by tenant administrator)</td>
<td>Import into the server PSE of the receiver system.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Tenant client certificate (to be provided by tenant administrator)</td>
<td></td>
<td>Tenant client certificate (to be provided by tenant administrator)</td>
<td>Define the client certificate-to-user mapping for the configuration of authorization checks.</td>
</tr>
<tr>
<td>SFTP</td>
<td>Outbound (tenant as SFTP client sends a request to an SFTP server)</td>
<td>SFTP server (receiver) public key (to be provided by SFTP server (receiver administrator))</td>
<td>Add to known_hosts file (to be deployed as Known Hosts artifact on tenant).</td>
<td>Tenant public key (to be provided by tenant administrator)</td>
<td>Add to authorized_keys file on the SFTP server side.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Is required by tenant to check whether SFTP server can be trusted.</td>
<td></td>
<td>Is used to authenticate tenant as a trusted SFTP client on the SFTP server side.</td>
<td></td>
</tr>
</tbody>
</table>
7.2.2 Security Elements (Message-Level Security)

The configuration of secure message exchange requires the exchange of public keys (or other security-related information) between the involved parties. Each message-level security option requires a specific set of keys to be exchanged.

The following tables provide a summary of how the required security elements (in **bold letters**) have to be distributed among the involved components (tenant and sender/receiver systems).

Table 22: Message-Level Security

<table>
<thead>
<tr>
<th>Security Option/Standard</th>
<th>Direction</th>
<th>Protection Method on Tenant</th>
<th>Required by tenant administrator …</th>
<th>… to do the following</th>
<th>Required by sender/receiver administrator …</th>
<th>… to do the following</th>
</tr>
</thead>
<tbody>
<tr>
<td>PKCS#7, WS-Security, XML Digital Signature (uses X.509 certificates) XML Digital Signature: only sign/encrypt</td>
<td>Inbound (sender calls tenant)</td>
<td>Decrypt</td>
<td>Tenant public key certificate (to be provided by tenant administrator) Is used to encrypt the message from the sender (that is to be encrypted by the tenant).</td>
<td>Import into sender key store</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Verify</td>
<td>Encrypt</td>
<td>Receiver public key certificate (to be provided by receiver administrator) Is used by the tenant to encrypt the message (sent to the receiver).</td>
<td>Import into tenant key store.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| PKCS#7, WS-Security, XML Digital Signature (uses X.509 certificates) XML Digital Signature: only sign/encrypt | Inbound (sender calls tenant) | Decrypt                      | Tenant public key certificate (to be provided by tenant administrator) Is used to encrypt the message from the sender (that is to be encrypted by the tenant). | Import into sender key store | |
| Verify                                   | Encrypt            | Receiver public key certificate (to be provided by receiver administrator) Is used by the tenant to encrypt the message (sent to the receiver). | Import into tenant key store. | |

120 PUBLIC

Connecting a Customer System to SAP Cloud Platform Integration Concepts of Secure Communication
<table>
<thead>
<tr>
<th>Security Option/Standard</th>
<th>Direction</th>
<th>Protection Method on Tenant</th>
<th>Required by tenant administrator ...</th>
<th>... to do the following</th>
<th>Required by sender/receiver administrator ...</th>
<th>... to do the following</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Sign</td>
<td></td>
<td></td>
<td>Tenant public key certificate</td>
<td>Import into receiver keystore</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(to be provided by tenant admin-</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>istrator)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Is used by the receiver to ver-</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>ify the message sent from the</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>tenant.</td>
<td></td>
</tr>
<tr>
<td>OpenPGP (uses PGP keys)</td>
<td>Inbound</td>
<td>Decrypt</td>
<td></td>
<td></td>
<td>Tenant public key (to be pro-</td>
<td>Import into sender PGP</td>
</tr>
<tr>
<td></td>
<td>(sender calls tenant)</td>
<td></td>
<td></td>
<td></td>
<td>vided by tenant administrator)</td>
<td>public keyring</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Is used to encrypt the mes-</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>sage from the sender (that is</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>to be encrypted by the tenant).</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>To make sure that the public</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>key originates from the cor-</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>rect source and that it has not</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>been changed on its way, con-</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>sider the note below this ta-</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>ble.</td>
<td></td>
</tr>
<tr>
<td>Security Option/Standard</td>
<td>Direction</td>
<td>Protection Method on Tenant</td>
<td>Required by tenant administrator …</td>
<td>… to do the following</td>
<td>Required by sender/receiver administrator …</td>
<td>… to do the following</td>
</tr>
<tr>
<td>--------------------------</td>
<td>-----------------</td>
<td>------------------------------</td>
<td>------------------------------------</td>
<td>------------------------</td>
<td>-------------------------------------------</td>
<td>------------------------</td>
</tr>
<tr>
<td></td>
<td>Verify</td>
<td></td>
<td><strong>Sender public key</strong> <em>(to be provided by sender administrator)</em></td>
<td>Is used by the tenant to verify the signature of the message sent from the sender system. To make sure that the public key originates from the correct source and that it has not been changed on its way, consider the note below this table.</td>
<td>Import into tenant PGP public keyring.</td>
<td></td>
</tr>
<tr>
<td>Outbound (tenant calls receiver)</td>
<td>Encrypt</td>
<td><strong>Receiver public key</strong> <em>(to be provided by receiver administrator)</em></td>
<td>Is used by the tenant to encrypt the message (sent to the receiver). To make sure that the public key originates from the correct source and that it has not been changed on its way, consider the note below this table.</td>
<td>Import into tenant PGP public keyring.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Security Option/Standard

