4Res Vendor Guide

Overview

This document corresponds to the 2.0.16 or higher release of 4Res®. 4Res is a cardholder data firewall that sits between the external reservation system and a hotel's property management system (PMS), and it allows cardholder data (CHD) to be tokenized before it enters the hotel's environment. For security and account linkage purposes, Lighthouse Transaction Manager assigns an Access Token that 4Res uses in all requests. The Access Token is 50 characters.

4Res currently supports interfaces using the OpenTravel Alliance (OTA) or Hotel Technology Next Generation (HTNG) interface XML standards. Support for other interface standards will be added based on market demands. 4Res was designed to require as few changes as possible to existing interfaces, optimally, utilizing a single URL change by one or both clients.

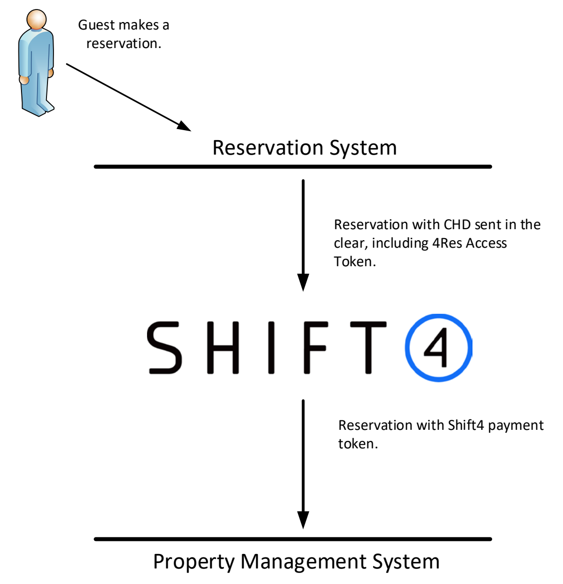
Depending on your business model, 4Res can be used in either a push method (where an external reservation system pushes reservations to a PMS) or a pull method (where the PMS polls an external reservation system for reservations), or both. It is recommended that you generate a separate Access Token for each external reservation system that will be used. This allows you to revoke an Access Token for an external reservation system that you are no longer using without having to generate a new Access Token for the rest of your reservation systems.

Note: 4Res supports sending multiple reservations in a single request. Each card number will be tokenized and returned with its respective reservation.

See the examples below.

Push vs. Pull

Push to Property Management System


Guest makes a reservation.
Reservation System
Reservation with CHD sent in the clear, including 4Res Access Token.
SHIFT4
Reservation with Shift4 payment token.
Property Management System

In this model, the reservation system pushes reservations through 4Res to the PMS. The flow is as follows:

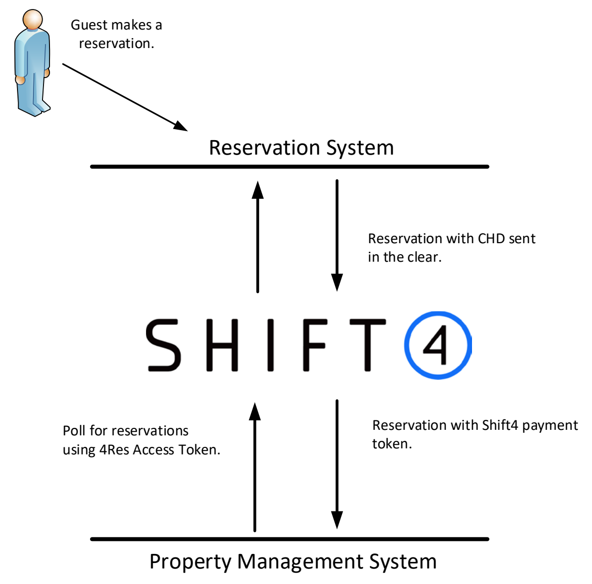
Step 1: A guest makes a reservation using a reservation company.

Step 2: The reservation company sends the reservation with the cardholder data, including the 4Res Access Token, to the 4Res URL.

Step 3: 4Res removes and stores the actual cardholder data. The cardholder data is replaced with a payment token, which is sent with the rest of the reservation data to the property management system.

Step 4: The property processes the reservation using the payment token. No cardholder data ever enters the property management system.

