UNITED NATIONS ECONOMIC COMMISSION FOR EUROPE

eTIR web services

Introduction document

UNECE

Table of Contents

1. Document revision note

This document has been published on **08/10/2020**, and is valid for the **eTIR international system version 1.0** based on the **eTIR specifications version 4.3a**.

Please ensure you get the latest version of this document from the eTIR documentation portal or contact the eTIR service desk (Support and contact).

2. Purpose

This document describes the eTIR international system web services, in particular: the intended target audience, the system architecture, the various messages and their sequencing, the security aspects, the accesses as well as the support contacts. It does not cover the detailed implementation required for each message, which can be found in the documents dedicated to each of the eTIR messages. Rather, it describes which components and processes should be put in place in the national customs systems to effectively interconnect with the eTIR international system.

3. Target audience

This guide is intended for the customs ICT team in charge of TIR processes and in charge of interconnecting with their national customs systems with the eTIR international system.

4. Prerequisites

This document is to be read after having an understanding of the eTIR concepts, and after having read and followed the Project guidelines for customs to connect to eTIR. It is most important to understand the implementation stages described in the above guidelines, and to understand where you currently stand in the implementation process.

In order to ensure an implementation that delivers the best value and services to the customs authorities, we highly recommend the ICT team to be accompanied by a TIR subject matter expert, as mentioned in the Project guidelines for customs to connect to eTIR.

5. eTIR documentation walkthrough

The eTIR international system relies on the TIR Convention, and as such the eTIR documentation relies on its articles and annexes, but also on various other key documents available online and introduced below. Some of these documents must be read and well understood while others are reference documents to be consulted when necessary. This section will help you understand what they contain and in which order we recommend reading them.

1. Project guidelines for Customs to connect to eTIR: this document is the starting point for the customs authorities of any contracting party and should be read before any other one.

If you are not familiar with the transport and border crossing facilitation problematics, we highly encourage you to read:

2. The TIR Convention handbook: this document contains all the legal elements that rule the TIR Convention (including Comments and Explanatory Notes). It also describes the processes to be followed by customs authorities, national associations and TIR Carnet holders.

3. Annex 11 to the TIR Convention: Annex 11 to the TIR Convention describes the eTIR procedure, how processes shall be adapted to be computerized and set the legal provisions for the implementation and usage of the eTIR international system.

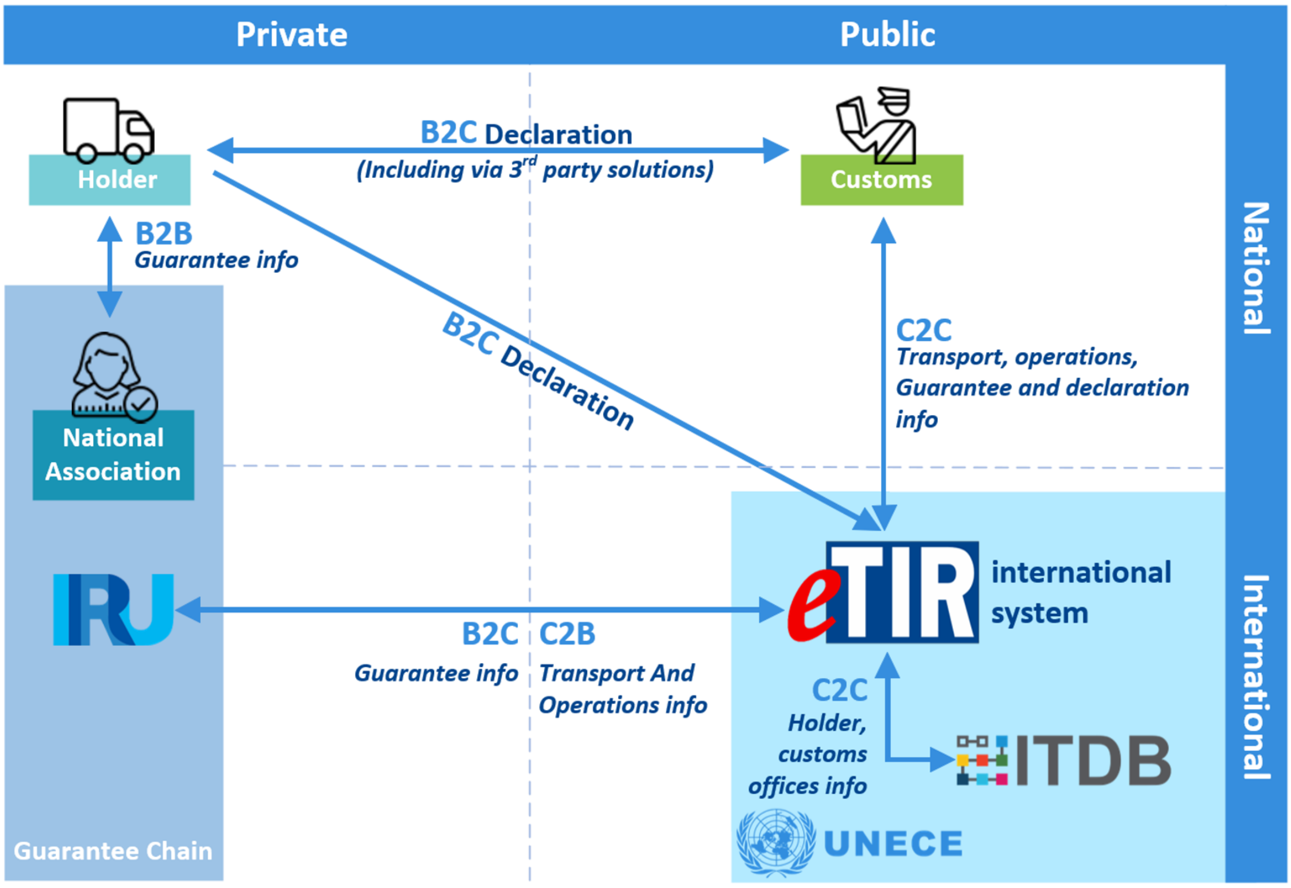
Once the TIR legal context and key processes are well understood, we strongly advise to read:

4. The introduction to the eTIR conceptual, functional and technical documentation: this document introduces the conceptual, functional and technical documentation for the TIR Procedure Computerization Project in accordance with the United Nations Centre for Trade Facilitation and Electronic Business (UN/CEFACT) Modelling Methodology. It is the first document to be read in order to approach how eTIR was envisioned as an extension of the TIR Convention.
5. The eTIR concepts: this document describes the approach and core concepts used to support the business logic, and to implement the eTIR international system. This document is also a reference document that describes all the use cases and all the business rules used for the technical implementation.
6. The eTIR functional specifications: this document is the most important to read to get a deep understanding of the mechanisms used to implement the eTIR international system. The current version (4.2) is being expanded in a working document (4.3a) and refined as the work progresses and as feedback is received from modelling work carried out by the Informal Ad hoc Expert Group on Conceptual and Technical Aspects of Computerization of the TIR Procedure (GE.1) and all the eTIR focal points. These specifications (once called Reference Model) also refer to the following additional documents and lists:
    a. eTIR XML schemas
    b. eTIR code lists
    c. eTIR error code list
7. Finally, if you wish to contact other contracting parties during this implementation, please refer to the The eTIR Focal Points page to find their contact information.

6. eTIR web service overview

6.1. High level architecture

The eTIR international system is based on the following functional overview diagram:


Private
Public
National
International
Holder
Customs
B2C Declaration
(Including via 3rd party solutions)
B2B
Guarantee info
B2C Declaration
C2C
Transport, operations, Guarantee and declaration info
National Association
IRU
B2C
Guarantee info
C2B
Transport And Operations info
eTIR international system
C2C
Holder, customs offices info
ITDB
UNECE
Guarantee Chain

Kindly note the following acronyms meanings:
B2B: Business to Business relationship (where business refers to the private sector).
C2B: Customs to Business relationship (where business refers to the private sector).
B2C: Business to Customs relationship (where business refers to the private sector).
C2C: Customs to Customs relationship.

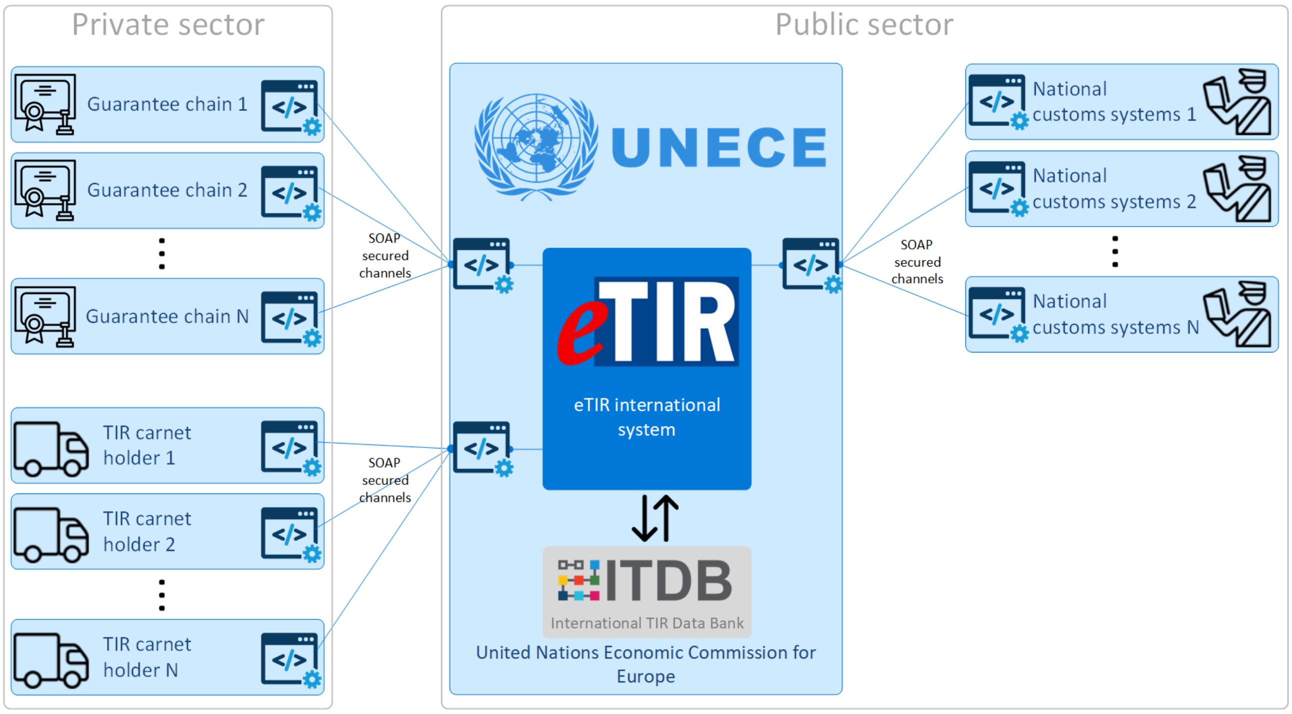
Technology stack:

The eTIR international system has been implemented using the following technologies:

- Java programming language
- PostgreSQL database
- Spring framework
- ActiveMQ
- SOAP-XML web services
- Apache CFX
- Apache Camel

Although these technologies are only listed for information, if the reader has any questions on the implementation methods and or on the technology stack used to implement the eTIR international system, please contact the eTIR service desk at etir@un.org.

The eTIR international system exposes secured web service endpoints with both the public and the private sector partners following the diagram below:



Note that at the moment there is only one guarantee chain in activity (namely the International Road Transport Union) but that the TIR Convention does not limit it to one.

6.2. Interface message overview

The list of eTIR messages are categorized internally, based on the message sender/addressee and based on message direction (eTIR centric):

- All messages codes start either:
  - by "I" for Internal (Internal to the public sector, meaning between the eTIR international system and either a national customs systems or ITDB)
  - by "E" for External (External to the public sector, meaning between eTIR and either a guarantee chain or a TIR Carnet holders)
- All messages work by pair (request-response), and their codes all end either:
  - by an "odd number" for the initiating/calling/request message
  - by an "even number" for the response/acknowledgment message

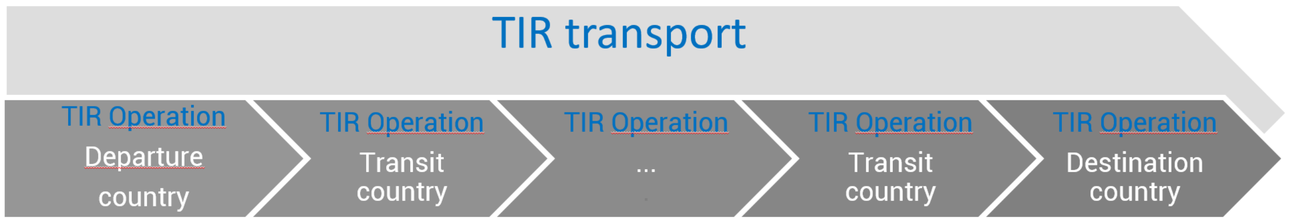
Find below the list of all internal and external messages:

External messages	Internal messages
E1 - Register guarantee	**I1 - Accept guarantee**
E2 – Register results	*I2 - Acceptance results*
E3 – Cancel guarantee	**I3 - Get holder information**
E4 – Cancellation results	*I4 - Holder information*
E5 – Query guarantee	**I5 - Query guarantee**
E6 – Query results	*I6 - Query results*
E7 – Notify guarantee chain	**I7 - Record declaration data**
E8 – Notification confirmation	*I8 - Record declaration data results*
E9 - Advance TIR data	**I9 - Start TIR operation**
E10 - Advance TIR data results	*I10 - Start results*
E11 - Advance amendment data	**I11 - Terminate TIR operation**
E12 - Advance amendment data results	*I12 - Termination results*
E13 - Cancel advance data	**I13 - Discharge TIR operation**
E14 - Cancel advance data results	*I14 - Discharge results*
	I15 - Notify customs
	I16 - Notification confirmation
	I17 - Refusal to start TIR operation
	I18 - Refusal results
	I19 - Check customs offices
	I20 - Customs offices validation

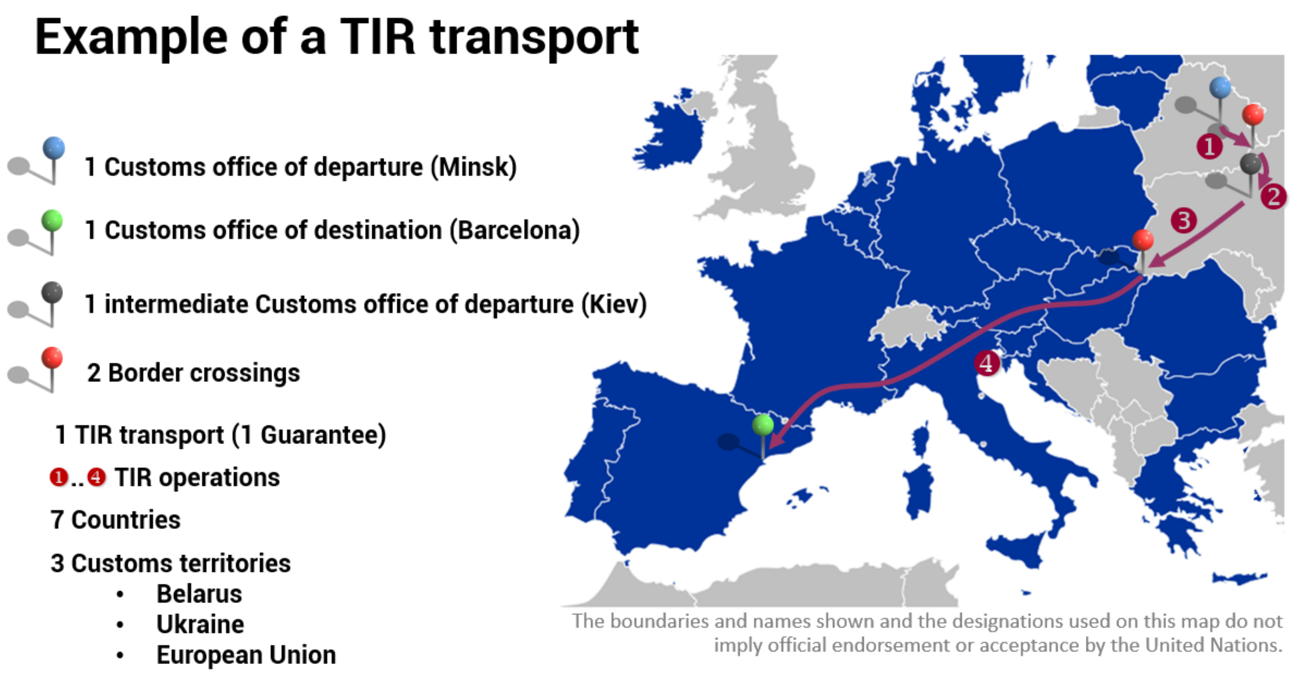
The eTIR international system aims to ensure the secure exchange of data between the national customs systems related to the international transit of goods, vehicles or containers according to the provisions of the TIR Convention and to allow customs to manage the data on guarantees, issued by guarantee chains, to holders authorized to use the TIR system.