<table>
<thead>
<tr>
<th>Security Option/Standard</th>
<th>Direction</th>
<th>Protection Method on Tenant</th>
<th>Required by tenant administrator … to do the following</th>
<th>Required by sender/receiver administrator … to do the following</th>
<th>… to do the following</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Sign</td>
<td></td>
<td>Tenant public key (to be provided by tenant administrator)</td>
<td>Import into receiver PGP public keyring</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Is used by the receiver to verify the message sent from the tenant.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>To make sure that the public key originates from the correct source and that it has not been changed on its way, consider the note below this table.</td>
<td></td>
</tr>
</tbody>
</table>

### Note

Relevant for the SAP-managed operating model: When **exchanging public PGP keys**, note the following:

To ensure that the information originates from the correct source and that it has not been changed on its way, the key should be exchanged using a secure channel (for example, encrypted e-mail).

If a secure channel is not available, the person who receives the public key from the key owner has to **verify the fingerprint** of the public key. One option is to phone the owner of the public key and compare the fingerprint.

### 7.2.3 Security Elements Related to the Mail Adapter

The usage of the mail adapter requires certificates both to validate the SSL connection and to encrypt the mail (in case S/MIME has been chosen).

The sender mail adapter enables the tenant to send an (encrypted) email to a receiver system, as illustrated in the following figure.
The tenant keystore needs to contain the following certificates:

Table 23: Certificates to be Imported into Tenant Keystore

<table>
<thead>
<tr>
<th>Certificate</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Receiver server root certificate</td>
<td>For SSL connection:</td>
</tr>
<tr>
<td></td>
<td>This certificate is required to identify the root CA that is at the top of the certificate chain that finally guarantees the trustability of the receiver server certificate.</td>
</tr>
<tr>
<td>Tenant client certificate</td>
<td>For SSL connection:</td>
</tr>
<tr>
<td></td>
<td>This certificate is required to authenticate the tenant when calling the receiver system as client.</td>
</tr>
<tr>
<td>Public key (selected according to public key alias name configured in mail sender adapter)</td>
<td>This certificate is required to encrypt the email.</td>
</tr>
</tbody>
</table>

7.2.4 How Security Artifacts Are Related to Integration Flow Configuration

To specify the security-related aspects of the message flow, certain settings have to be made in the involved integration flows. These security settings are related to the security artifact deployed on the involved tenant.

The following example gives you an idea of how security artifacts and integration flow settings are related to each other: In order to specify in detail how a message is to be digitally encrypted, you need to define an Encryptor step in the relevant integration flow. At runtime, this Encryptor step needs to access the required public key to encrypt the message content. The public key itself has to be available in the keystore that is deployed on the involved tenant.