Pull from Property Management System


Guest makes a reservation.
Reservation System
Reservation with CHD sent in the clear.
SHIFT4
Poll for reservations using 4Res Access Token.
Reservation with Shift4 payment token.
Property Management System

In this model, the PMS polls the reservation system through 4Res. The flow is as follows:

Step 1: A guest makes a reservation using a reservation company.

Step 2: The property management system sends a request for reservations, along with the 4Res Access Token, to the 4Res URL. 4Res passes the request to the reservation company.

Step 3: The reservation company sends the reservation with the cardholder data to the 4Res URL.

Step 4: 4Res removes and stores the actual cardholder data. The cardholder data is replaced with a payment token, which is sent with the rest of the reservation data to the property management system.

Step 5: The property processes the reservation using the payment token. No cardholder data ever enters the property management system.

Shift4 Firewall Requirements

For security reasons, 4Res is restricted to a list of trusted inbound and outbound IP addresses, known as a whitelist. 4Res only accepts requests from Shift4 whitelisted sources, and only sends requests to Shift4 whitelisted destinations.

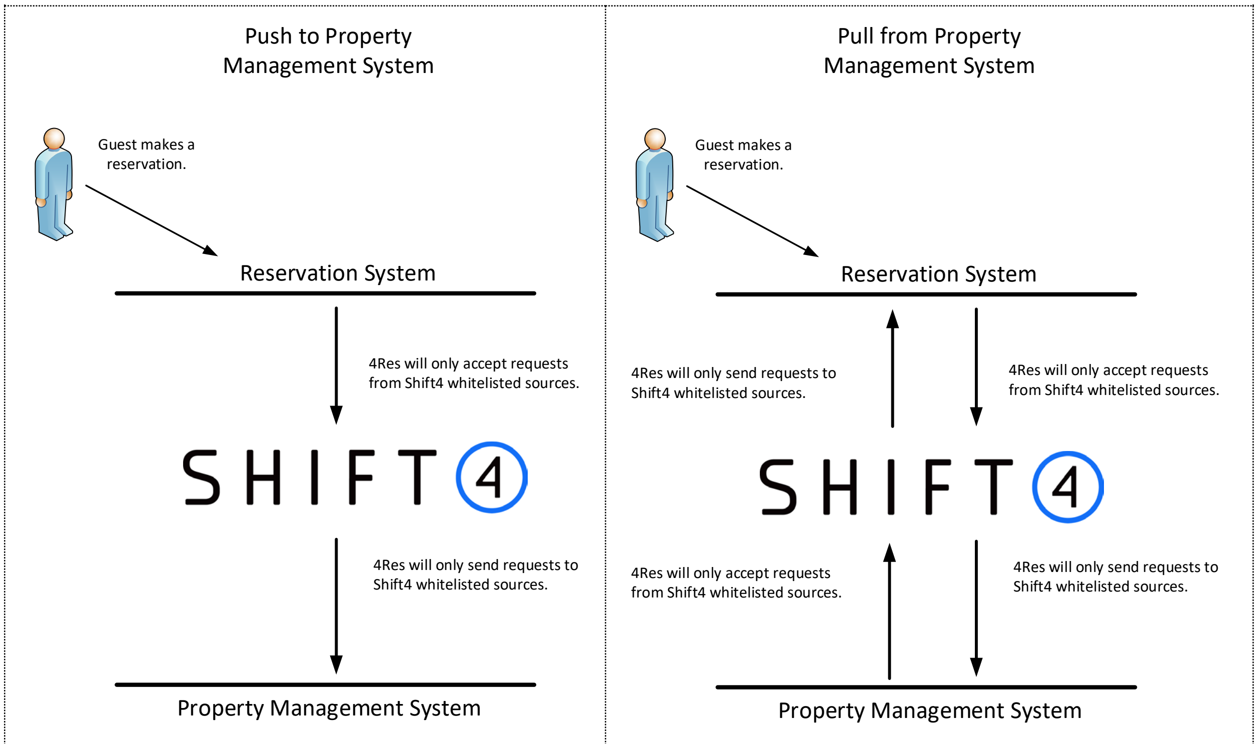
The information below must be submitted to Shift4 for each merchant and external reservation source that will be sending and receiving reservations. Shift4 will review the source information. If any sources or destinations are not currently whitelisted, Shift4 will attempt to whitelist them.

Both reservation services IP addresses and merchant IP addresses must be submitted to Shift4 well in advance of installation to prevent delays. Each IP address must indicate whether it is inbound to 4Res or outbound from 4Res.