6.3. TIR transport and TIR operation in eTIR

The diagram below highlights the important concepts of **TIR transport** and **TIR operations** and gives an example of them:



It is important to note that a TIR transport may have multiple loading and unloading points, and of course multiple border crossings. Each of them separates the TIR operations. Also note that the border crossings separating the TIR operations are between each customs territories (that can be a country, or a common customs territory like in case of the European Union). More details can be found in the dedicated page of the eTIR introduction document.


Example of a TIR transport

1 Customs office of departure (Minsk)

1 Customs office of destination (Barcelona)

1 intermediate Customs office of departure (Kiev)

2 Border crossings

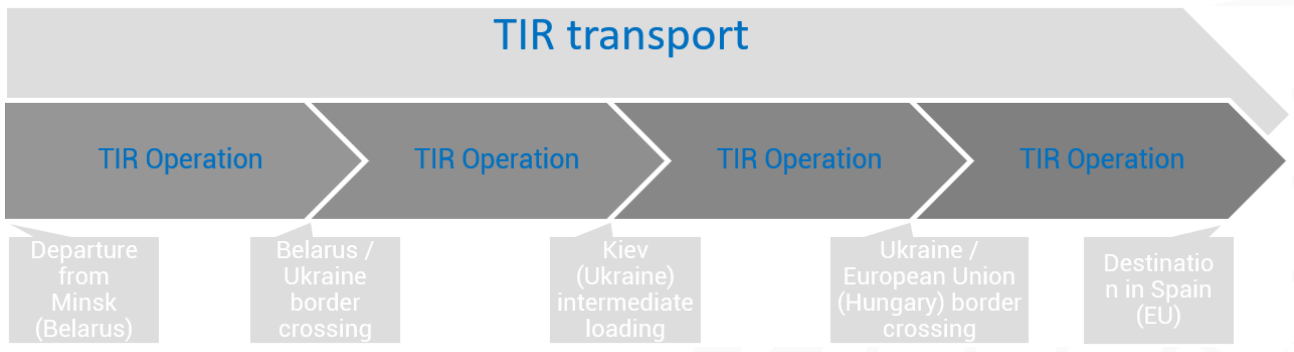
1 TIR transport (1 Guarantee)

❶..❹ TIR operations

7 Countries

3 Customs territories
- Belarus
- Ukraine
- European Union

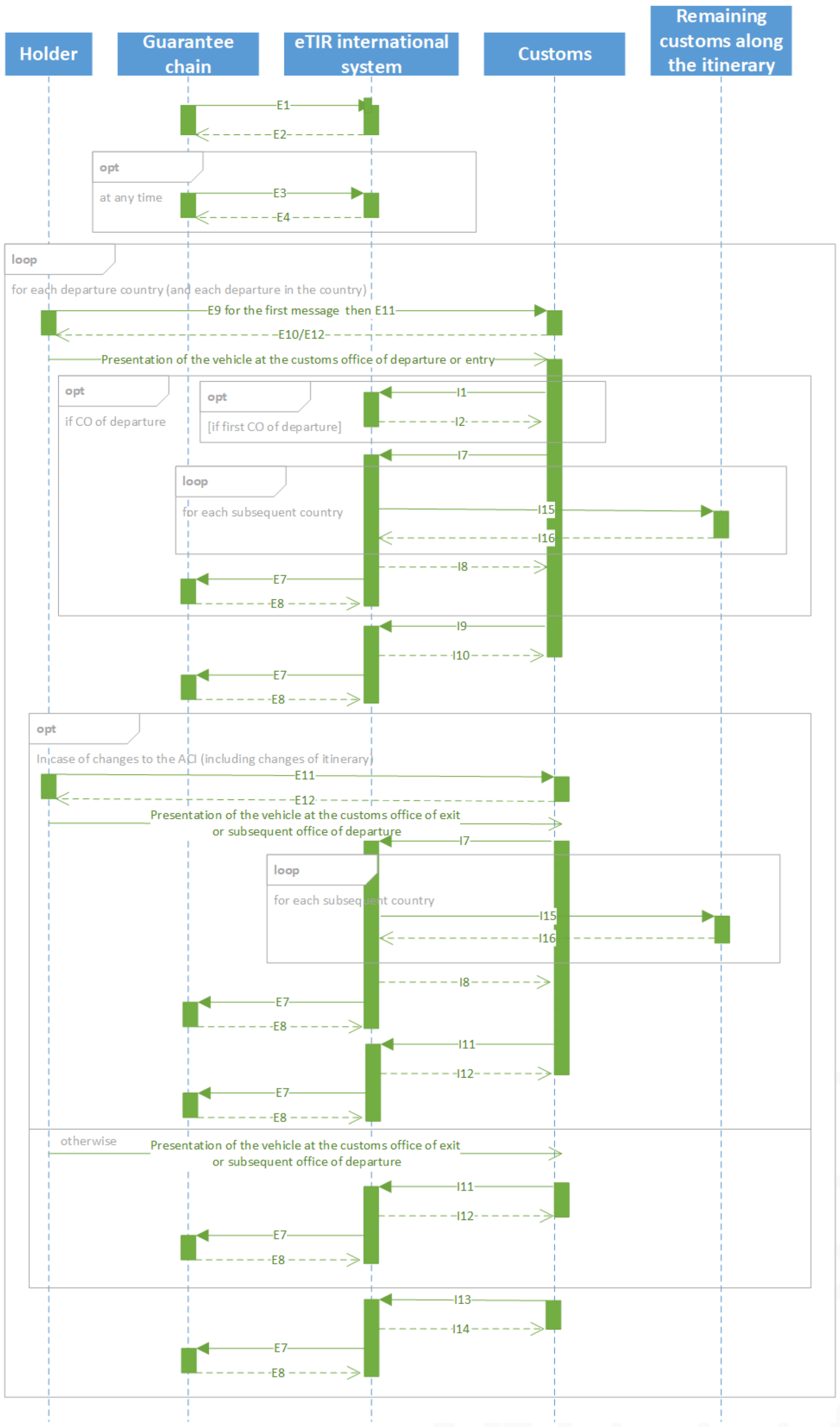
The boundaries and names shown and the designations used on this map do not imply official endorsement or acceptance by the United Nations.


TIR transport
TIR Operation
TIR Operation
TIR Operation
TIR Operation
Departure from Minsk (Belarus)
Belarus / Ukraine border crossing
Kiev (Ukraine) intermediate loading
Ukraine / European Union (Hungary) border crossing
Destination in Spain (EU)

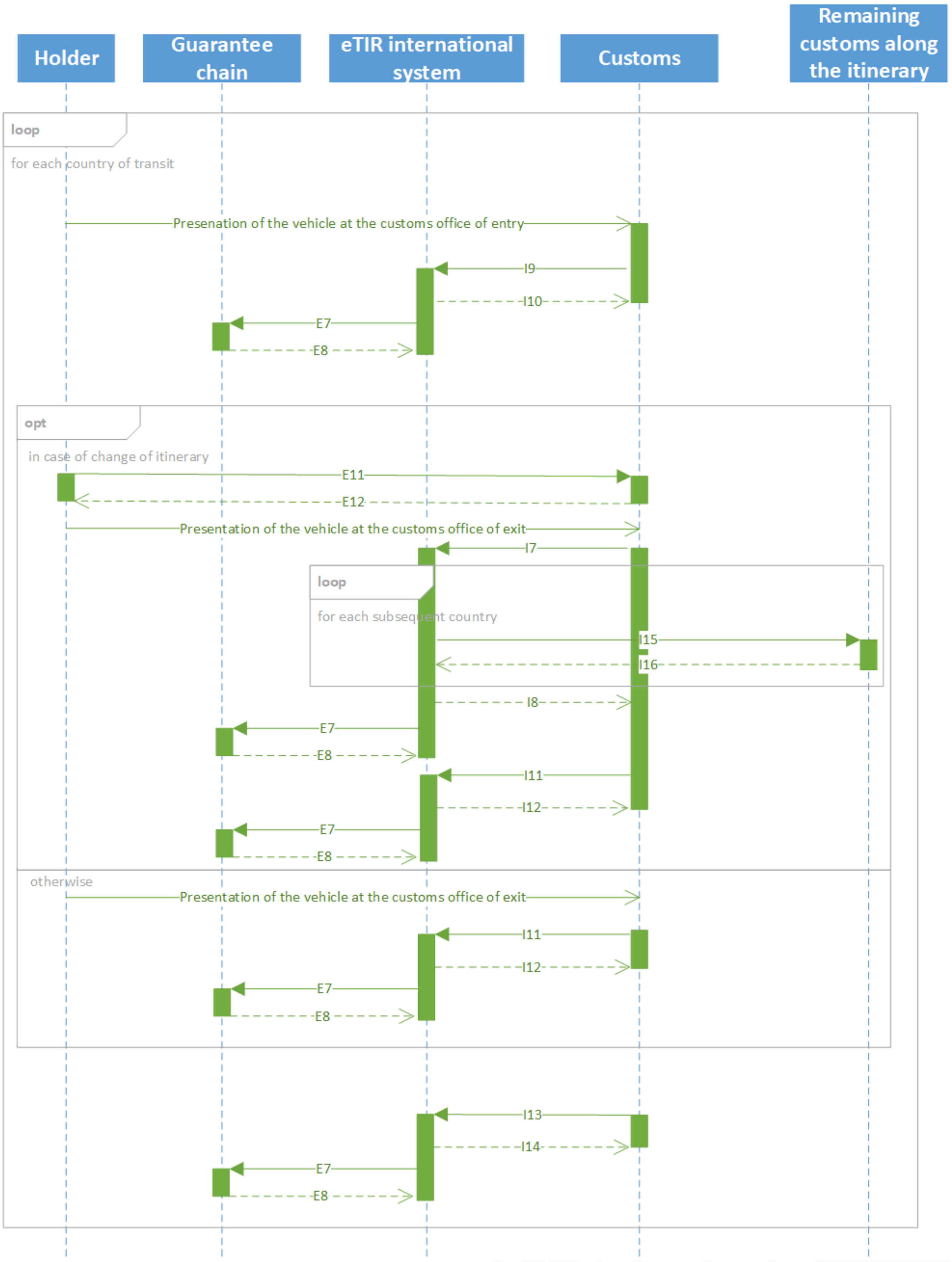
6.4. eTIR sequence diagrams

The usage of the eTIR messages is described in the following eTIR sequence diagrams for countries of departure, transit and destination:

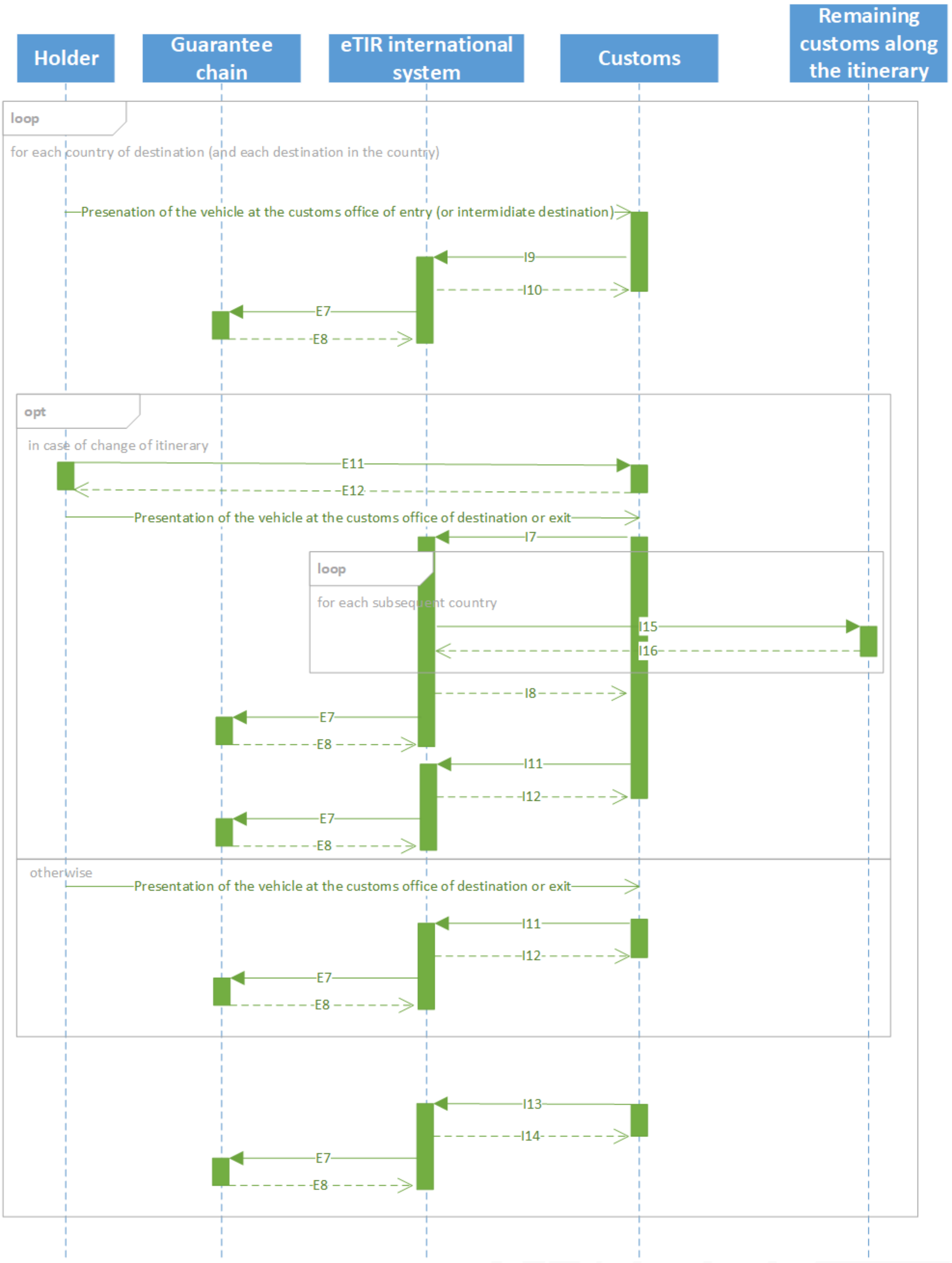
6.4.1. Message sequence for countries of departure


Holder
Guarantee chain
eTIR international system
Customs
Remaining customs along the itinerary
E1
E2
opt
at any time
E3
E4
loop
for each departure country (and each departure in the country)
E9 for the first message then E11
E10/E12
Presentation of the vehicle at the customs office of departure or entry
opt
if CO of departure
opt
[if first CO of departure]
I1
I2
I7
loop
for each subsequent country
I15
I16
I8
E7
E8
I9
I10
E7
E8
opt
In case of changes to the ACI (including changes of itinerary)
E11
E12
Presentation of the vehicle at the customs office of exit or subsequent office of departure
I7
loop
for each subsequent country
I15
I16
I8
E7
E8
I11
I12
E7
E8
otherwise
Presentation of the vehicle at the customs office of exit or subsequent office of departure
I11
I12
E7
E8
I13
I14
E7
E8

6.4.2. Message sequence for countries of transit


Holder
Guarantee chain
eTIR international system
Customs
Remaining customs along the itinerary
loop
for each country of transit
Presenation of the vehicle at the customs office of entry
I9
I10
E7
E8
opt
in case of change of itinerary
E11
E12
Presentation of the vehicle at the customs office of exit
I7
loop
for each subsequent country
I15
I16
I8
E7
E8
I11
I12
E7
E8
otherwise
Presentation of the vehicle at the customs office of exit
I11
I12
E7
E8
I13
I14
E7
E8

6.4.3. Message sequence for countries of destination


Holder
Guarantee chain
eTIR international system
Customs
Remaining customs along the itinerary
loop
for each country of destination (and each destination in the country)
Presenation of the vehicle at the customs office of entry (or intermidiate destination)
I9
I10
E7
E8
opt
in case of change of itinerary
E11
E12
Presentation of the vehicle at the customs office of destination or exit
I7
loop
for each subsequent country
I15
I16
I8
E7
E8
I11
I12
E7
E8
otherwise
Presentation of the vehicle at the customs office of destination or exit
I11
I12
E7
E8
I13
I14
E7
E8

7. Security aspects

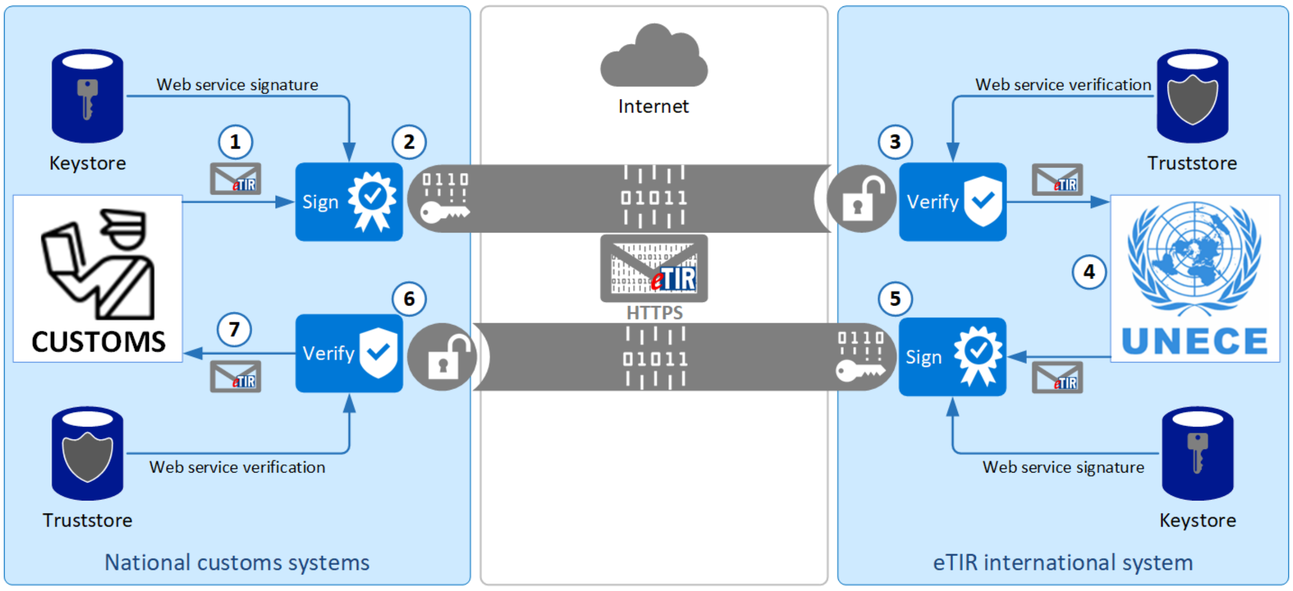
The eTIR international system web services uses WS-Security to electronically sign the messages. WS-Security is not used to provide encryption of the message. This framework has been released as a full industry-recognized recommendation in March 2004.