This section summarizes the following information for each security option:
The required security artifact type (to be deployed on the tenant)

The required step or adapter type (relevant for the related integration flow design)

Table 24: Transport-Level Security Key Types

<table>
<thead>
<tr>
<th>Transport-Level Security</th>
<th>Key Type</th>
<th>Artifact Type (to Deploy on Tenant)</th>
<th>Integration Flow Step/Adapter Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTPS - Basic Authentication</td>
<td>User credentials (user name and password)</td>
<td>User Credentials</td>
<td>SOAP adapter, IDoc adapter, HTTP adapter, SuccessFactors adapter</td>
</tr>
<tr>
<td>HTTPS (SSL) - Certificate-Based Authentication</td>
<td>X.509 certificates</td>
<td>Keystore</td>
<td>SOAP adapter, IDoc adapter, HTTP adapter, SuccessFactors adapter</td>
</tr>
<tr>
<td>SFTP (SSH)</td>
<td>SFTP key and known_hosts</td>
<td>Keystore</td>
<td>SFTP adapter</td>
</tr>
</tbody>
</table>

Table 25: Message-Level Security Key Types

<table>
<thead>
<tr>
<th>Message-Level Security</th>
<th>Key Type</th>
<th>Artifact Type (to Deploy on Tenant)</th>
<th>Integration Flow</th>
</tr>
</thead>
<tbody>
<tr>
<td>PKCS#7</td>
<td>X.509 certificates</td>
<td>Keystore</td>
<td>Signer, Encryptor, Verifier, Decryptor</td>
</tr>
<tr>
<td>XML Digital Signature</td>
<td>This is the same key type as used when setting up HTTPS-based transport-level security.</td>
<td></td>
<td>Signer, Encryptor, Verifier, Decryptor</td>
</tr>
<tr>
<td>WS-Security</td>
<td>When setting up the security level for a tenant, you can use the same keystore for transport-level security keys (if you are setting up HTTPS-based communication) and for message-level security keys. Note, however, that we recommend using different keys for transport-level and message-level security.</td>
<td></td>
<td>SOAP adapter (SOAP 1.x)</td>
</tr>
<tr>
<td>OpenPGP</td>
<td>PGP public keys and PGP secret keys</td>
<td>PGP public keyring, PGP secret keyring</td>
<td>Signer, Encryptor, Verifier, Decryptor</td>
</tr>
</tbody>
</table>

The following figure illustrates the overall setup with regard to the tenant (for a situation where a keystore containing a public-private key pair is deployed on the tenant as a security artifact).
Connecting a Customer System to SAP Cloud Platform Integration

Concepts of Secure Communication
8 Setting up Message-Level Security Use Cases

On top of the secure transport channel (that is based either on HTTPS or SFTP), you can additionally protect the message exchange by digital encrypting and signing the message.

To do that, you can use different security standards.

Related Information

Inbound: Message-Level Security With PKCS#7, XML DigitalSignature [page 127]
Inbound: Message-Level Security with OpenPGP [page 129]
Outbound: Message-Level Security With PKCS#7, XML DigitalSignature [page 131]
Outbound: Message-Level Security with OpenPGP [page 134]

8.1 Inbound: Message-Level Security With PKCS#7, XML DigitalSignature

On top of a secure transport channel (for example, based on HTTPS), you have the option to implement message-level security capabilities. That way, you can protect the message by applying digital signing or encryption. Asymmetric key technology is used in the following way to implement these features:

Table 26: Keys for Message-Level Security

<table>
<thead>
<tr>
<th>Key Type</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Private key</strong></td>
<td>Used by a sender to sign a message</td>
</tr>
<tr>
<td></td>
<td>Used by a receiver to decrypt a message (that has been encrypted by a sender)</td>
</tr>
<tr>
<td><strong>Public key</strong></td>
<td>Used by a receiver to verify a message (signed by a sender)</td>
</tr>
<tr>
<td></td>
<td>Used by a sender to encrypt a message</td>
</tr>
</tbody>
</table>

In the inbound case, the tenant acts as receiver that either decrypts or verifies a message.

To implement message-level security for the standards PKCS#7, WS-Security, and XML Digital Signature, you use X.509 certificates (the same type of certificates as used for HTTPS-based transport-level security). However, note that different keys are usually used for message-level security and SSL transport-level security. XML Digital Signature supports only the use cases of signing/verifying messages.
Configuring the Sender

Configure the sender keystore in the following way:

- Generate a key pair (and get it signed by a CA).
- Import the tenant public key into the sender keystore.

Provide SAP with the public key (is used to verify messages sent to the tenant).