Send Shift4 the following information for every source or destination:

- Reservation Company Name
- Reservation Company Contact Name
- Reservation Company Contact Phone Number
- Reservation Company Website URL
- Reservation Company IP Addresses
- Merchant Company Name
- Merchant Contact Name
- Merchant Contact Phone Number
- Merchant Website URL
- Merchant Company IP Addresses

Whitelist Flow Diagram


Push to Property Management System
Guest makes a reservation.
Reservation System
4Res will only accept requests from Shift4 whitelisted sources.
SHIFT4
4Res will only send requests to Shift4 whitelisted sources.
Property Management System
Pull from Property Management System
Guest makes a reservation.
Reservation System
4Res will only send requests to Shift4 whitelisted sources.
4Res will only accept requests from Shift4 whitelisted sources.
SHIFT4
4Res will only accept requests from Shift4 whitelisted sources.
4Res will only send requests to Shift4 whitelisted sources.
Property Management System

In this model, the reservation system pushes reservations through 4Res to the PMS.

In this model, the PMS polls the reservation system through 4Res.

Interface Vendor Requirements

The interface vendor will need to supply a means to allow the merchant Account Administrator to plug the Access Token into the application. The application will need to store the Access Token and use it for future processing. The PMS or central reservation system will need to make the following changes:

- Change the URL you are currently using to send the reservations to the appropriate 4Res URL.
    - Testing URL: https://4res.shift4test.com/index.cfm
    - Production URL: https://secure.4res.net/index.cfm
- Parameters
    - fa
        - OTA Push Method: ota.toPMS
        - OTA Pull Method: ota.toRES
        - HTNG Push Method: htng.toPMS
        - HTNG Pull Method: htng.toRES
    - location
        - Push Method: Current PMS URL
        - Pull Method: Current reservation URL

Requirement: If you will be using Web Services Description Language (WSDL), you must include `options=getAssumesWSDL` in your URL. This requirement applies to both the push and pull methods described in this document.

-
    - accessToken
        - Will be dynamic for each site.

In addition, vendors will need to supply written documentation instructing interface users how to plug in the URL and Access Token. This documentation should include whether the interface is push or pull.

Requirement: For security, 4Res is restricted to a whitelist of inbound and outbound IP addresses. Both reservation services IP addresses and merchant IP addresses must be submitted to Shift4 well in advance of installation to prevent delays. Each IP address must indicate whether it is inbound or outbound.

Push Method Example

The following is an example of a 4Res push request. The order is not important, as long as the content is formatted correctly:

`https://4res.shift4test.com/index.cfm?fa=ota.toPMS&location=<`*`Current PMS URL`*`>&accessToken=0925436E-E8F4-E1B6-57A4519619F13B5B`

Vendors integrating with 4Res will need to make the Access Token and Location URL parameters configurable per property.

Pull Method Example

The following is an example of a 4Res pull request. The order is not important, as long as the content is formatted correctly:

`https://4res.shift4test.com/index.cfm?fa=ota.toRES&location=< `*`Current reservation URL`*` >&accessToken=0925436E-E8F4-E1B6-57A4519619F13B5B`

Vendors integrating with 4Res will need to make the Access Token and Location URL parameters configurable per property.

Note: The first time you use the generated Access Token, it is locked to the location defined above. If the location parameter is changed in your interface, you will have to generate a new Access Token for that location.

Using Basic Access Authentication

Basic authentication is a simple technique for enforcing access controls to web resources without requiring cookies, session identifiers, or login pages. Instead, it uses standard fields in the HTTP header, obviating the need for handshakes.

If you are working with an external reservation system that uses basic access authentication, there are two methods for using this protocol with 4Res and they are detailed below.

Method One: Server-Side URL Encoding

The first method requires URL encoding your username and password within the location.

For example, if your username is Aladdin and your password is OpenSesame, the URL would be: https://aladdin:opensesame@somedomain.com/...) Using this method, you are basically creating a username:password@ prefixed URL domain.

Method Two: Client-Side 4res-passthrough-Authorization:Basic xxx

This client-side method requires the following steps:

1. Combine your username and password into a string separated by a colon. For example, username:password.
2. Encode the resulting string using the RFC2045-MIME variant of Base64, except not limited to 76 char/line.
3. Add the authorization method and a space (i.e., "Basic " before the encoded string).
4. Add “4res-passthrough-“ before “Basic ” in the encoded string.