Digital signature, using X.509 certificates (version 3), are used to identify the caller to the web service and provide non-repudiation capability. An encryption protocol (TLS v1.2 or v1.3) is used to send the messages over HTTPS to ensure the confidentiality of the information exchanged in the messages. It should be noted that the digital signature and the encryption protocol (TLS) use different asymmetric key pairs.

The next paragraphs describe the web services, security related terms and concepts that are used throughout this document. They often refer on the terms "Web Services Security (WS-Security)", "X.509 token", "keystore" and "truststore" which are explained in detail in the Glossary.

7.1. eTIR security model

The figure below depicts the WS-Security model and this section provides an overview illustrating how this security model works.



Kindly note that if this diagram represents most of the use cases, in a few other cases the eTIR international system actually play the role of client, and the national customs systems, the role of service provider.

In the example above, as a first step, the X.509 certificate of the national customs systems will be installed in the eTIR international system truststore. Similarly, the eTIR international system certificate will be installed in the national customs systems truststore. This mandatory initial step allows the validation of the digital signatures that are transferred as security tokens in all SOAP messages exchanged in the context of the eTIR procedure.

Kindly find below a description of how a request is sent by the national customs systems to the eTIR international system, and how the related response is sent back:

1. The national customs systems generates a request message to be sent to the eTIR international system web service.

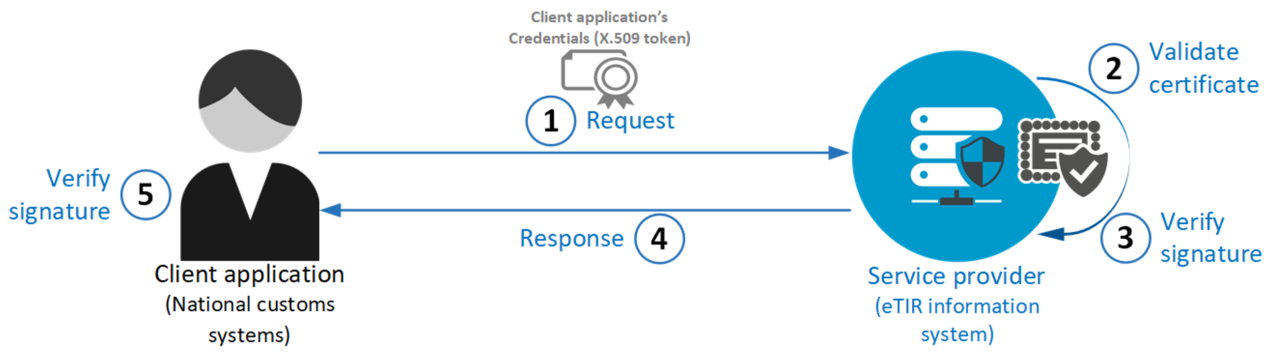
2. The request message is signed with the private key of the national customs systems' X.509 certificate (stored in the keystore) and sent using an encryption protocol (TLS) over HTTPS.
3. The eTIR international system receives the request message, verifies the signature of the message using the public key of the sender to authenticate it and to confirm its integrity.
4. The eTIR international system processes the request message and generates a response message in return.
5. The response message is signed with the private key of the eTIR international system (stored in the keystore) and sent using an encryption protocol (TLS) over HTTPS.
6. The national customs systems receives the response message, and verifies the signature of the message using the public key of the service provider to authenticate it and to confirm its integrity.

The completion of this process illustrates the implementation of four security aspects that the eTIR international system wishes to achieve with this security model: **authentication**, **integrity**, **confidentiality** and **non-repudiation**.

7.2. Authentication, Integrity and Non-repudiation

Authentication is used to ensure that the different parties of a transaction are really who they claim to be; thus, a proof of identity is required. This proof can be claimed in various ways. One simple example is to provide a user ID together with a secret password. A more secured approach is to use an X.509 certificate issued from a trusted Certificate Authority (later referred as "CA" in this section). The X.509 certificate identify the end user. In addition to the authentication feature, the private key to the X.509 certificate is also used to sign SOAP messages. This electronic signature not only ensures the identity of the sender but also guarantees that the message content has not been tampered during the transmission, thus ensuring integrity. The sender uses his private key to sign the message (the hash of the message to be more precise) and the recipient of the message uses the sender's public key to verify the signature, thus providing proofs of non-repudiation.

7.2.1. Authentication steps using an X.509 certificate



The client application (national customs systems) sends a message to the service provider (eTIR international system - Step #1). The message includes the client application's credentials, signed with the private key that is paired with the public key in the client application's X.509 certificate. The service provider validates the certificate by performing a number of checks (Step #2), including:

- Verifying that the certificate has not expired. If the expiration date of the certificate is past due, then the certificate is no longer valid.
- Verifying that the certificate is internally consistent. The service checks that the data in the certificate has not been tampered with by verifying the certificate contents against the signature of the issuing CA.
- Verifying the issuing CA of the client's X.509 certificate. This is done by comparing the issuer's signature on the client application's X.509 certificate with the X.509 certificate of the issuing CA.

For this step to work properly, the CA that issued the client's X.509 certificate must be trusted by both the client application and the service provider.

- Verifying that the issuing CA has not revoked the certificate. The service provider checks this by making sure that the X.509 certificate does not appear on a Certificate Revocation List (CRL) published by the issuing CA. The service can check the revocation status of the certificate by directly accessing it from the CA or by checking against a CRL that was previously downloaded from the issuing CA to the certificate repository used by the service to look up X.509 certificates.
- The service provider uses the public key in the client application's X.509 certificate to verify the client application's signature (Step #3). This allows the service provider to authenticate the client application and ensure that the signed data has not been tampered with after the message was signed.

If the authentication is unsuccessful, the service provider then sends back an HTTP 500 error message. If it is successful, it will process the content of the request and produce the appropriate response message following the eTIR specifications (Step #4) that will, upon reception, require the same checks described above (Step #5).

7.2.2. Key pairs generation

Customs can obtain a public and private key pair from a trusted certificate authority or create self-signed ones. They should generate an RSA key pair and provide to the UNECE TIR secretariat team the public part (X.509 certificate) of the generated key and keep well protected in their keystore the private key. See in the Annexes the description of the two ways to generate an RSA key pair.

7.3. Confidentiality

The HyperText Transfer Protocol Secure (HTTPS), used to access the eTIR international system endpoints, is an extension of the HyperText Transfer Protocol (HTTP) where communication is encrypted using Transport Layer Security (TLS), a cryptographic protocol designed to provide communications security over public networks like the Internet. The principal reason for using HTTPS is the need to ensure confidentiality and integrity of the exchanged data while in transit. It protects against information security threats like the Man-in-the-middle attack. The bidirectional encryption of communications between a client and server protects against from eavesdropping and tampering of the communication. Therefore, communication between the eTIR international system and the national customs systems is secured by the bidirectional encryption using TLS (at least v1.2).

8. Access to eTIR web services

8.1. Prerequisites

8.1.1. Network requirements

The national customs systems must have access to the Internet and all the domains of the User Acceptance Testing and Production environments described in the section below should be white-listed by the Network engineers of the customs authorities. In addition, in the eTIR international system context, SOAP is implemented to use HTTPS on port TCP/8083, therefore this protocol and port should be opened in the firewalls of the customs authorities to be usable by the national customs systems.

8.1.2. Certificates

As explained in the Security aspects section, the connection between the eTIR international system and the national customs systems requires both parties to share their X.509 certificates and to mutually store them in their truststores.

This procedure must be completed for each client-service environment pair during the implementation phase, the national customs systems User acceptance Testing (UAT) environment should be connected to the eTIR international system UAT environment. Once ready for production, both parties Production environments should exchange their dedicated Production X.509 certificates to allow for communication between them and usage of the eTIR procedure when the launch is confirmed.

Both systems (the eTIR international system and national customs systems) should use different X.509 certificates for their respective UAT and Production environments. Also note that if this documentation refers to methods to generate locally certificates using an open source application that may be useful for test and development purposes, the certificates generated and shared by the customs ICT team is entirely up to their choice so long as it respects the security requirements and aspects mentioned in this document.

In addition to the generated X.509 certificate, the ECE server certificate (unece.org.cer) will be used in the encryption mechanism (HTTPS) of the message. Depending on your case, you may need to retrieve and store it in your truststore or your application may retrieve it automatically (preferred option).

During the certificate generation process explained in the KeyStore: step by step generation of key pairs section, the email field should be correctly filled as it will be used by the eTIR international system to send email notifications.

8.1.3. Whitelisting servers

The servers of the national customs systems first need to be whitelisted by the ECE IT security team to allow the connection to the servers of the eTIR international system. To do this, the IT experts of the customs authorities need to provide the TIR secretariat with all the IP addresses of the servers of the national customs systems which will send request to the eTIR international system. Similarly, the TIR secretariat stands ready to provide the IP addresses of its servers to perform the same operation on the customs authorities side.

8.2. Test and production URLs

The list below provides the information about the web service end-points both for the production and UAT environments as well as the accompanying URLs for the Web Service Description Language (WSDL) files:

UAT web service base URL
: https://etir-uat-01.unece.org/etir/v4.3/customs

UAT WSDL URLs
: https://etir-uat-01.unece.org/etir/v4.3/customs?wsdl

Production web service base URL
: https://etir-prod.unece.org/etir/v4.3/customs

Production WSDL URL
: https://etir-prod.unece.org/etir/v4.3/customs?wsdl

9. Implementation and test of eTIR messages

9.1. Recommended general approach

9.1.1. Purpose

In this section, the TIR secretariat wishes to share the development processes followed to ensure a timely and qualitative delivery of the eTIR international system, hoping that by doing so, the reader may in turn be able to benefit from our lessons learned.

9.1.2. General approach

The TIR secretariat adopted an Agile approach following the principles of the Agile manifesto. The members of the IT team meets every week to review the work accomplished during the previous week, to discuss about work to be done during the current week and to mention any potential impediment on which other team members could bring their help. To support the work of the TIR secretariat, an internal Knowledge Management System (KMS) features the following components:

- An issue-tracking system which manages all tasks to be done, which offers excellent traceability and accountability;
- A documentation platform which hosts all development, managerial and operational aspects of the eTIR project;
- A source code management system which hosts the code repositories of the TIR information systems.

9.1.3. Continuous Integration

The TIR secretariat also adopted best practices from the DevOps community. In particular, we recognize the important added value of automating as many processes as possible in the software development lifecycle to relieve human beings from mundane tasks, increase reliability by reducing the probability of human errors, and drastically increasing productivity.

In order to have a system as reliable as possible, we deploy regularly our new code through our continuous integration (CI) pipeline. We also rebuild our entire code base every time new code is committed to our code repository, to ensure that at any given time, every completed functionality works as intended.

9.1.4. Extensive testing

We use automated a static code analysis tool to check on a daily basis our source code against sets of rules and best practices created by the IT industry community. This allows to ensure a high quality of the source code, and a continuous training of the members of the IT team.

We also have a target of 70% of our functional code base covered by unit tests, as well as a comprehensive suite of functional non-regression tests written with Apache JMeter. By combining these two types of tests, we ensure that every part of the eTIR international system works as intended, and we protect ourselves from possible regressions when adding new code to our code base.

9.2. Tool kit: list of what we believe is useful

The list below represents the various softwares that are believed to be useful for the implementation of a client application to the eTIR international system and for which the TIR secretariat has experience with:

- SOAP-UI testing suite: a useful tool for performing ad-hoc tests on SOAP messages (note that you can find in annex the SOAP UI quick guide);
- JMeter testing suite: a useful tool for automating message testing and covering a large scale of scenarios;
- GIT version control system: the industry leading version control system, ensuring that all eTIR international system code and other configuration items are properly versioned;
- IntelliJ IDEA: one of the industry leading Integrated Development Environment (IDE) fitting the eTIR international system technology stack, that proved to enhance our productivity on various aspects;
- SonarQube static code analysis tool: software used for continuous inspection of the code quality that performs automatic reviews with static analysis of the code to detect bugs, so-called "code smells" (which are coding bad practices), and security vulnerabilities.

Kindly note that installing and using these applications is not required. However, those tools have proven to be useful to the TIR secretariat team during the implementation of the eTIR international system.

9.3. SOAP Header security generation

As a SOAP extension, WS-Security provides the SOAP header element titled "Security", which is designed to act as a container to store all security related information for SOAP request and response messages. Here is how the **Security** element is defined in the WS-Security schema.

9.3.1. Security Element

The **wsse:Security** header element provides a mechanism for attaching security-related information targeted at a specific recipient in the form of a SOAP actor/role. This element represents the signing steps the message producer took to create the message. This prepending rule ensures that the receiving application can process sub-elements in the order they appear in the **wsse:Security** header element, because there will be no forward dependency among the sub-elements.

Signature element

The **Signature** element is the root element of an XML Signature

Schema Definition

```
<element name="Signature" type="ds:SignatureType"/>
  <complexType name="SignatureType">
    <sequence>
      <element ref="ds:SignedInfo"/>
      <element ref="ds:SignatureValue"/>
      <element ref="ds:KeyInfo" minOccurs="0"/>
      <element ref="ds:Object" minOccurs="0" maxOccurs="unbounded"/>
    </sequence>
    <attribute name="Id" type="ID" use="optional"/>
</complexType>
```

In message example:

```
<ds:Signature Id="SIG-AD473EF9595256C9D11540973359402173"
        xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
        <ds:SignedInfo>
          <ds:CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#">
            <ec:InclusiveNamespaces PrefixList="SOAP-ENV cus urn"
              xmlns:ec="http://www.w3.org/2001/10/xml-exc-c14n#"/>
          </ds:CanonicalizationMethod>
          <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
          <ds:Reference URI="#id-AD473EF9595256C9D11540973359401172">
            <ds:Transforms>
              <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#">
                <ec:InclusiveNamespaces PrefixList="cus urn"
                  xmlns:ec="http://www.w3.org/2001/10/xml-exc-c14n#"/>
              </ds:Transform>
            </ds:Transforms>
            <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
            <ds:DigestValue>H2Ai…</ds:DigestValue>
          </ds:Reference>
        </ds:SignedInfo>
        <ds:SignatureValue>bY65hsJdkxh40…
        </ds:SignatureValue>
        <ds:KeyInfo Id="KI-AD473EF9595256C9D11540973359401170">
          <wsse:SecurityTokenReference wsu:Id="STR-AD473EF9595256C9D11540973359401171">
            <wsse:KeyIdentifier EncodingType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-
security-1.0#Base64Binary"
                                ValueType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-
profile-1.0#X509v3">
              G9w0BAQsFAAN...
            </wsse:KeyIdentifier>
          </wsse:SecurityTokenReference>
        </ds:KeyInfo>
      </ds:Signature>
```

SignatureValue element

The **SignatureValue** element contains the information about the sender's public key which is needed for the eTIR international system to validate the digital signature of the caller. It is always encoded using base64.

Schema Definition

```
<element name="SignatureValue" type="ds:SignatureValueType" />

  <complexType name="SignatureValueType">
    <simpleContent>
      <extension base="base64Binary">
        <attribute name="Id" type="ID" use="optional"/>
      </extension>
    </simpleContent>
  </complexType>
```

In message example:

```
<ds:SignatureValue>
    bY65hsJdkxh40…
</ds:SignatureValue>
```

The **SignatureValue** element contains a signature that only covers the **SignedInfo** element: only the content of this **SignedInfo** element is included into the signature digest.