Configuring the Integration Flow Steps for Message-Level Security

Depending on the desired option, configure the security-related integration flow steps.

- Configure the **Verifier** (PKCS7 or XML Signature Verifier) step.
  - Specify the **Public Key Aliases** in order to select the relevant keys from the tenant keystore.
- Configure the **Decryptor** (PKCS7) step.
  - Make sure that you specify the **Public Key Aliases** for all expected senders (only if you have specified **Enveloped or Signed and Enveloped Data** or **Signed and Enveloped Data for Signatures in PKCS7 Message**).
  - These are the public key aliases corresponding to the private keys (of the expected senders) that are used to sign the payload. The public key aliases specified in this step restrict the list of expected senders and, in this way, act as an authorization check.
In general, an alias is a reference to an entry in a keystore. A keystore can contain multiple public keys. You can use a public key alias to refer to and select a specific public key from a keystore.

Related Information

- How PKCS#7 Works [page 99]
- How XML Signature Works [page 102]
- How WS-Security Works [page 103]
- Creating Keys for the Usage of PKCS#7, XML Digital Signature and WS-Security [page 73]

### 8.2 Inbound: Message-Level Security with OpenPGP

On top of a secure transport channel (for example, based on HTTPS), you have the option to implement message-level security capabilities. That way, you can protect the message by applying digital signing or encryption. Asymmetric key technology is used in the following way to implement these features:

<table>
<thead>
<tr>
<th>Key Type</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Private key</td>
<td>Used by a sender to sign a message</td>
</tr>
</tbody>
</table>
In the inbound case, the tenant acts as receiver that either decrypts or verifies a message.

To implement message-level security for OpenPGP, you use PGP keys.

### Configuring the Sender

1. Generate and configure the PGP keys and the storage locations (PGP secret and public keyrings) for the sender system.
2. Import the related public keys from the tenant into the public PGP keyring of the sender and finish the configuration of the sender system.

Provide SAP with the public key (is used to verify messages sent to the tenant).

### Configuring the Integration Flow Steps for Message-Level Security

Configure the security-related integration flow steps.
Configure the **Decryptor** (PGP) and **Verifier** (PGP) step.

When signatures are expected, make sure that you specify the **Signer User ID of Key(s) from Public Keyring** for all expected senders.

Based on the signer user ID of key(s) parts, the public key (for message verification) is looked up in the PGP public keyring. The signer user ID of key(s) key parts specified in this step restrict the list of expected senders and, in this way, act as an authorization check.

---

**Related Information**

- [How OpenPGP Works](#) [page 104]
- [Creating OpenPGP Keys](#) [page 56]

---

### 8.3 Outbound: Message-Level Security With PKCS#7, XML DigitalSignature

On top of a secure transport channel (for example, based on HTTPS), you have the option to implement message-level security capabilities. That way, you can protect the message by applying digital signing or encryption. Asymmetric key technology is used in the following way to implement these features:

**Table 28: Keys for Message-Level Security**

<table>
<thead>
<tr>
<th>Key Type</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Private key</strong></td>
<td>Used by a sender to sign a message</td>
</tr>
</tbody>
</table>
### Key Type

<table>
<thead>
<tr>
<th>Key Type</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Public key</strong></td>
<td>Used by a receiver to decrypt a message (that has been encrypted by a sender)</td>
</tr>
<tr>
<td></td>
<td>Used by a receiver to verify a message (signed by a sender)</td>
</tr>
<tr>
<td></td>
<td>Used by a sender to encrypt a message</td>
</tr>
</tbody>
</table>

In the outbound case, the tenant acts as sender that either encrypts or signs a message.

To implement message-level security for standards PKCS#7, WS-Security, and XML Digital Signature, you use X.509 certificates (the same type of certificates as used for HTTPS-based transport-level security). However, note that different keys are usually used for message-level security and SSL transport-level security. XML Digital Signature supports only use cases for signing and verifying messages.

#### Configuring the Receiver

Configure the receiver keystore in the following way:

- Generate a key pair (and get it signed by a CA).
- Import the tenant public key into the receiver keystore.

Provide SAP with the public key (is used to encrypt messages sent to the receiver).
Configuring the Integration Flow Steps for Message-Level Security

Depending on the desired option, configure the security-related integration flow steps.