   For example, if the user agent uses Aladdin as the username and OpenSesame as the password, the field is formed as follows: 4res-passthrough-Authorization:Basic QWxhZGRpbjpPcGVuU2VzYW1l

Method two is used to prevent premature usage of credentials. The header name “4res-passthrough-“ will be stripped as the request is forwarded to the provider. 4res-passthrough- is not case sensitive.

Which method you use will depend on your external reservation system, and how they handle the protocol.

API Standards

Much of this design document references OTA and HTNG API specifications. While the initial 4Res design relies on and references specifics in these APIs, it is not a design goal for 4Res to be an OTA/HTNG-only solution. The ultimate goal is to support as many APIs as feasibly and economically possible. OTA and HTNG are currently the most used interface standards for the hospitality industry.

OpenTravel Alliance

OTA (http://www.opentravel.org/) provides a community where companies in the electronic distribution supply chain work together to create an accepted structure for electronic messages, enabling suppliers and distributors to speak the same interoperability language, trading partner to trading partner.

Shift4 worked directly with OTA to include tokenization support in their API. OTA version 2012A includes these Shift4-inspired changes. Whenever this document refers to the OTA API, it is referring to the OTA 2012A version.

Hotel Technology Next Generation

HTNG (http://www.htng.org/) is a global trade association dedicated to enhancing the deployment of technology in hotels. The HTNG API is a subset of OTA.

OTA/HTNG Compatibility

4Res inserts the tokenized data based on the OTA 2014B specification. 4Res is not locked to any specific OTA version. OTA versions prior to 2012A will be formatted based on a hybrid format. This hybrid format preserves the existing data as much as possible, removes the cardholder data, and inserts token data based on the 2014B specification.

XML Best Practices

Due to the inline nature of 4Res and the hybrid data format, the following should be noted:

- 4Res requires well-formed XML, not necessarily valid XML relative to OTA/HTNG. Unknown elements and attributes will be ignored.
- Interface partners should also require well-formed XML, as any hybrid message is not considered valid XML.
- Neither XML nor 4Res is element or attribute order sensitive.
- While 4Res is not case sensitive, XML is generally case sensitive. 4Res will preserve the case whenever possible and insert any new elements and attributes based on the corresponding specification.

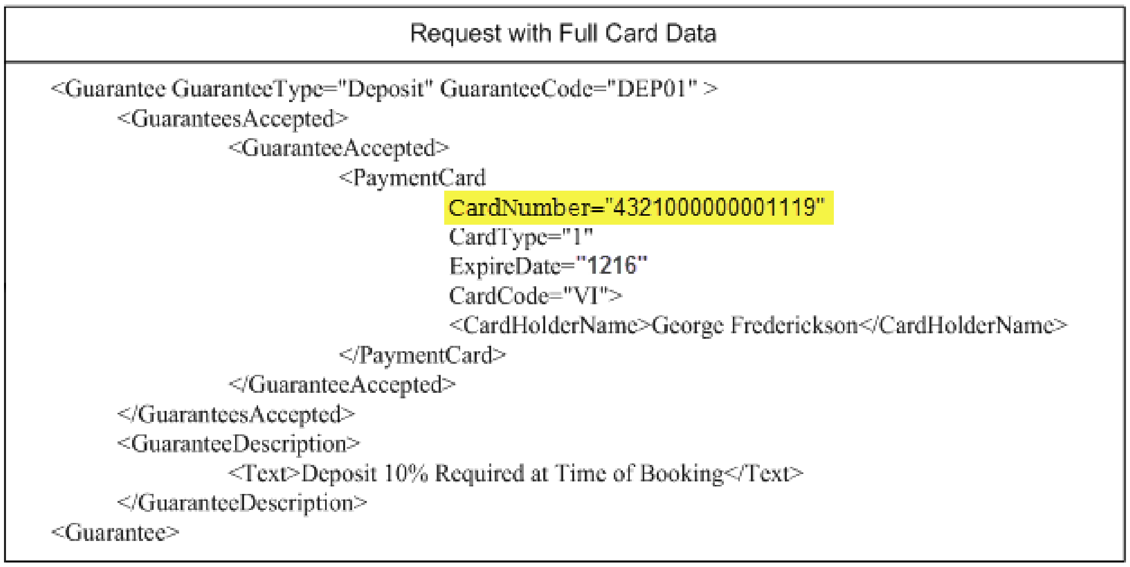
REST vs. SOAP

The 4Res proxy service supports both Representational State Transfer (REST) and Simple Object Access Protocol (SOAP). Currently, 4Res generated errors always respond using a SOAP fault message format that is described later in this document.