SignedInfo element

The **SignedInfo** element describes which part of the message was used to generate the signature. It has an attribute **Id** pointing to the SOAP body part of the signed message and we will explain it in more detail in the next element (**Reference**). The structure of the **SignedInfo** element includes the canonicalization algorithm, a signature algorithm, and one or more references. The content of the **SignedInfo** element can be divided into two parts: information about the **SignatureValue** and information about the application content, as we can see in the following XML Schema fragment:

Schema definition:

```
<element name="SignedInfo" type="ds:SignedInfoType"/>

  <complexType name="SignedInfoType">
    <sequence>
      <element ref="ds:CanonicalizationMethod"/>
      <element ref="ds:SignatureMethod"/>
      <element ref="ds:Reference" maxOccurs="unbounded"/>
    </sequence>
    <attribute name="Id" type="ID" use="optional"/>
  </complexType>
```

In message example:

```
<ds:SignedInfo>
  <ds:CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#">
    <ec:InclusiveNamespaces PrefixList="SOAP-ENV cus urn"
      xmlns:ec="http://www.w3.org/2001/10/xml-exc-c14n#"/>
  </ds:CanonicalizationMethod>
  <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
  <ds:Reference URI="#id-AD473EF9595256C9D11540973359401172">
    <ds:Transforms>
      <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#">
        <ec:InclusiveNamespaces PrefixList="cus urn"
          xmlns:ec="http://www.w3.org/2001/10/xml-exc-c14n#"/>
      </ds:Transform>
    </ds:Transforms>
    <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
    <ds:DigestValue>H2Ai…</ds:DigestValue>
  </ds:Reference>
</ds:SignedInfo>
```

Canonicalization, or C14N (exclusive XML Canonicalization), is the process of picking one path through all the possible output options, so that sender and receiver can generate the exact same byte value, no matter what intermediate XML software might be involved. The **SignedInfo/CanonicalizationMethod** element specifies how to reconstruct the exact byte stream. The **SignedInfo/SignatureMethod** element specifies what type of algorithm (e.g., Kerberos or RSA) is used to generate the signature. Together, these two elements describes how to create the digest, and how to protect it from any modification.

In a java based client, this can be configured as following:

```
org.apache.wss4j.dom.message.WSSecSignature wsSecSignature = new WSSecSignature();
//http://www.w3.org/2001/10/xml-exc-c14n#
wsSecSignature.setSigCanonicalization(WSConstants.C14N_EXCL_OMIT_COMMENTS);
//http://www.w3.org/2000/09/xmldsig#rsa-sha1
wsSecSignature.setSignatureAlgorithm(WSConstants.RSA_SHA1);
```

Reference element

The **Reference** element is used to refer to another content. It contains a digest of the content, a description of how that digest was generated (e.g. SHA1), and a specification of how the content should be transformed before the digest is generated. The transformations provide flexibility on how the XML signature is built.

Schema definition:

```
<element name="Reference" type="ds:ReferenceType"/>

  <complexType name="ReferenceType">
    <sequence>
      <element ref="ds:Transforms" minOccurs="0"/>
      <element ref="ds:DigestMethod"/>
      <element ref="ds:DigestValue"/>
    </sequence>
    <attribute name="Id" type="ID" use="optional"/>
    <attribute name="URI" type="anyURI" use="optional"/>
    <attribute name="Type" type="anyURI" use="optional"/>
  </complexType>
```

In message example:

```
<ds:Reference URI="#id-AD473EF9595256C9D11540973359401172">
  <ds:Transforms>
    <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#">
      <ec:InclusiveNamespaces PrefixList="cus urn"
        xmlns:ec="http://www.w3.org/2001/10/xml-exc-c14n#"/>
    </ds:Transform>
  </ds:Transforms>
  <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
  <ds:DigestValue>H2Ai…</ds:DigestValue>
</ds:Reference>
```

The **DigestMethod** element specifies the hashing algorithm, and the **DigestValue** element is the value of the hash of the content encoded in base64. The key part of the **Reference** element is the set of transforms that may be used. The **Transforms** element is a list of **Transform** elements, each of which specifies a transformation step.

Transforms element

The schema defines an array of transforms, with one XPath element that has a defined structure:

Schema definition:

```
<element name="Transforms" type="ds:TransformsType"/>
  <complexType name="TransformsType">
    <sequence>
      <element ref="ds:Transform" maxOccurs="unbounded"/>
    </sequence>
  </complexType>

<element name="Transform" type="ds:TransformType"/>
  <complexType name="TransformType" mixed="true">
    <choice minOccurs="0" maxOccurs="unbounded">
      <any namespace="##other" processContents="lax"/>
      <element name="XPath" type="string"/>
    </choice>
    <attribute name="Algorithm" type="anyURI" use="required"/>
  </complexType>
```

The content in a **Transform** element will depend on the Algorithm attribute. For example, if simple XML is being signed, then there will most likely be a single **Transform** element that specifies a C14N algorithm:

In message example:

```
<ds:Transforms>
  <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#">
    <ec:InclusiveNamespaces PrefixList="cus urn"
      xmlns:ec="http://www.w3.org/2001/10/xml-exc-c14n#"/>
  </ds:Transform>
</ds:Transforms>
```

KeyInfo element

The last step is to identify the signer, or at least the key that generated the signature (the key that protects the digest from being modified). This task is ensured by the **KeyInfo** element:

Schema definition:

```
<element name="KeyInfo" type="ds:KeyInfoType"/>

  <complexType name="KeyInfoType" mixed="true">
    <choice maxOccurs="unbounded">
      <element ref="ds:KeyName"/>
      <element ref="ds:KeyValue"/>
      <element ref="ds:RetrievalMethod"/>
      <element ref="ds:X509Data"/>
      <element ref="ds:PGPData"/>
      <element ref="ds:SPKIData"/>
      <element ref="ds:MgmtData"/>
      <any processContents="lax" namespace="##other"/>
    </choice>
    <attribute name="Id" type="ID" use="optional"/>
  </complexType>
```

In message example:

```
<ds:KeyInfo Id="KI-AD473EF9595256C9D11540973359401170">
  <wsse:SecurityTokenReference wsu:Id="STR-AD473EF9595256C9D11540973359401171">
    <wsse:KeyIdentifier EncodingType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-
1.0#Base64Binary"
                        ValueType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-
1.0#X509v3">
              G9w0BAQsFAAN...
    </wsse:KeyIdentifier>
  </wsse:SecurityTokenReference>
</ds:KeyInfo>
```

SecurityTokenReference element

The digital signature operation requires that a key be specified. The element containing the key in question may be located elsewhere in the message or completely outside the message. The **SecurityTokenReference** element provides a mechanism for referencing security tokens and other key bearing elements. The **SecurityTokenReference** element provides an open content model for referencing key bearing elements. It should contain either a **keyIdentifier** element or a **X509Data** element. If we don't specify the **KeyIdentifier** configurations by default **wsse:SecurityTokenReference** will contain a **X509Data** element and both are accepted.

The following configuration will set the **keyIdentifier** element instead of using a **X509Data** element:

```
org.apache.wss4j.dom.message.WSSecSignature wsSecSignature = new WSSecSignature();
wsSecSignature.setKeyIdentifierType(WSConstants.X509_KEY_IDENTIFIER);
```

KeyIdentifier element

A token should be referenced using a **KeyIdentifier** element. The following table list both encoding and value types.

URI Fragment	URL
#Base64Binary	http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0#Base64Binary
#X509v3	http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509v3

In message example:

```
<ds:KeyInfo Id="KI-AD473EF9595256C9D11540973359401170">
  <wsse:SecurityTokenReference wsu:Id="STR-AD473EF9595256C9D11540973359401171">
    <wsse:KeyIdentifier EncodingType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-
1.0#Base64Binary"
                        ValueType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-
1.0#X509v3">
        G9w0BAQsFAAN...
    </wsse:KeyIdentifier>
  </wsse:SecurityTokenReference>
</ds:KeyInfo>
```

X509Data element

Schema definition:

```
<complexType name="X509IssuerSerialType">
  <sequence>
    <element name="X509IssuerName" type="string"/>
    <element name="X509SerialNumber" type="integer"/>
  </sequence>
</complexType>
```

X.509 certificates are supported through the **ds:X509Data** element. This element allows the signer to embed its certificate (encoded in Base64), or any other alternative methods to verify the certificate: a subject's name, the issuer's name and serial number, the key identifier, or another format. The signer can also include a current copy of the Certificate Revocation List (CRL), to show that the signer's identity was valid at the time the document was signed.

9.3.2. SOAP header example

```xml
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:cus="http://etir.unece.org/services/CustomsToETIR-1"
  xmlns:urn="urn:wco:datamodel:WCO:I5:1">
  <SOAP-ENV:Header>
    <wsse:Security xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd"
      xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd">
      <ds:Signature Id="SIG-AD473EF9595256C9D11540973359402173"
        xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
        <ds:SignedInfo>
          <ds:CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#">
            <ec:InclusiveNamespaces PrefixList="SOAP-ENV cus urn"
              xmlns:ec="http://www.w3.org/2001/10/xml-exc-c14n#"/>
          </ds:CanonicalizationMethod>
          <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
          <ds:Reference URI="#id-AD473EF9595256C9D11540973359401172">
            <ds:Transforms>
              <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#">
                <ec:InclusiveNamespaces PrefixList="cus urn"
                  xmlns:ec="http://www.w3.org/2001/10/xml-exc-c14n#"/>
              </ds:Transform>
            </ds:Transforms>
            <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
            <ds:DigestValue>H2Ai…</ds:DigestValue>
          </ds:Reference>
        </ds:SignedInfo>
        <ds:SignatureValue>bY65hsJdkxh40…
        </ds:SignatureValue>
        <ds:KeyInfo Id="KI-AD473EF9595256C9D11540973359401170">
          <wsse:SecurityTokenReference wsu:Id="STR-AD473EF9595256C9D11540973359401171">
            <wsse:KeyIdentifier EncodingType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-
security-1.0#Base64Binary" ValueType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-
1.0#X509v3">
              G9w0BAQsFAAN...
            </wsse:KeyIdentifier>
          </wsse:SecurityTokenReference>
        </ds:KeyInfo>
      </ds:Signature>
    </wsse:Security>
    <wsa:Action>http://etir.unece.org/services/GuaranteeChainToETIR-1/queryGuarantee </wsa:Action>
    <wsa:MessageID>uuid:8a20af11-8170-495d-9563-6a89b32ef745</wsa:MessageID>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
wsu:Id="id-942cd702-1de3-4f27-bfcd-6628ab5d3143">
    <cus:queryGuarantee>
      <urn:InterGov>
        <urn:FunctionCode>9</urn:FunctionCode>
        <urn:ID>uuid-query</urn:ID>
        <urn:TypeCode>I5</urn:TypeCode>
        <urn:ReplyTypeCode>00</urn:ReplyTypeCode>
        <urn:ObligationGuarantee>
          <urn:ReferenceID>XC95xxxxxx</urn:ReferenceID>
        </urn:ObligationGuarantee>
      </urn:InterGov>
    </cus:queryGuarantee>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

9.4. Implementation of the Message Identifier / Functional Reference

9.4.1. General

All messages sent and received are uniquely identified using the **Message Identifier** field. This field must be set by the Sender in the request message. The Receiver will set another unique value for the **Message Identifier** field of the response message. In addition to that, the Receiver will also set the **Functional Reference** field of the response message with the value of the **Message Identifier** field of the related request message. This method allows a full traceability of the Request/Response messages.

9.4.2. Request message

The Receiver must set the **Message Identifier** field of the message using the following format:

```
SenderID:UUID
```

Where:

- SenderID: a unique value identifying the sending entity. This value should be the same as the one set in the **Sender identification** field in the message.
- UUID: a universally unique identifier v4 as detailed in RFC 4122. Version 4 is based on pseudo-random numbers hence the need to keep the SenderID to decrease the probabilities of collisions between messages sent by various eTIR stakeholders.

The main programming languages provide native helper classes to generate a UUID v4:

In Java:

```
java.util.UUID.randomUUID();
```

In C#:

```
System.Guid.NewGuid();
```

Here are some samples of valid values for the **Message Identifier** field:

Sample 1:

```
<urn:ID>CustomsCountryA:6aca5f82-2285-4f00-b4ae-36269d4cc865</urn:ID>
```

Sample 2:

```
<urn:ID>eTIRInternationalSystem:1486e5b7-c6ae-4d27-b794-44c4bf545fb3</urn:ID>
```

9.4.3. Response message

The Receiver receives the request message from the Sender and stores the value of the **Message Identifier** field. When preparing the response message, the Receiver will set a new value for the **Message Identifier** field of that message following the same way as described in the section above. Then the stored value of the **Message Identifier** field of the request message will be set to the **Functional reference** of the response message being prepared.

9.5. Implementation of the date fields

The eTIR messages contain several fields in which dates have to be entered. For some of these fields, the format only includes a date and for some others, it also includes the time. This section explains more in detail how to properly populate these fields.

9.5.1. Fields with dates only

In the XML Schema Definition (XSD) files, this type is named **EtirDateType** and is defined as follows, along with the types it inherits from.

EtirDateType definition

```
<xs:complexType name="EtirDateType">
  <xs:simpleContent>
    <xs:extension base="ds:DateTimeType_102_S">
      <xs:attribute name="formatCode" use="optional">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:enumeration value="102"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
<xs:simpleType name="DateTimeType_102_S">
  <xs:restriction base="ds:DateTimeType_S">
    <xs:pattern value="[1-9][0-9][0-9][0-9](([0][1|3|5|7|8])([0][1-9]|[1-2][0-9]|[3][0-1])|([0][4|6|9])([0][1-9]|[1-
2][0-9]|[3][0])|([0][2])([0][1-9]|[1-2][0-9])|([1][0|2])([0][1-9]|[1-2][0-9]|[3][0-1])|([1][1])([0][1-9]|[1-2][0-
9]|[3][0]))"/>
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name="DateTimeType_S">
  <xs:restriction base="xs:string">
    <xs:pattern value=".{1,35}"/>
  </xs:restriction>
</xs:simpleType>
```

The format of this type of date is aligned on the UN/EDIFACT format code 102: CCYYMMDD

With:

- CCYY: the year on four digits. Examples: 1979, 2020;
- MM: the month on two digits from 01 to 12 starting with 01 for January;
- DD: the day of the month on two digits from 01 to 31.

Samples:

- 01 January 1970 is coded as "19700101";
- 29 February 2020 is coded as "20200229";
- 31 December 2045 is coded as "20451231".

The date field also contains an optional attribute named **formatCode** which value is therefore always "102". With this format, there is no notion of time zone and the date has to be regarded as valid in all time zones.

The following XML code show a sample date only field.

Expiration of a guarantee on 01 August 2024

```
<ExpirationDateTime formatCode="102">20240801</ExpirationDateTime>
```

9.5.2. Fields with dates and time

In the XML Schema Definition (XSD) files, this type is named **EtirDateTimeType** and is defined as follows, along with the types it inherits from.

EtirDateTimeType definition

```
<xs:complexType name="EtirDateTimeType">
  <xs:simpleContent>
    <xs:extension base="ds:DateTimeType_208_S">
      <xs:attribute name="formatCode" use="optional">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:enumeration value="208"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
<xs:simpleType name="DateTimeType_208_S">
  <xs:restriction base="ds:DateTimeType_S">
    <xs:pattern value="[1-9][0-9][0-9][0-9](([0][1|3|5|7|8])([0][1-9]|[1-2][0-9]|[3][0-1])|([0][4|6|9])([0][1-9]|[1-
2][0-9]|[3][0])|([0][2])([0][1-9]|[1-2][0-9])|([1][0|2])([0][1-9]|[1-2][0-9]|[3][0-1])|([1][1])([0][1-9]|[1-2][0-
9]|[3][0]))(([0-1][0-9])|(2[0-3]))[0-5][0-9](([0-5][0-9]|60))[\-+]((0[0-9])|(1[0-4]))[0-5][0-9]"/>
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name="DateTimeType_S">
  <xs:restriction base="xs:string">
    <xs:pattern value=".{1,35}"/>
  </xs:restriction>
</xs:simpleType>
```

The format of this type of date and time is aligned on the UN/EDIFACT format code 208: CCYYMMDDHHMMSSZHHMM

With, defined sequentially:

- CCYY: the year on four digits. Examples: 1979, 2020;
- MM: the month on two digits from 01 to 12 starting with 01 for January;
- DD: the day of the month on two digits from 01 to 31;
- HH: the hour of the day on two digits from 00 (for midnight) to 23 (for eleven PM);
- MM: the minutes of the day on two digits from 00 to 59;
- SS: the seconds of the day on two digits from 00 to 59. 60 is also allowed in the case of a leap second;
- Z: the introduction of the time zone with either a '' or a '-'. If the time zone has no offset, either '-' or '' can be used;
- HH: the hours of the offset of the time zone from 01 to 14;
- MM: the minutes of the offset of the time zone from 00 to 59.

Samples:

- 01 January 1970 00:00:00 in London (Time offset: +00:00) is coded as "19700101000000+0000";
- 29 February 2020 09:45:36 in New York (Time offset: -05:00) is coded as "20200229094536-0500";
- 31 December 2045 22:06:59 in South Tarawa, Kiribati (Time offset: +14:00) is coded as "20451231220659+1400".

The date and time field also contains an optional attribute named **formatCode** which value is therefore always "208".

The following XML code show a sample date and time field.

Acceptance of a guarantee on 01 July 2021 10:03:42 in Istanbul (Time offset +03:00)