- Configure the **Verifier** (PKCS7 or XML Signature Verifier) step. Specify the **Public Key Aliases** in order to select the relevant keys from the tenant keystore.
- Configure the **Decryptor** (PKCS7) step. Make sure that you specify the **Public Key Aliases** for all expected senders (only if you have specified *Enveloped or Signed and Enveloped Data* or *Signed and Enveloped Data* for *Signatures in PKCS7 Message*). These are the public key aliases corresponding to the private keys (of the expected senders) that are used to sign the payload. The public key aliases specified in this step restrict the list of expected senders and, in this way, act as an authorization check.

In general, an alias is a reference to an entry in a keystore. A keystore can contain multiple public keys. You can use a public key alias to refer to and select a specific public key from a keystore.

Related Information

- How PKCS#7 Works [page 99]
- How XML Signature Works [page 102]
- How WS-Security Works [page 103]
- Creating Keys for the Usage of PKCS#7, XML Digital Signature and WS-Security [page 73]
8.4 Outbound: Message-Level Security with OpenPGP

On top of a secure transport channel (for example, based on HTTPS), you have the option to implement message-level security capabilities. That way, you can protect the message by applying digital signing or encryption. Asymmetric key technology is used in the following way to implement these features:

Table 29: Keys for Message-Level Security

<table>
<thead>
<tr>
<th>Key Type</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Private key</td>
<td>Used by a sender to sign a message</td>
</tr>
<tr>
<td></td>
<td>Used by a receiver to decrypt a message (that has been encrypted by a sender)</td>
</tr>
<tr>
<td>Public key</td>
<td>Used by a receiver to verify a message (signed by a sender)</td>
</tr>
<tr>
<td></td>
<td>Used by a sender to encrypt a message</td>
</tr>
</tbody>
</table>

In the outbound case, the tenant acts as sender that either encrypts or signs a message.

To implement message-level security for OpenPGP, you use PGP keys.

![Diagram showing the process of sending a message with OpenPGP](image)

Configuring the Receiver

1. Generate the PGP keys and the storage locations (PGP secret and public keyrings) for the receiver system.
2. Import the related public keys from the tenant into the public PGP keyring of the receiver and finish the configuration of the receiver system.

Provide SAP with the public key (used to encrypt messages sent to the receiver).

**Configuring the Integration Flow Steps for Message-Level Security**

Depending on the desired option, configure the security-related integration flow steps.

- Configure the **Encryptor (PGP)** step. Specify the **User ID of Key(s) from Public Keyring** in order to select the relevant public receiver keys from the PGP public keyring.

**Related Information**

- [How OpenPGP Works](#) [page 104]
- [Creating OpenPGP Keys](#) [page 56]
Specific Use Cases

Setting Up Principal Propagation (Example Scenario)

Use principal propagation to forward the principal (identity of a user) across several connections in a complex system landscape.

In the following example setup, the principal of the inbound user is forwarded to SAP Cloud Connector, and from there to the back-end receiver system.

We assume the following:

- The authentication option OAuth is used (using OAuth SAML Bearer Destination) for inbound communication (from the sender to SAP Cloud Platform).
- An on-premise SAP system based on Application Server ABAP is used as the receiver system. The on-premise receiver system is connected to SAP Cloud Platform through SAP Cloud Connector.
- The receiver system is associated with an identity provider, which mediates a trust relationship between the sender, SAP Cloud Platform, and the receiver.
- To establish an outbound connection (from SAP Cloud Platform to SAP Cloud Connector), an adapter that supports principal propagation is used (for example, the HTTP receiver adapter).
- All systems that communicate with each other have to provide the same user. This can be achieved by using an identity provider, as indicated in the figure above as an example setup.

Caution

Using SAP Cloud Connector is a mandatory when configuring principal propagation.

To configure principal propagation for this setup, perform the following steps.

1. Enable OAuth (with SAML Bearer Destination) for the inbound connection from the sender to SAP Cloud Platform.
   More information: OAuth SAML Bearer Destination [page 26]
Note that currently only the following (sender) adapter types can be used on the inbound side: HTTPS, SOAP (SOAP 1.x), SOAP (SAP RM), and IDoc.

For special use cases, this authentication method can also be used with the AS2 adapter.