Synchronous vs. Asynchronous

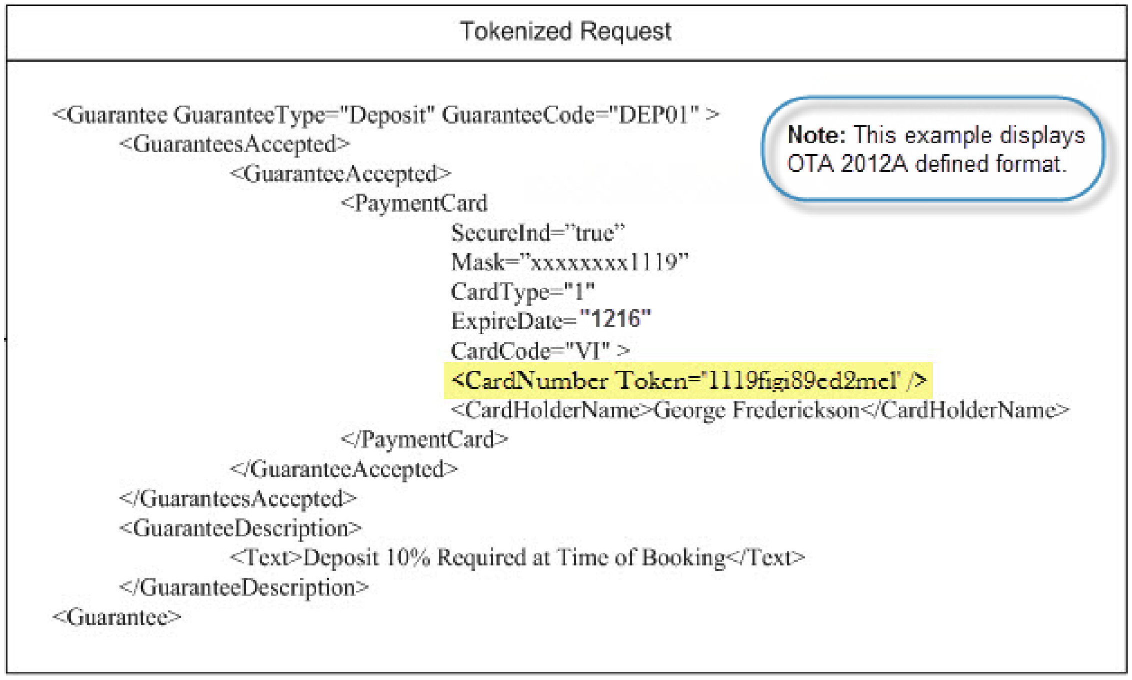
The 4Res proxy service supports both synchronous and asynchronous requests. While 4Res was designed for minimal impact to current data flows, it is important to understand that asynchronous support requires changes to the data flow on both the client-side and the server-side of an interface. Synchronous interfaces, on the other hand, only require modifications on the client-side. Both methods require changes to the PMS to handle tokens instead of card data within the XML requests.

Reservation Record XML Examples



Request with Full Card Data

```
<Guarantee GuaranteeType="Deposit" GuaranteeCode="DEP01" >
        <GuaranteesAccepted>
                <GuaranteeAccepted>
                        <PaymentCard
                                CardNumber="4321000000001119"
                                CardType="1"
                                ExpireDate="1216"
                                CardCode="VI">
                                <CardHolderName>George Frederickson</CardHolderName>
                        </PaymentCard>
                </GuaranteeAccepted>
        </GuaranteesAccepted>
        <GuaranteeDescription>
                <Text>Deposit 10% Required at Time of Booking</Text>
        </GuaranteeDescription>
<Guarantee>
```



Tokenized Request

```
<Guarantee GuaranteeType="Deposit" GuaranteeCode="DEP01" >
        <GuaranteesAccepted>
                <GuaranteeAccepted>
                        <PaymentCard
                                SecureInd="true"
                                Mask="xxxxxxxx1119"
                                CardType="1"
                                ExpireDate="1216"
                                CardCode="VI" >
                                <CardNumber Token="1119figi89ed2mel' />
                                <CardHolderName>George Frederickson</CardHolderName>
                        </PaymentCard>
                </GuaranteeAccepted>
        </GuaranteesAccepted>
        <GuaranteeDescription>
                <Text>Deposit 10% Required at Time of Booking</Text>
        </GuaranteeDescription>
<Guarantee>
```

Note: This example displays OTA 2012A defined format.