```
<ExpirationDateTime formatCode="102">20210701100342+0300</ExpirationDateTime>
```

9.6. Error management

9.6.1. Introduction to error management

When the eTIR international system receives and processes a message, it performs a series of validations on the message itself, in the context of the related guarantee, holder or transport and issues a response to the system which has sent the message in the first place. If anything goes wrong during these validation and processing steps, a list of errors is sent back in the response. Each of these errors is presented as an **Error code** with a **Pointer** which can be used to point towards a specific XML element of the message. The list of all the error codes is available on the Error code (eTIR) page.

9.6.2. Presentation of error codes

The list of error codes is specific to eTIR as it allows IT teams to better understand errors while implementing the interconnection to the eTIR international system. This should result in a faster implementation overall and more accurate processing of the errors from the system sending messages to the eTIR international system. Furthermore, a detailed error code system will also greatly simplify the communication between the stakeholders and the eTIR service desk, in case of an incident, to identify and correct the underlying problem. The list of error codes is based on the best practices from the IT industry. Like the list of HTTP status codes, all error codes have three digits, and the first digit of the status code defines the type of errors:

- **1XX - Validation**: validation of the message and if its parameters;
- **2XX - Workflow**: workflow related problems;
- **3XX - Functional**: other functional problems;
- **4XX - Internal**: eTIR international system internal problems.

Each type of error has a default error code which indicates, at least, the type of the error if the system cannot send a more explicit error.

9.6.3. Sample errors

Below is an example of how a single error is returned:

Missing required element

```
<ns14:Error>
   <ns14:ValidationCode>101</ns14:ValidationCode>
   <ns14:Pointer>
      <ns14:SequenceNumeric>1</ns14:SequenceNumeric>
      <ns14:DocumentSectionCode>/InterGov/ObligationGuarantee/ReferenceID</ns14:DocumentSectionCode>
   </ns14:Pointer>
</ns14:Error>
```

Here the **ValidationCode** is the error code and the **DocumentSectionCode** element inside the **Pointer** element points towards the problematic element of the request using the XPath syntax.

When multiple errors of the same type are returned, the eTIR international system returns a single error element, with a list of pointer elements.

Example of missing multiple required element

```
<ns14:Error>
   <ns14:ValidationCode>101</ns14:ValidationCode>
   <ns14:Pointer>
      <ns14:SequenceNumeric>1</ns14:SequenceNumeric>
      <ns14:DocumentSectionCode>/InterGov/ObligationGuarantee/ReferenceID</ns14:DocumentSectionCode>
   </ns14:Pointer>
   <ns14:Pointer>
      <ns14:SequenceNumeric>2</ns14:SequenceNumeric>
      <ns14:DocumentSectionCode>/InterGov/ObligationGuarantee/Surety/ID</ns14:DocumentSectionCode>
   </ns14:Pointer>
</ns14:Error>
```

Finally as an example, please find below a complete response to an **I1 - Accept guarantee** message containing multiple errors:

Example of a complete I2 - Acceptance results message body with multiple errors returned