2. In the receiver channel of the integration flow, as Authorization option, enable Principal Propagation. More information:

3. Prepare SAP Cloud Connector to support principal propagation with X.509 certificates (for the communication with the receiver system).
   You need a certificate chain with at least one intermediate certification authority. The intermediate certification authority signs a short-lived certificate, which is used for principal propagation. Use the user name (associated with the identity to be propagated) as the subject common name (subject CN) of this certificate.
   SAP Cloud Connector forwards the identity (to be propagated) in a short-living X.509 certificate in HTTP header SSL_CLIENT_CERT.
   More information: Configuring a CA Certificate for Principal Propagation

4. In SAP Cloud Connector, configure the trust relationship with the SAP Cloud Platform application. More information: Configuring the Cloud Connector
   You can find a step-by-step description for an example configuration in the following document under Configure HCC for Principal Propagation: HCP, OData Provisioning Principal Propagation

5. Configure the receiver system. You need to do the following:
   - Configure the receiver system to trust the certificate of the SAP Cloud Connector.
   - Configure the Internet Communication Manager (ICM) to trust the system certificate for principal propagation.
   - Map the short-living certificate (from SAP Cloud Connector) to the user (whose identity is being propagated).
   More information: Configuring Principal Propagation to an ABAP System for HTTPS

9.2 Technical Landscape for On Premise-On Demand Integration

As one example for certificate-based connectivity, customer intends to connect a customer-based SAP on-premise system (based on SAP Application Server ABAP with Cloud Integration).

The following figure illustrates the required keystores and security artifacts for the mentioned landscape.
Note
We use the following abbreviations in this documentation:

- **AS** for SAP Application Server
- **WD** for SAP Web Dispatcher
- **Cloud Integration** for SAP Cloud Platform Integration

In the proposed system landscape, SAP Web Dispatcher is used in the on premise customer landscape to receive incoming calls from Cloud Integration. SAP Web Dispatcher (as reverse proxy) is the entry point for HTTPS requests into the customer system landscape.

**Communication Cloud Integration to SAP Application Server**

In the proposed landscape, two SSL connections have to be implemented on the way in between Cloud Integration and AS, because SAP Web Dispatcher - interconnected in between - terminates all SSL calls from Cloud Integration. Therefore, the following trust relationships have to be implemented:

- Trust relationship between SAP Web Dispatcher and Cloud Integration.
  As this connection spans the Internet, it is strongly recommended to use certificates that are signed by a certification authority (CA) that both parties (SAP Web Dispatcher and Cloud Integration) trust.
- Trust relationship between SAP Web Dispatcher and AS.
  As this connection resides within the customer landscape, it might be an option to use self-signed certificates for this connection.

Note
For reasons of simplicity, within this guide we assume that self-signed certificates are used for this connection.

The following table summarizes the required certificates and the related keystores.
### Table 30: Keystores

<table>
<thead>
<tr>
<th>Keystore</th>
<th>Certificate/Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cloud Integration client keystore</td>
<td>Cloud Integration client certificate (private and public key)</td>
<td>Required to authenticate Cloud Integration as sender of messages. This security artifact has to be generated at SAP side and contains the public and private key of Cloud Integration. The certificate has to be signed by a certification authority (CA) that both SAP (Cloud Integration) and the customer (WD) trust.</td>
</tr>
<tr>
<td>WD server root certificate (of the CA</td>
<td></td>
<td>Required to authenticate WD as receiver of messages. This certificate identifies the CA that has signed the server certificate.</td>
</tr>
<tr>
<td>that has signed the server certificate</td>
<td></td>
<td></td>
</tr>
<tr>
<td>WD server keystore (SSL server PSE)</td>
<td>Cloud Integration client root certificate</td>
<td>Required to identify Cloud Integration as trusted communication partner. This certificate identifies the CA that has signed the Cloud Integration client certificate.</td>
</tr>
<tr>
<td>WD server certificate</td>
<td></td>
<td>Required to authenticate WD as trusted communication partner to receive calls. This certificate is signed by the CA to which both WD and Cloud Integration have established a trust relationship.</td>
</tr>
<tr>
<td>WD client keystore (SSL client PSE)</td>
<td>WD client certificate (private and public key)</td>
<td>Required to authenticate WD as sender of messages. This security artifact has to be generated at customer side and contains the public and private key of the WD. As the related communication path resides within the customer landscape, it might be sufficient to use a self-signed certificate.</td>
</tr>
<tr>
<td>i Note</td>
<td></td>
<td>Customers can extend the use case in a way that also this certificate is signed by a CA. This is not covered in this guide.</td>
</tr>
</tbody>
</table>
Communication SAP Application Server to Cloud Integration

In the proposed landscape, the SSL connection is not terminated on the way in between AS and Cloud Integration (transparent proxy). Therefore, a trust relationship has to be set up between AS and Cloud Integration.