Inserting 4Res into the Mix

4Res is designed as a web service proxy. The initial version assumes OTA/HTNG XML formatted data with or without a SOAP envelope.

Error Responses

Any `soap:Fault` generated by the target host will pass through unaltered, unless there is card information in the response, in which case it will be tokenized. The error responses displayed below are Shift4 4Res generated `soap:Fault` responses.

If a valid `soap:Envelope` and a valid `soap:Body` is detected, the response will contain the original request message with the `soap:Fault` block prepended to the `soap:Body` section. See the example below:

```
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/" xmlns:ns="http://www.opentravel.org/OTA/2003/05">
     <env:Header/>
     <env:Body>
          <env:Fault>
               <faultcode>Client.AppError</faultcode>
               <faultstring>4Res Proxy Error</faultstring>
               <detail>
                    <message>Invalid access token specified.</message>
                    <errorcode>45000</errorcode>
                    <errortype>AccessAuthentication</errortype>
                    <docurl>http://www.shift4.com/4Res/apiErrors.cfm?code=45000.AccessAuthentication</docurl>
               </detail>
          </env:Fault>
          <ns:OTA_HotelAvailRQ EchoToken="?" MaxResponses="?" PrimaryLangID="?" TimeStamp="?" Version="?">
               <ns:POS>
                    <ns:Source ISOCountry="?" ISOCurrency="?">
                         <ns:RequestorID ID="?" ID_Context="?" Type="?"/>
                    </ns:Source>
               </ns:POS>
               <ns:AvailRequestSegments>
                    <ns:AvailRequestSegment>
                         <ns:HotelSearchCriteria>
                              <ns:Criterion>
                                        <ns:HotelRef BrandCode="?" ChainCode="?" HotelCityCode="?" HotelCode="?"/>
                                        <ns:StayDateRange End="?" Start="?"/>
                                        <ns:RoomStayCandidates>
                                                  <ns:RoomStayCandidate BedTypeCode="?" NonSmoking="?">
                                                            <ns:GuestCounts>
                                                                      <!--1 to 10 repetitions:-->
                                                                      <ns:GuestCount Age="?" AgeQualifyingCode="?" Count="?"/>
                                                            </ns:GuestCounts>
```

```
                                                </ns:RoomStayCandidate>
                                        </ns:RoomStayCandidates>
                                </ns:Criterion>
                        </ns:HotelSearchCriteria>
                    </ns:AvailRequestSegment>
                </ns:AvailRequestSegments>
            </ns:OTA_HotelAvailRQ>
        </env:Body>
</env:Envelope>
```

There are three things to note in the response example:

1. The Fault tag is the first tag in the Body.
2. The Fault tag namespace matches the namespace provided in the request for Envelope and Body; in this case "env."
3. The message is a reflection of the request sent, not a corresponding response message. In the example above, an OTA_HotelAvailRQ was sent and the OTA_HotelAvailRQ was reflected back as opposed to an OTA_HotelAvailRS which would normally be returned (RQ vs. RS).

In the following example a valid `soap:Envelope` and a valid `soap:Body` is not detected:

```
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
        <soap:Header />
        <soap:Fault>
                <faultcode>soap:Client.AppError</faultcode>
                <faultstring>4Res Proxy Error</faultstring>
                <detail>
                        <message>[string] </message>
                        <errorcode>[integer]</errorcode>
                        <errortype>[string]</errortype>
                        <docurl>[string]</docurl>
                </detail>
        </soap:Fault>
</soap:Envelope>
```

Web Server Certificates

4Res requires the use of HTTPS both inbound and outbound. There are no exceptions to this requirement. Our servers natively handle web server certificates issued by most of the well-known certificate authorities. While we highly recommend using a well-known provider, we can install lesser-known intermediate certificates as well as import self-signed certificates, but these may require extra setup time.