As this connection spans the Internet, it is strongly recommended to use certificates that are signed by a certification authority (CA) that both parties (AS and Cloud Integration) trust.

The following table summarizes the required certificates and the related keystores.

Table 31: Keystores

<table>
<thead>
<tr>
<th>Keystore</th>
<th>Certificate/Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AS server keystore (SSL server PSE)</td>
<td>WD client certificate (public key)</td>
<td>Required to authenticate WD as sender of messages. This public key has to be imported into the AS server keystore.</td>
</tr>
<tr>
<td>AS client keystore</td>
<td>AS client certificate (private and public key)</td>
<td>Required to authenticate AS as sender of messages. This security artifact has to be generated at customer side and contains the public and private key of AS. The certificate has to be signed by a certification authority (CA) that both SAP (Cloud Integration) and the customer (AS) trust.</td>
</tr>
<tr>
<td>Cloud Integration server root certificate</td>
<td></td>
<td>Required to authenticate Cloud Integration as trusted receiver of messages. This certificate identifies the CA that has signed the Cloud Integration server certificate.</td>
</tr>
<tr>
<td>Cloud Integration server keystore</td>
<td>AS client root certificate</td>
<td>Required to authenticate AS as sender of messages. This certificate identifies the CA that has signed the AS client certificate. This artifact has to be provided by the customer for SAP during the connection setup process, and the expert at SAP side has to import it into the Cloud Integration server keystore.</td>
</tr>
<tr>
<td>Keystore</td>
<td>Certificate/Key</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>-----------------</td>
<td>-------------</td>
</tr>
<tr>
<td></td>
<td>Cloud Integration server certificate</td>
<td>Required to authenticate Cloud Integration as trusted communication partner to receive calls. This certificate is signed by the CA to which both AS and Cloud Integration have established a trust relationship.</td>
</tr>
</tbody>
</table>

You can find more information on this landscape in the *Technical Connectivity Guide for SAP Cloud for Travel* at [https://service.sap.com/ondemand](https://service.sap.com/ondemand) under *SAP Cloud for Travel and Expense*. 
Important Disclaimers and Legal Information

Coding Samples

Any software coding and/or code lines / strings (“Code”) included in this documentation are only examples and are not intended to be used in a productive system environment. The Code is only intended to better explain and visualize the syntax and phrasing rules of certain coding. SAP does not warrant the correctness and completeness of the Code given herein, and SAP shall not be liable for errors or damages caused by the usage of the Code, unless damages were caused by SAP intentionally or by SAP’s gross negligence.

Accessibility

The information contained in the SAP documentation represents SAP’s current view of accessibility criteria as of the date of publication; it is in no way intended to be a binding guideline on how to ensure accessibility of software products. SAP in particular disclaims any liability in relation to this document. This disclaimer, however, does not apply in cases of willful misconduct or gross negligence of SAP. Furthermore, this document does not result in any direct or indirect contractual obligations of SAP.

Gender-Neutral Language

As far as possible, SAP documentation is gender neutral. Depending on the context, the reader is addressed directly with “you”, or a gender-neutral noun (such as “sales person” or “working days”) is used. If when referring to members of both sexes, however, the third person singular cannot be avoided or a gender-neutral noun does not exist, SAP reserves the right to use the masculine form of the noun and pronoun. This is to ensure that the documentation remains comprehensible.

Internet Hyperlinks

The SAP documentation may contain hyperlinks to the Internet. These hyperlinks are intended to serve as a hint about where to find related information. SAP does not warrant the availability and correctness of this related information or the ability of this information to serve a particular purpose. SAP shall not be liable for any damages caused by the use of related information unless damages have been caused by SAP’s gross negligence or willful misconduct. All links are categorized for transparency (see: http://help.sap.com/disclaimer).