```
<soap:Body xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
          wsu:Id="id-30706360-7e18-4764-b080-05364eb55892">
    <ns3:acceptGuaranteeResponse xmlns:ns3="http://etir.unece.org/services/CustomsToETIR-1"
                               xmlns:ns14="urn:wco:datamodel:WCO:I2:1">
     <ns14:InterGov>
        <ns14:FunctionCode>27</ns14:FunctionCode>
        <ns14:TypeCode>I2</ns14:TypeCode>
        <ns14:Error>
           <ns14:ValidationCode>102</ns14:ValidationCode>
           <ns14:Pointer>
              <ns14:SequenceNumeric>1</ns14:SequenceNumeric>
              <ns14:DocumentSectionCode>/InterGov/FunctionCode</ns14:DocumentSectionCode>
           </ns14:Pointer>
           <ns14:Pointer>
              <ns14:SequenceNumeric>2</ns14:SequenceNumeric>
              <ns14:DocumentSectionCode>/InterGov/TypeCode</ns14:DocumentSectionCode>
           </ns14:Pointer>
        </ns14:Error>
        <ns14:Error>
           <ns14:ValidationCode>101</ns14:ValidationCode>
           <ns14:Pointer>
              <ns14:SequenceNumeric>3</ns14:SequenceNumeric>
              <ns14:DocumentSectionCode>/InterGov/ObligationGuarantee/ReferenceID</ns14:DocumentSectionCode>
           </ns14:Pointer>
           <ns14:Pointer>
              <ns14:SequenceNumeric>4</ns14:SequenceNumeric>
              <ns14:DocumentSectionCode>/InterGov/ObligationGuarantee/Surety/ID</ns14:DocumentSectionCode>
           </ns14:Pointer>
        </ns14:Error>
     </ns14:InterGov>
    </ns3:acceptGuaranteeResponse>
    </soap:Body>
```

9.6.4. Error handling

Each national customs systems which interconnects with the eTIR international system needs to properly handle the errors returned in the response message. When implementing the various pairs of eTIR messages, developers will find it convenient to refer to the eTIR error codes page, especially to the 'Observations' column in the tables, to see which error codes could be raised. When receiving messages from the eTIR international system, errors should be handled in such a way that the relevant information is transmitted to the relevant national customs systems. As all errors are critical and mean a failure to process the message, the appropriate follow-up actions should be performed by the users of the national customs systems.

The same goes for the national customs systems which may send back errors to the eTIR international system when receiving requests that they cannot process (this might be the case for the following pairs of messages: I15/I16, E9/E10, E11/E12 and E13/E14). The national customs systems should follow the same specifications as detailed above when errors have to be reported to the eTIR international system when sending response messages.

10. Fallback procedures

eTIR functional fallback procedures are currently described in the Approved amendments to the eTIR conceptual, functional and technical documentation - v.4.2a that will be included in the next version of the eTIR specification (v4.3). Moreover, each message may have a technical fallback procedure that will be described in the corresponding eTIR message implementation guides.

11. Support and contact

Kindly note that in the context of the interconnections projects by customs, the TIR secretariat stands ready to assist contracting parties while interconnecting their national customs systems to the eTIR international system. Also in case of questions or issues related to this document or to the eTIR international system, you can use the contact details below (contacts by email should be preferred).

Organization	United Nations Economic Commission For Europe TIR secretariat Palais des Nations, 1211 Geneva 10, Switzerland
Contact	Email: etir@un.org Phone: +41 (0)22 917 55 06

12. Annexes

12.1. Version history

Date	Author	Version	Notes	eTIR specification version reference
28/07/2020	UNECE TIR Secreatariat	1.0	Initial draft	4.3a
08/10/2020	UNECE TIR Secreatariat	1.1	Updated part related to security and wording simplification	4.3a

12.2. Glossary

API

An Application Programming Interface (API) is a software interface which is used for accessing an application or a service from a program.

Asymmetric encryption algorithm

A cryptographic system that uses two keys: a public key known to everyone and a private (or secret) key only known by the owner of the key pair. For example, when Alice wants to send a secured message to Bob, she uses Bob's public key to encrypt the message. Bob then uses his private key to decrypt it. RSA is an example of asymmetric algorithm.

Authentication

The process of verifying or testing that the claimed identity is valid is authentication. Authentication requires the subject to provide additional information that corresponds to the identity they are claiming. The most common form of authentication is using a password (this includes the password variations of personal identification numbers (PINs) and passphrases). Authentication verifies the identity of the subject by comparing one or more factors against the database of valid identities (that is, user accounts).

B2B

Business to Business relationship (where business refers to the private sector).

B2C

Business to Customs relationship (where business refers to the private sector).

C2B

Customs to Business relationship (where business refers to the private sector).

C2C

Customs to Customs relationship.

Certification Authority (CA)

A certification authority, or CA, holds a trusted position because the certificate that it issues binds the identity of a person or business to the public and private key pair (asymmetric cryptography) that are used to secure most internet transactions. For example, when a business or person wants to use these technologies, they request to a CA to issue them a certificate. The CA collects information about the person or business that it will certify before issuing the certificate.

Confidentiality

Confidentiality is the concept of the measures used to ensure the protection of the secrecy of data, objects, or resources. The goal of confidentiality protection is to prevent or minimize unauthorized access to data. Confidentiality focuses security measures on ensuring that no one other than the intended recipient of a message receives it or is able to read it. Confidentiality protection provides a means for authorized users to access and interact with resources, but it actively prevents unauthorized users from doing so.

Digital Signature

A digital code that can be attached to an electronically transmitted message that two distinct goals: 1) Digitally signed messages assure the recipient that the message truly came from the claimed sender. They enforce non-repudiation (that is, they preclude the sender from later claiming that the message is a forgery) and 2) Digitally signed messages assure the recipient that the message was not altered while in transit between the sender and recipient. This protects against both malicious modification (a third party altering the meaning of the message) and unintentional modification (because of faults in the communications process, such as electrical interference).

Environments

Software is developed and maintained on several environments: the Development environment is used for implementing the piece of software, the User Acceptance Test environment is used to test and validate it with other stakeholders and the Production environment is used to exploit the system when it is "live", available as a service to its end users.

Error

An error is a severe validation failure, which will cause the message to be rejected.

eTIR IS

Acronym occasionally used in diagrams to refer to the eTIR international system.

eTIR stakeholder

An entity being part of the eTIR system and using the eTIR procedure as described in the Annex 11 of the TIR Convention. An eTIR stakeholder uses its information systems to be part of the eTIR system and can be any of the following entities:

- UNECE with the eTIR international system;
- IRU, representing the guarantee chain, with their information systems;
- Customs Authorities with their national customs systems;
- Holders with their information systems.

Hash

A hash value (or simply hash), also called a message digest, is a value generated from a text. The hash is substantially smaller than the text itself, and is generated by a formula in such a way that it is extremely unlikely that some other text will produce the same hash value.

Integrity

Integrity is the concept of protecting the reliability and correctness of data. Integrity protection prevents unauthorized alterations of data. It ensures that data remains correct, unaltered, and preserved. Properly implemented integrity protection provides a means for authorized changes while protecting against intended and malicious unauthorized activities (such as viruses and intrusions) as well as mistakes made by authorized users (such as mistakes or oversights).

ITDB

The International TIR DataBank is a UNECE application custodian of all TIR Carnet holder and customs office data. The data in this information system is maintained by the national associations and customs authorities of the TIR system.

Keystore

A keystore is a database of keys and is used to store certificates, including the certificates of all trusted parties, for use by a program. Through its keystore, an entity, can authenticate other parties, as well as authenticate itself to other parties.

Non-repudiation

Non-repudiation ensures that the subject of an activity or who caused an event cannot deny that the event occurred. Non-repudiation prevents a subject from claiming not to have sent a message, not to have performed an action, or not to have been the cause of an event. It is made possible through identification, authentication, authorization, accountability, and auditing. Non-repudiation can be established using digital certificates, session identifiers, transaction logs, and numerous other transactional and access control mechanisms.

OASIS

Organization for the Advancement of Structured Information Standards (OASIS) is a nonprofit, international consortium whose goal is to promote the adoption of product-independent standards.

PKI

Public-Key Infrastructure (PKI) is the infrastructure needed to support asymmetric cryptography.

Receiver

In the context of this document and of all the documents related to the message pairs, the "receiver" is the information system of the eTIR stakeholder which receives a message sent by another eTIR message and processes it.

RSA

The RSA algorithm was invented by Ronald L. Rivest, Adi Shamir, and Leonard Adleman in 1977. It is an asymmetric algorithm, that is to say it uses two different keys with a mathematic relationship to each other. The public key and private keys are carefully generated using the RSA algorithm; they can be used to encrypt information or sign it.

Sender

In the context of this document and of all the documents related to the message pairs, the "sender" is the information system of the eTIR stakeholder which prepares and sends an eTIR message to another eTIR stakeholder.

SOAP

Simple Object Access Protocol (SOAP) is a messaging protocol specification for exchanging information in the implementation of web services. It is an XML-based protocol consisting of three parts:

- an envelope, which defines the message structure (a haeder and a body) and how to process it;
- a set of encoding rules for expressing instances of application-defined datatypes;
- a convention for representing procedure calls and responses.

Token

A token (sometimes called a security token) is an object that controls access to a digital asset. Traditionally, this term has been used to describe a hardware authenticator, a small device used in a networked environment, to create a one-time password that the owner enters into a login screen along with an ID and a PIN. However, in the context of web services and with the emerging need for devices and processes to authenticate to each other over open networks, the term token has been expanded to include software mechanisms too. A token may be an X.509 certificate, that associates an identity to a public key for example.

Truststore

A truststore is a KeyStore file that contains certificates from other parties that you expect to communicate with, or from Certificate Authorities that you trust to identify other parties.

Web service

The definition of a web service is a communication over a network between two applications (not between a person and an application for example). Machine-to-machine is another term to define this type of communication.

Web Services Security (WS-Security)

The Web Services Security (WS-Security) specification describes enhancements to SOAP 1.1 that increase the protection (integrity) and confidentiality of the messages. These enhancements include functionality to secure SOAP messages through XML digital signature, confidentiality through XML encryption, and credential propagation through security tokens (e.g. X.509 token).

WSDL

Web Service Description Language (WSDL) is an XML-based interface description language that is used for describing the functionality offered by a web service.

X.509 digital certificate

In general use, a certificate is a document issued by some authority to attest to a truth or to offer certain evidence. A digital certificate is commonly used to offer evidence in electronic form about the holder of the certificate. In PKI it comes from a trusted third party, called a Certification Authority (CA) and it bears the digital signature of that authority.

X.509 token

The X.509 token is the X.509 certificate with the end user's credentials that can be passed in the SOAP message. The X.509 token can be used to authenticate the user based on the trust relationship (PKI). The X.509 token is also used to sign, encrypt and decrypt the SOAP message.

XML

XML stands for eXtensible Markup Language which is a language that defines a set of rules for encoding documents in a format that is both human-readable and machine-readable. It used by SOAP to encode messages sent by web services.

XML Signature

The XML Signature specification is a joint effort between W3C and IETF. XML Signatures provide integrity, message authentication and/or signer authentication services for data of any type, whether located within the XML that includes the signature or elsewhere.

XSD

XSD (XML Schema Definition) is a World Wide Web Consortium (W3C) recommendation that specifies how to formally describe the elements in an Extensible Markup Language (XML) document.

12.3. Technical summary

This annex lists all technical information in one page for ease of reference.

Key	Value	Comments
eTIR Specifications	4.3	
URL of the Production environment	https://etir-prod.unece.org/	
URL of the UAT environment n°1	https://etir-uat-01.unece.org/	
SOAP endpoint for Internal messages	{environment URL}/etir/v4.3/customs	
SOAP endpoint for External messages	{environment URL}/etir/v4.3/guaranteeChain	
Protocol used for HTTPS	TLS v1.2 and v1.3	It is recommended to ensure that your information system can use TLS v1.3 as TLS v1.2 will like ly to be decommissioned for security reasons in 2021.
Version of the X.509 certificate	3	These are the X.509 certificates used as electronic signatures to sign the messages sent between the various of the eTIR system
Size of the key for the generation of the X.509 certificate	2048 bits	
Signature algorithm for the generation of the X.509 certificate	SHA-256 with RSA	
Version of UUID to use	4	UUIDs are used in the **Message Identifier** and **Functional Reference** fields

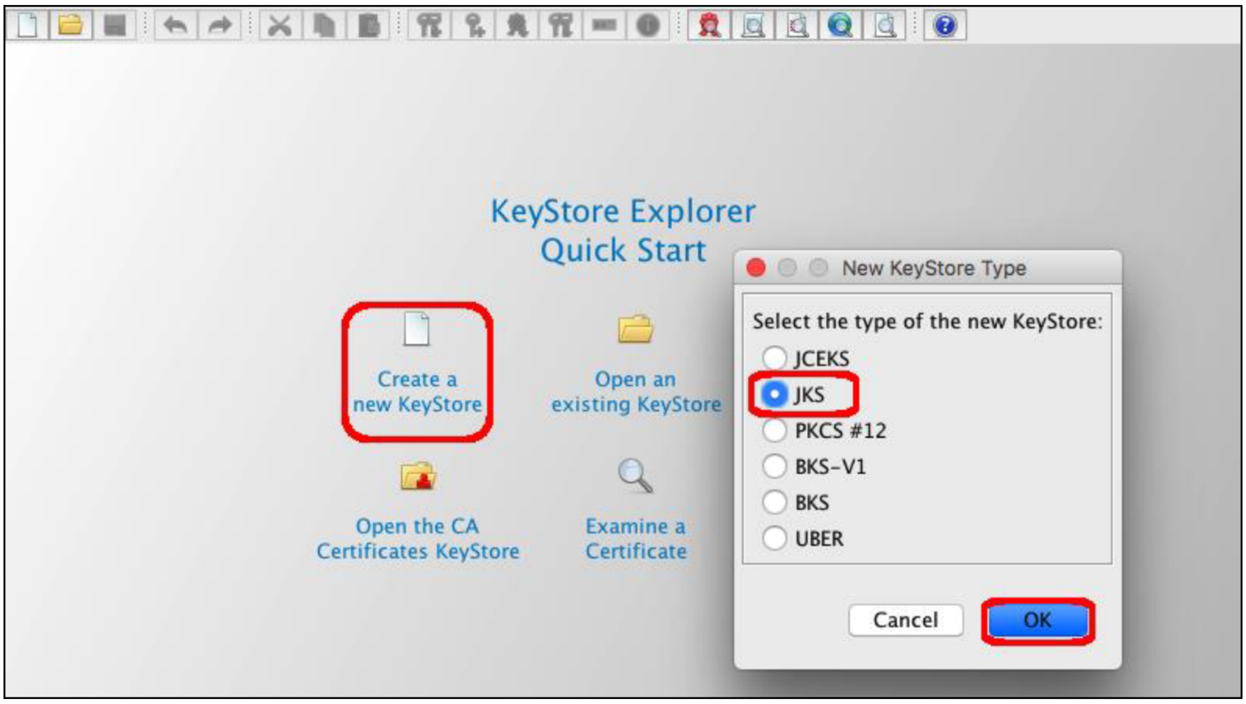
12.4. KeyStore: step by step generation of key pairs

12.4.1. Using the KeyStore Explorer application

This procedure describes keystore manipulations using the open-source GUI tool KeyStore Explorer. The latest version of the KeyStore Explorer is available for download on this page: http://www.keystore-explorer.org/

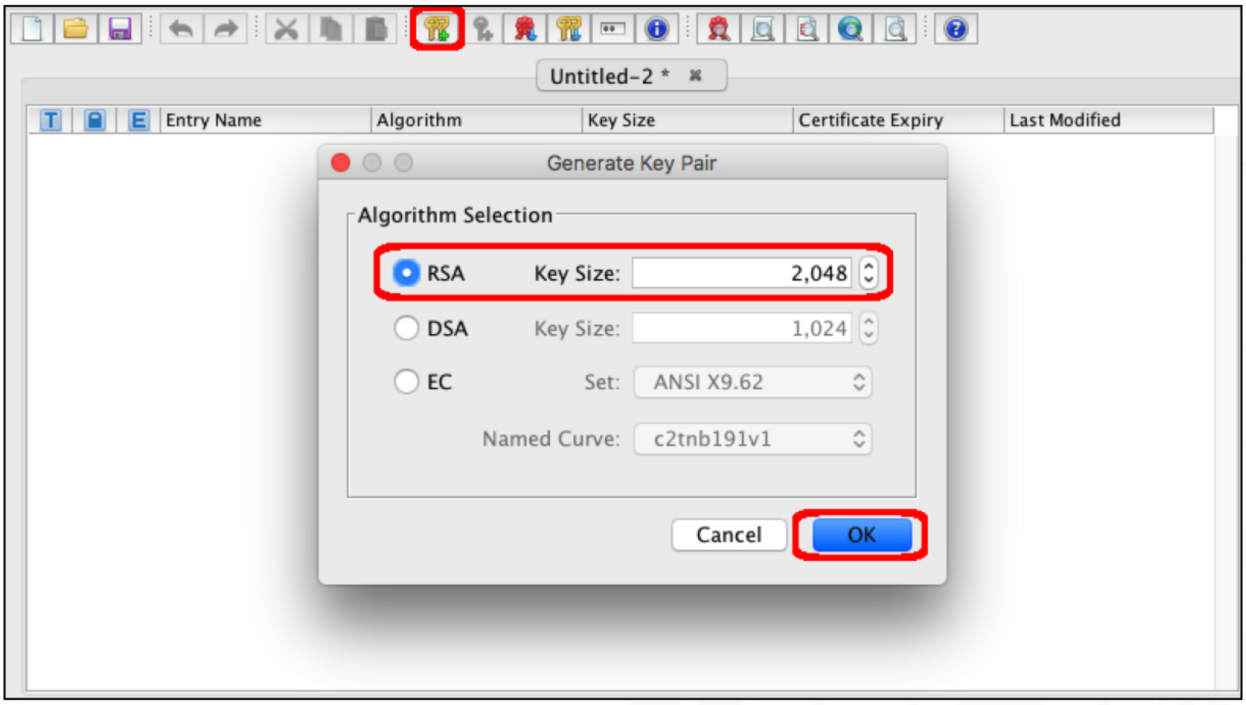
Key pair generation

1. Open KeyStore Explorer, click create a new KeyStore button and choose JKS as a type



Select the type of the new KeyStore:
JCEKS [✓ JKS] PKCS #12 BKS-V1 BKS UBER

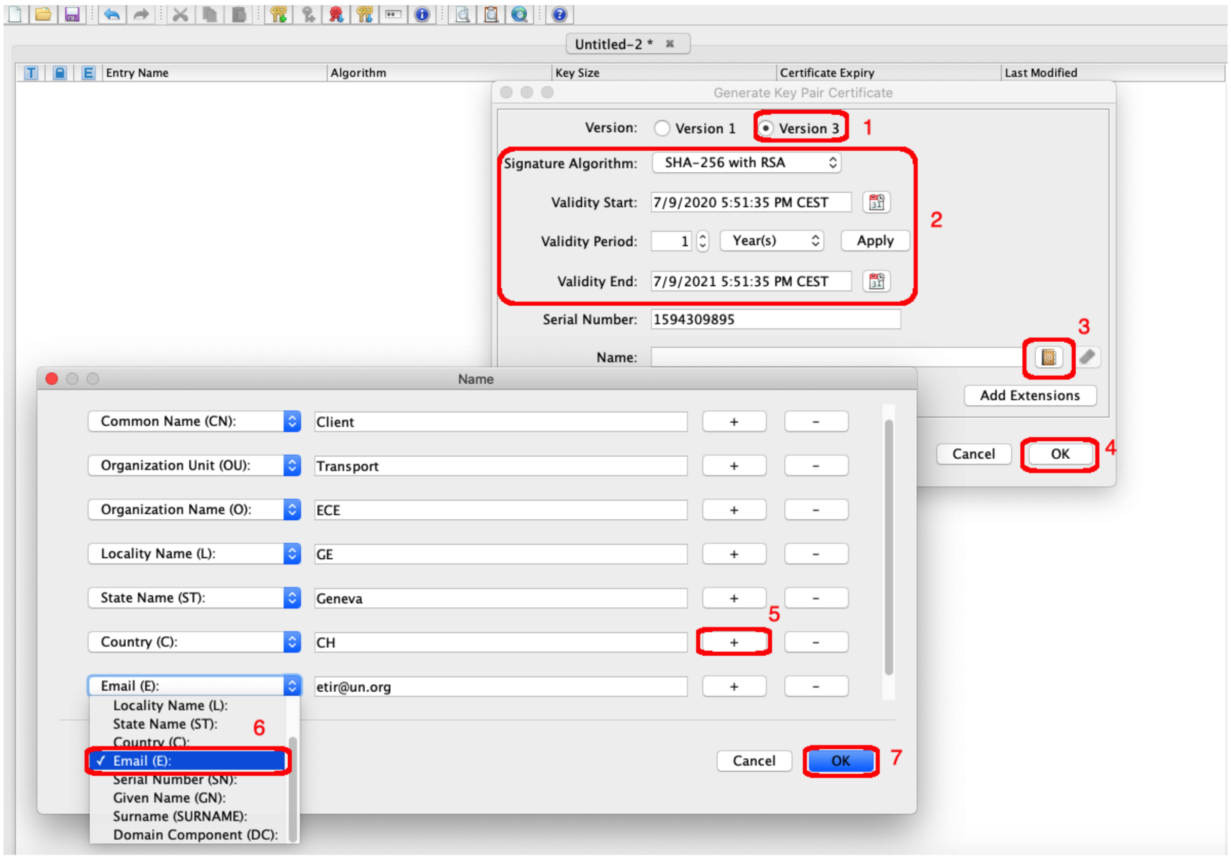
2. Generate Key Pair: choose RSA as algorithm with 2048 key size



Algorithm Selection
[✓ RSA] Key Size: 2,048
DSA Key Size: 1,024
EC Set: ANSI X9.62
Named Curve: c2tnb191v1

3. Generate Key Pair Certificate: choose SHA-256 with RSA (version 3) as algorithm with the desired validity period and follow the steps from one to four.

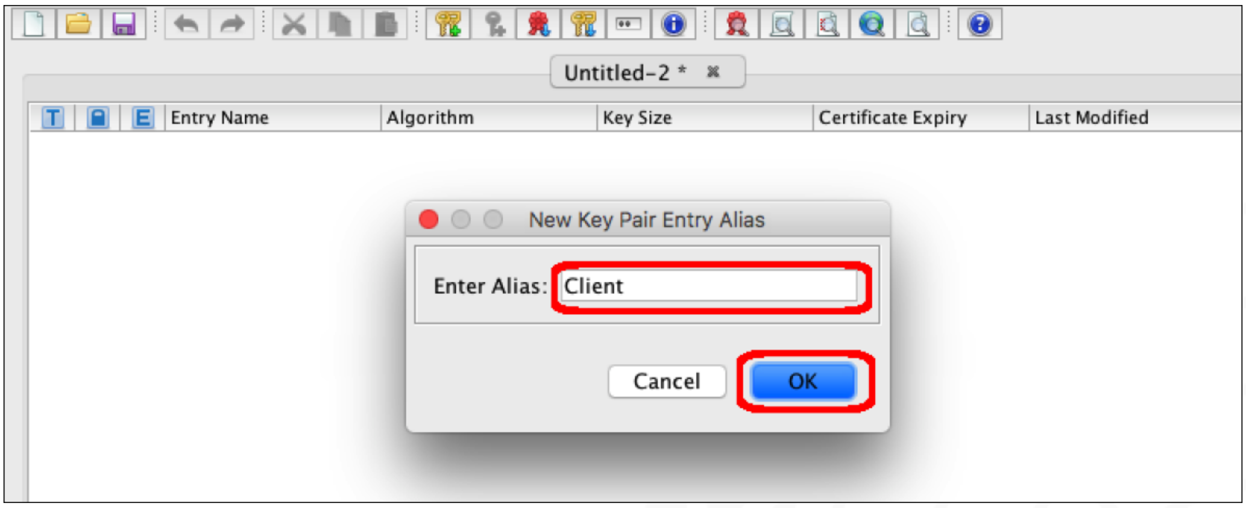
Make sure to type in the actual information related to your Customs office, not the sample values displayed in the screenshot below.



Version: [✓ Version 3]
Signature Algorithm: SHA-256 with RSA
Validity Start: 7/9/2020 5:51:35 PM CEST
Validity Period: 1 Year(s) Apply
Validity End: 7/9/2021 5:51:35 PM CEST
Serial Number: 1594309895
Name:

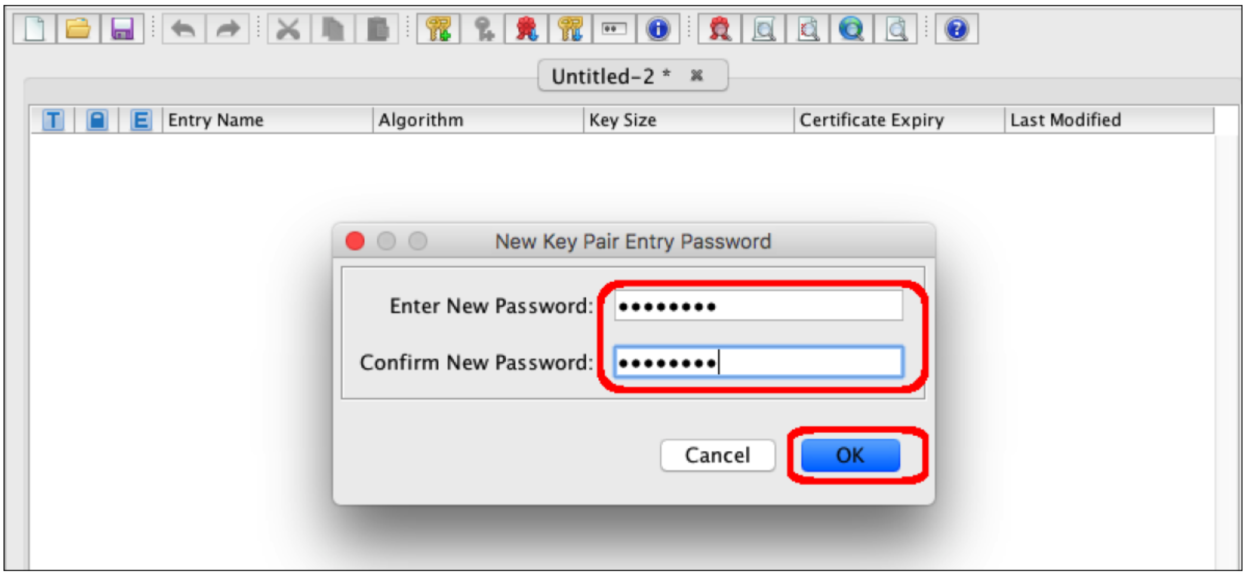
Common Name (CN):	Client
Organization Unit (OU):	Transport
Organization Name (O):	ECE
Locality Name (L):	GE
State Name (ST):	Geneva
Country (C):	CH
Email (E):	etir@un.org

4. A popup window appears to define an alias for the key pair



Enter Alias: Client

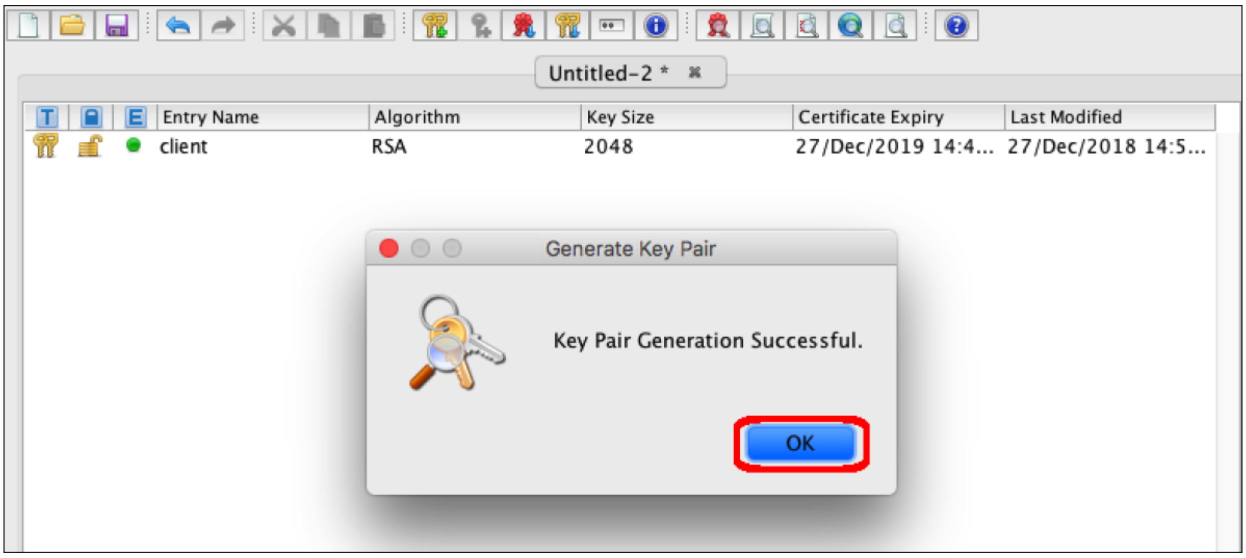
5. Another popup appears to define a key pair password.



Enter New Password: ••••••••

Confirm New Password: ••••••••

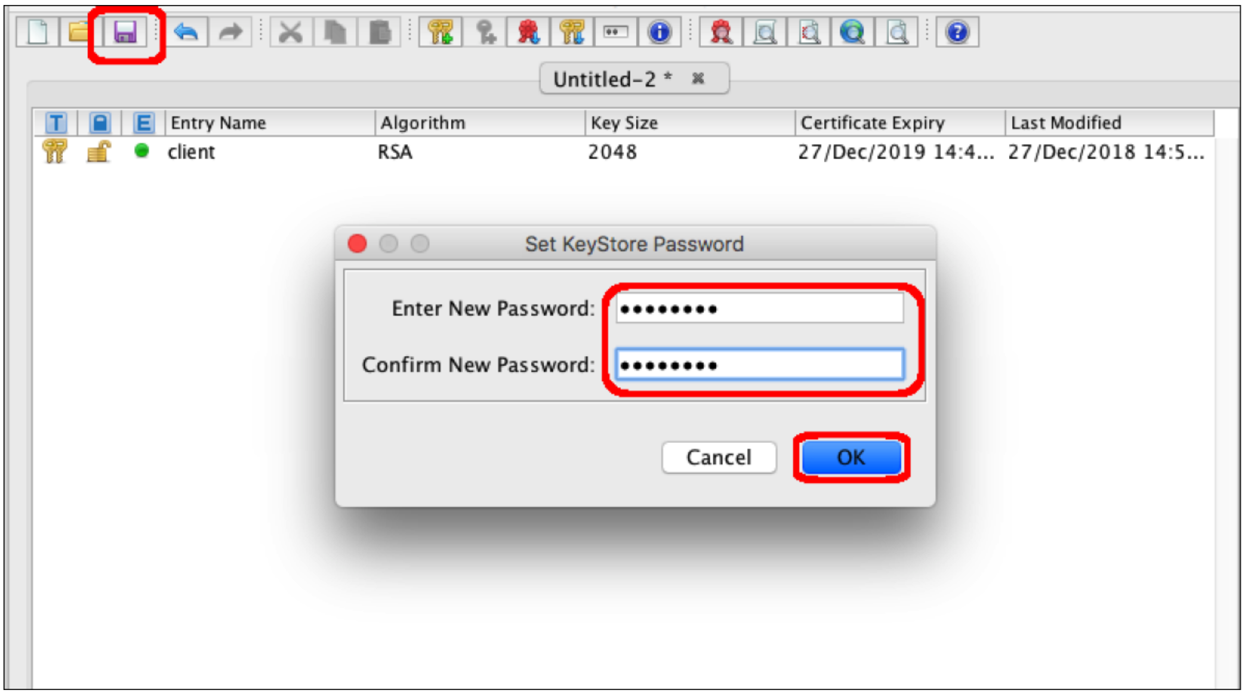
6. Key pair generation confirmed.



Entry Name	Algorithm	Key Size	Certificate Expiry	Last Modified
client	RSA	2048	27/Dec/2019 14:4...	27/Dec/2018 14:5...

Key Pair Generation Successful.

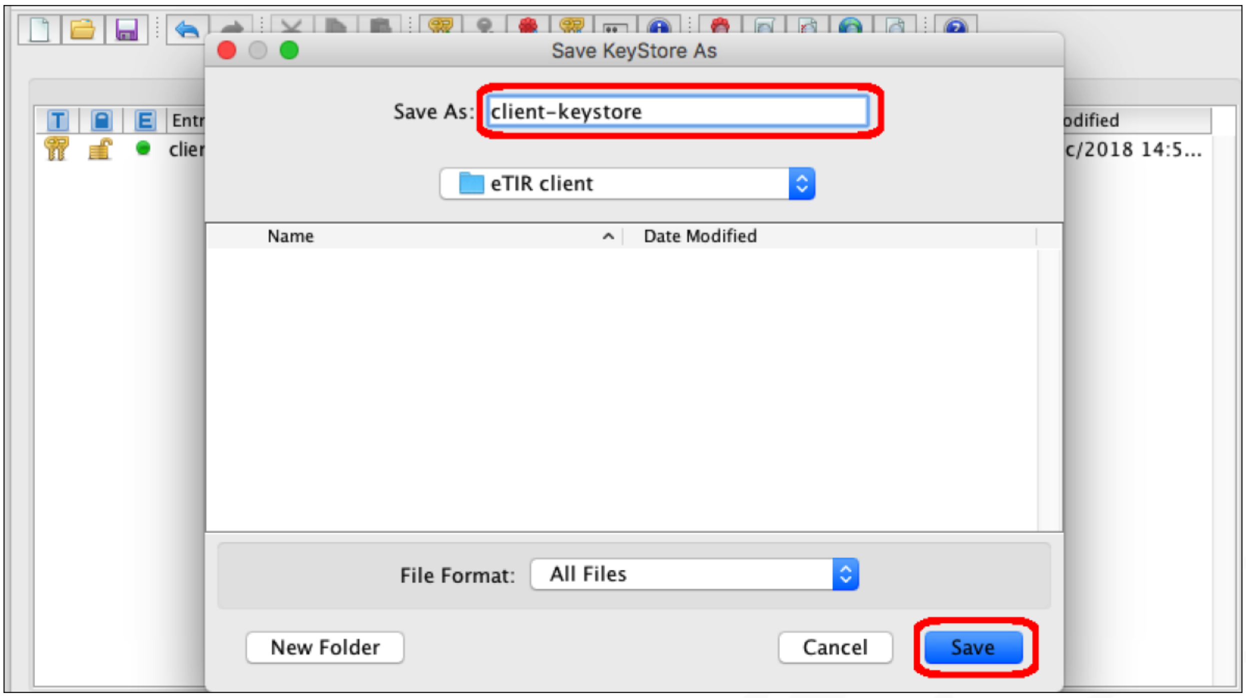
7. Once the key pair generated, click on the save button, a popup window appears to choose the keystore password (it is advised to use another password than for the key pair).



Entry Name	Algorithm	Key Size	Certificate Expiry	Last Modified
client	RSA	2048	27/Dec/2019 14:4...	27/Dec/2018 14:5...

Enter New Password: ••••••••
Confirm New Password: ••••••••

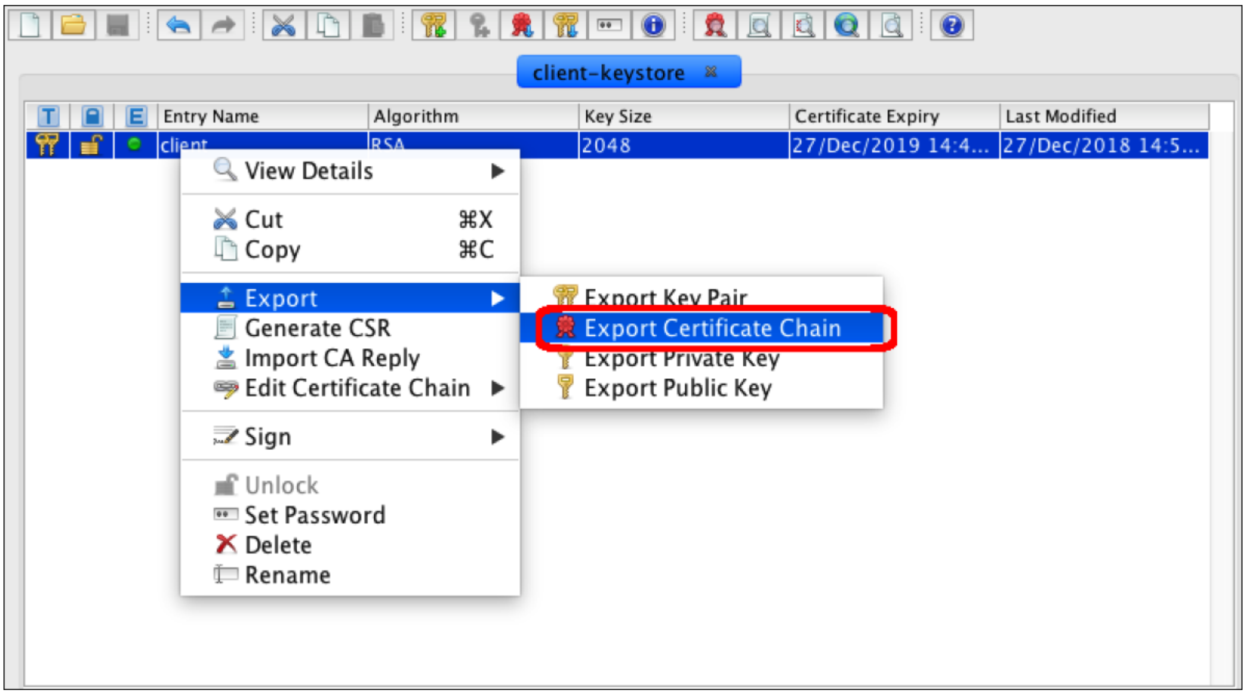
8. Specify the keystore name and path (the extension of the file should be ".jks") and then click on the save button.



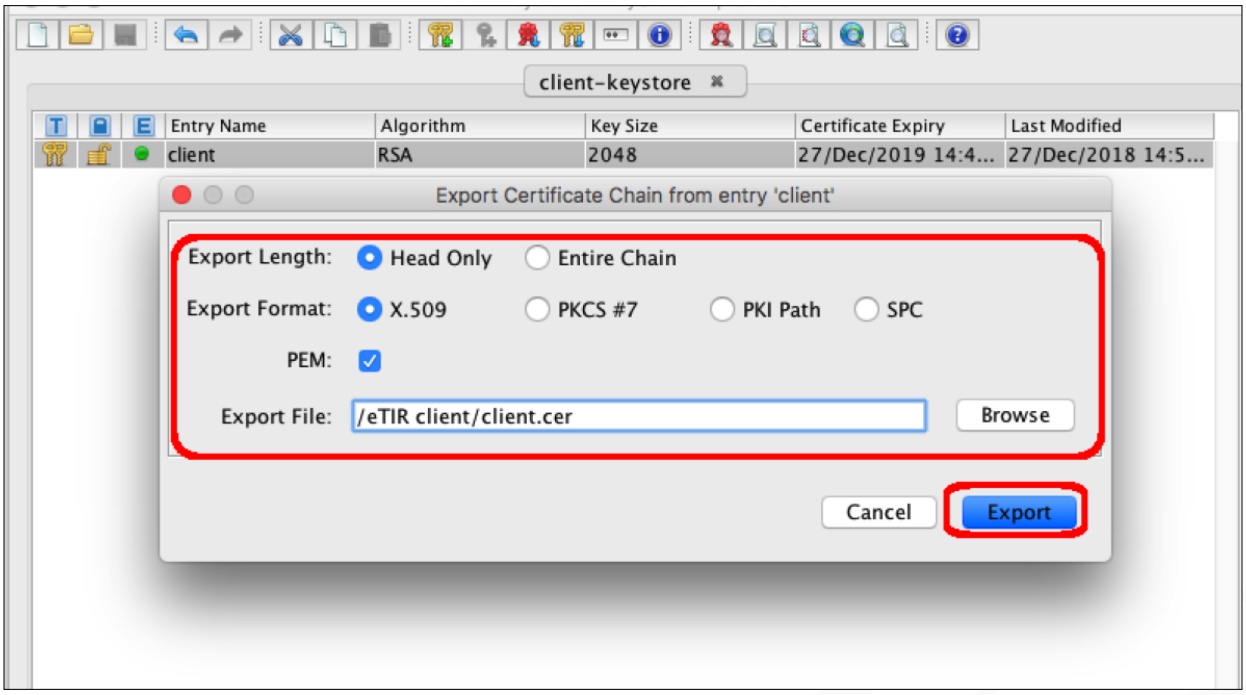
Save As: client-keystore
eTIR client
File Format: All Files

Exporting client's certificate

1. Open the generated keystore from the KeyStore Explorer, right click on the generated key pair, select Export then Export Certificate Chain.



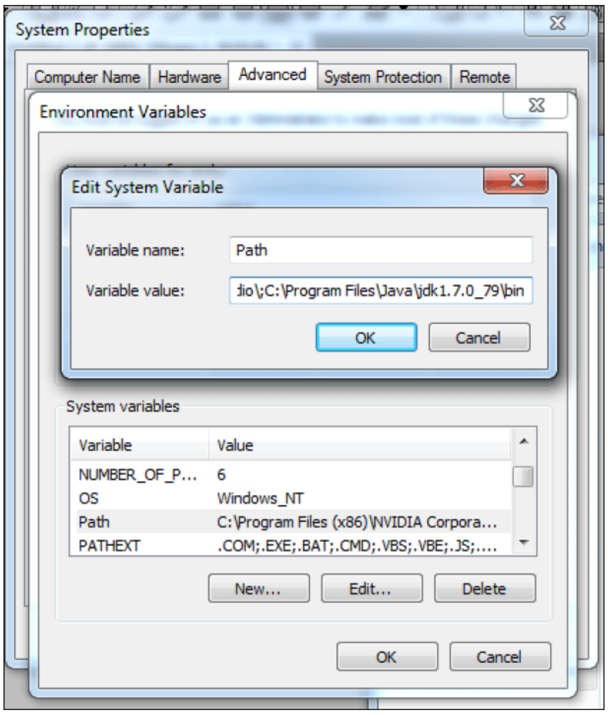
2. Keep the default values, choose the path of the exported certificate and hit export.



Export Length: [✓ Head Only] Entire Chain
Export Format: [✓ X.509] PKCS #7 PKI Path SPC
PEM: ✓
Export File: /eTIR client/client.cer

12.4.2. Using the Keytool command line interface

The Java Keytool is a command line tool to generate public/private key pairs and store them in a Java KeyStore. Before starting to use the keytool command line, the java bin directory should be added to the path environment variable as shown below:



Variable name: Path

Variable value: dio\;C:\Program Files\Java\jdk1.7.0_79\bin

Variable	Value
NUMBER_OF_P...	6
OS	Windows_NT
Path	C:\Program Files (x86)\NVIDIA Corpora...
PATHEXT	.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;....

The Keytool executable is called keytool. To execute it, open a command line interface (cmd, console, shell etc.). and change directory into the bin directory of your Java SDK installation. Enter keytool and you should see something similar to this:

```
C:\Program Files\Java\jdk1.8.0_111\bin>keytool
Key and Certificate Management Tool

Commands:
 -certreq                Generates a certificate request
 -changealias            Changes an entry''s alias
 -delete                 Deletes an entry
 -exportcert             Exports certificate
 -genkeypair             Generates a key pair
 -genseckey              Generates a secret key
 -gencert                Generates certificate from a certificate request
 -importcert             Imports a certificate or a certificate chain
 -importpass             Imports a password
 -importkeystore         Imports one or all entries from another keystore
 -keypasswd              Changes the key password of an entry
 -list                   Lists entries in a keystore
 -printcert              Prints the content of a certificate -printcertreq Prints the content of a certificate
request
 -printcrl               Prints the content of a CRL file
 -storepasswd            Changes the store password of a keystore

Use "keytool -command_name -help" for usage of command_name
```

The Keytool command can take a lot of arguments which may be hard to remember how to set them correctly. Therefore, it is a good idea to create some scripts to simplify the execution of the keytool commands later which also simplifies the maintenance aspects.

Key pair generation

Generating a public/private key pair is one of the most common tasks when using the Keytool. The generated key pair is recorded into a Java KeyStore file as a self-signed key pair. Here is the general command line format for generating a key pair with Keytool:

```
-genkeypair {-alias alias} {-keyalg keyalg} {-keysize keysize} {-sigalg sigalg} [-dname dname] [-keypass keypass] {-
validity valDays} {-storetype storetype} {-keystore keystore} [-storepass storepass] {-providerClass
provider_class_name {-providerArg provider_arg}} {-v} {-protected} {-Jjavaoption}
```

The keystore can be generated using the following set of instructions:

```
keytool -genkeypair -alias client -keystore ClientKeyStore.jks -keyalg RSA \
-dname "CN=Client, OU= Transport, O=ECE, L=GE, ST=GENEVA, C=CH, EMAILADDRESS=etir@un.org" \
-sigalg SHA256withRSA -validity 1000

Enter keystore password: keyStorePassword
Re-enter new password: keyStorePassword

Enter key password for <client>
        (RETURN if same as keystore password): certificatePassword
Re-enter new password: certificatePassword
```

Exporting client's certificate

The Keytool command line can also export certificates stored in a KeyStore. Find below the Keytool command templates for exporting certificates:

```
-exportcert {-alias alias} {-file cert_file} {-storetype storetype} {-keystore keystore} [-storepass storepass] {-
providerName provider_name} {-providerClass provider_class_name {-providerArg provider_arg}} {-rfc} {-v} {-
protected} {-Jjavaoption}
```

In the same folder as the generated Keystore, extract the public key certificate using the following command line and send it to UNECE email addresses (to be stored in the application server's truststore):

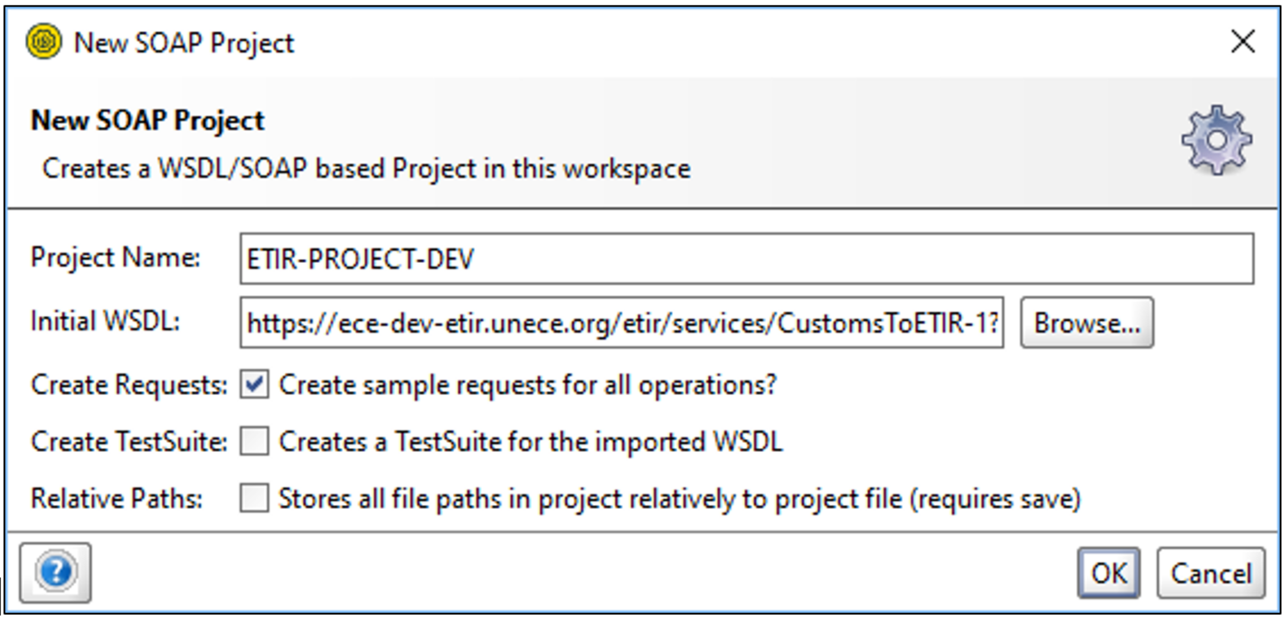
```
keytool -exportcert -keystore ClientKeyStore.jks -alias client -file client.cer -storepass certificatePassword
```

Hit enter then the certificate will be stored in the same folder as the keystore:

```
Certificate stored in file <client.cer>
```

12.5. SOAP UI quick guide

Please find bellow the basic steps required to install and configure SOAP UI testing tool, and for sending an encrypting a SOAP request via SoapUI. Download and install the SoapUI Open Source edition application from SOAP UI website download web page. Start SoapUI application and create a new SOAP project by clicking on File and New SOAP Project. Then, enter a name and set the test URL for the WSDL: https://etir-uat-01.unece.org/etir/v4.3?wsdl


New SOAP Project

New SOAP Project

Creates a WSDL/SOAP based Project in this workspace

Project Name: ETIR-PROJECT-DEV

Initial WSDL: https://ece-dev-etir.unece.org/etir/services/CustomsToETIR-1? Browse...

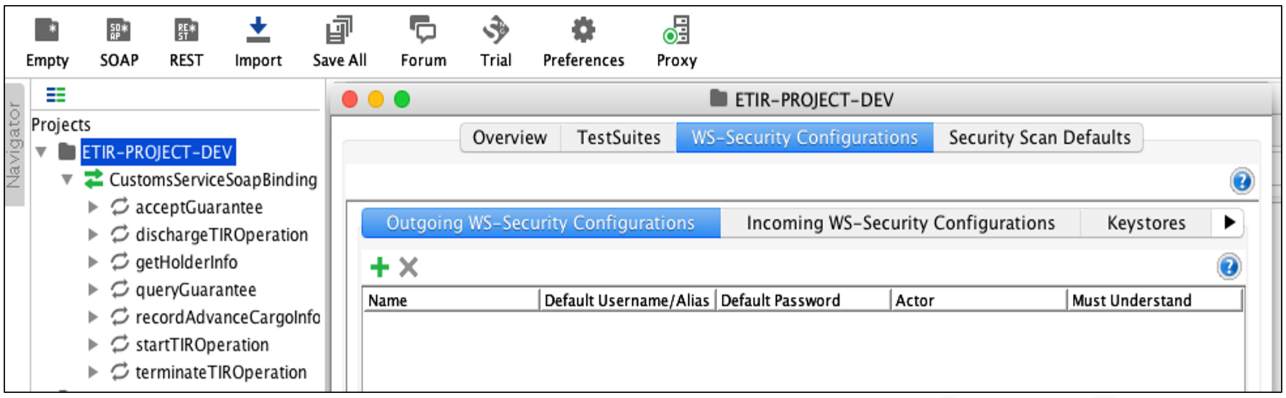
Create Requests: ☑ Create sample requests for all operations?

Create TestSuite: ☐ Creates a TestSuite for the imported WSDL

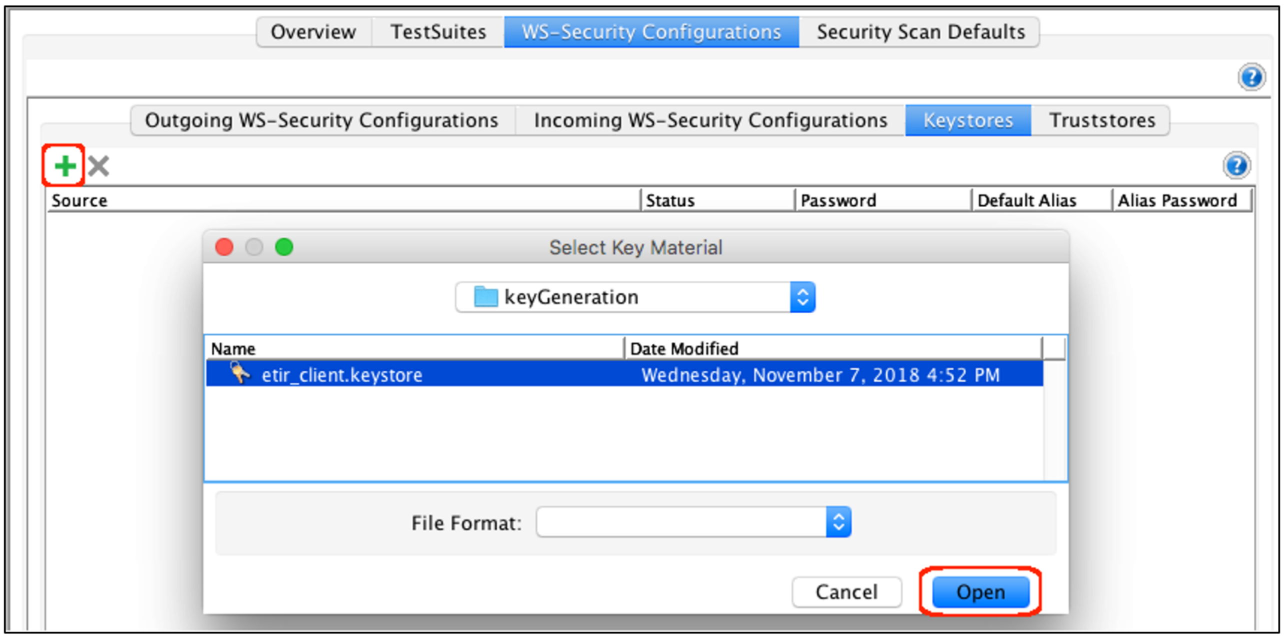
Relative Paths: ☐ Stores all file paths in project relatively to project file (requires save)

OK Cancel

3. After creating a test project, go to the WS-Security Configurations tab inside the 'Show Project View' Menu by right clicking the project folder.

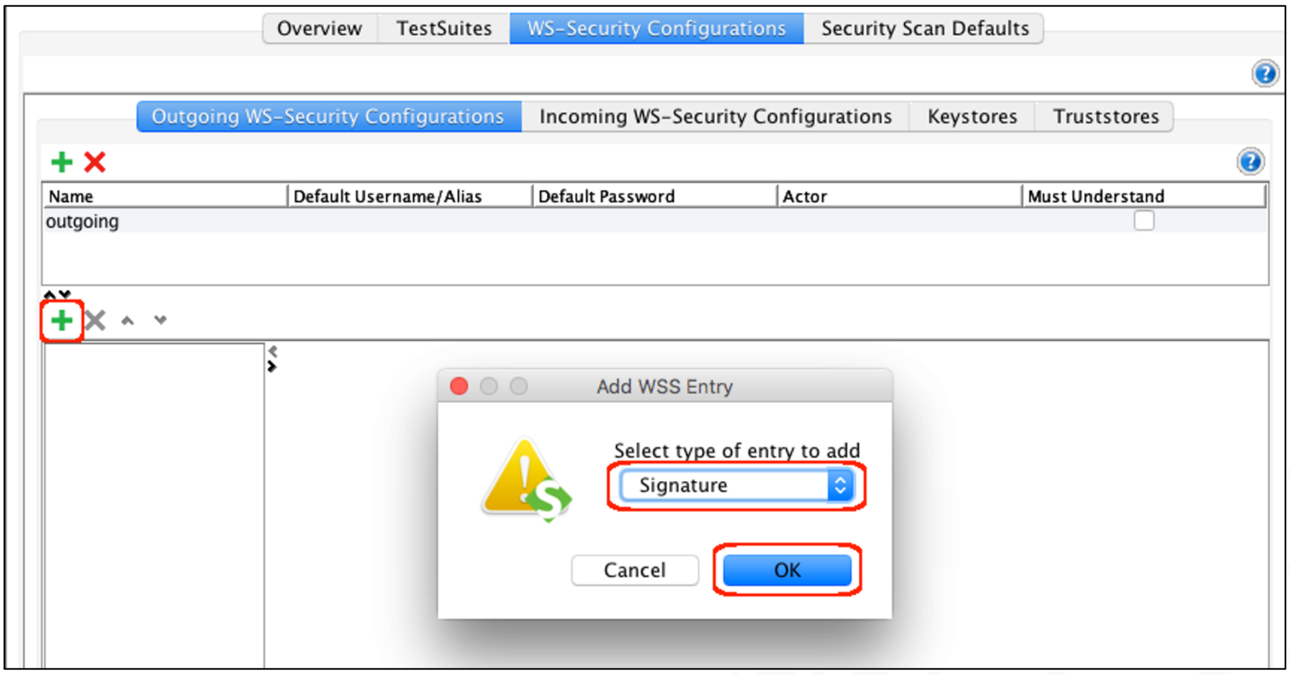


4. Add a KeyStore by clicking the add button and browsing to your keystore file, enter the password and click ok.

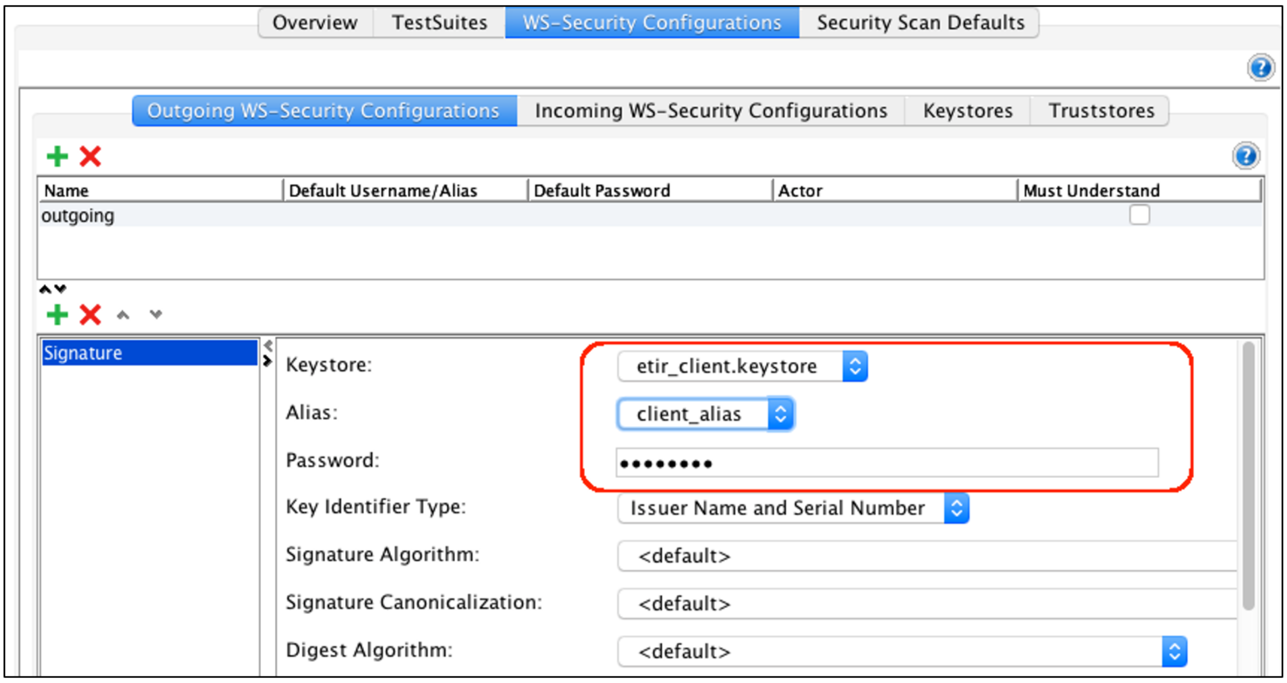


Then click on the **Outgoing WS-Security Configurations** tab, to display the configurations that should be applied to outgoing messages, including requests. This configuration type is used for encryption, signing and adding SAML, timestamp and username headers. In this situation, only the signing part is used.

5. Add a signature entry



6. Select the keystore and key alias (key) that will be used along with the password for that alias. These are the configurable fields:
    - **Keystore**: the keystore to use when signing the message.
    - **Alias**: the alias (key pair) to use when signing the message.
    - **Password**: the password of the alias.

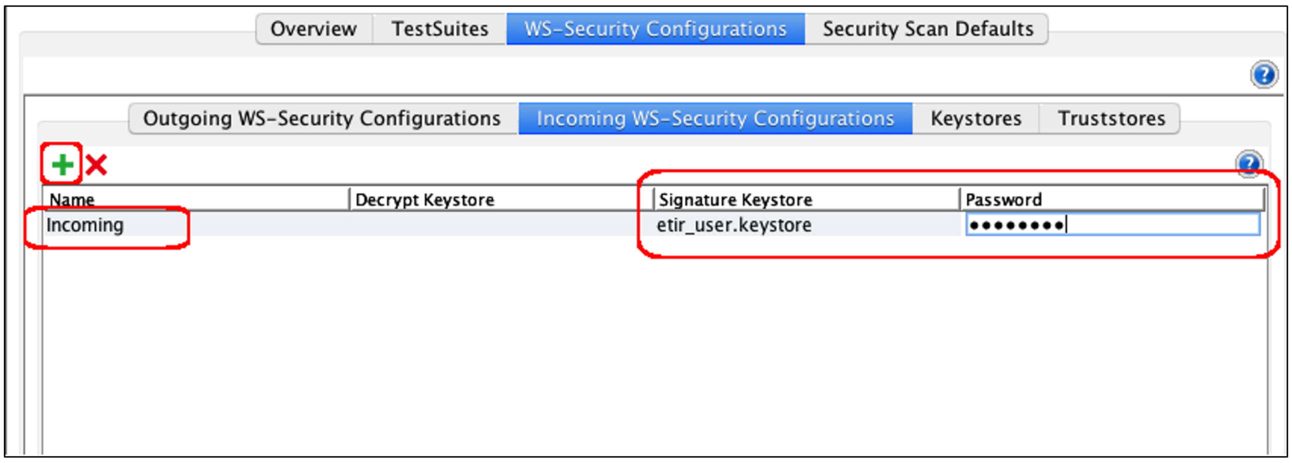

Overview
TestSuites
WS-Security Configurations
Security Scan Defaults
Outgoing WS-Security Configurations
Incoming WS-Security Configurations
Keystores
Truststores

Name	Default Username/Alias	Default Password	Actor	Must Understand
outgoing				☐

Signature

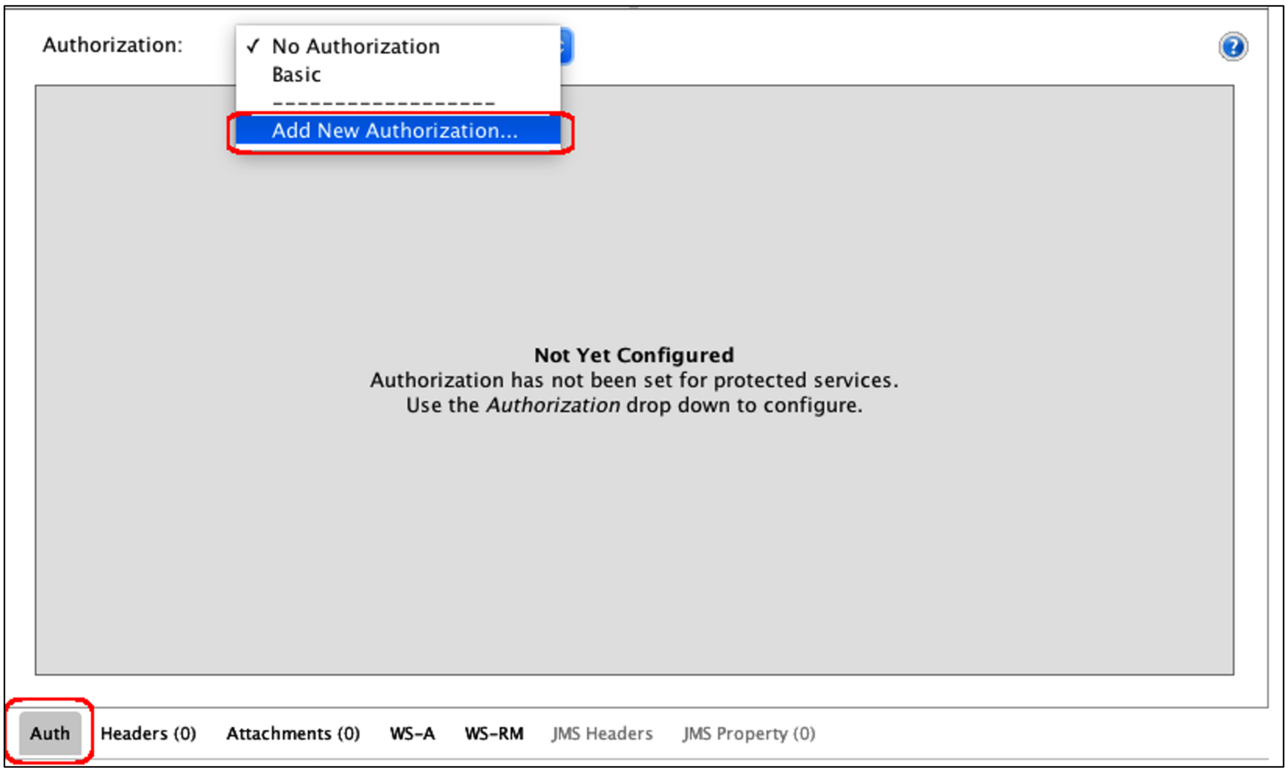
Keystore: etir_client.keystore
Alias: client_alias
Password: ••••••••
Key Identifier Type: Issuer Name and Serial Number
Signature Algorithm: <default>
Signature Canonicalization: <default>
Digest Algorithm: <default>

7. Click on the **Incoming WS-Security Configurations** tab, to configure what should be applied to incoming messages, including responses. Specify the keystore name and the password. This configuration will ensure the validation of the signature of the incoming responses from eTIR server.


Overview
TestSuites
WS-Security Configurations
Security Scan Defaults
Outgoing WS-Security Configurations
Incoming WS-Security Configurations
Keystores
Truststores

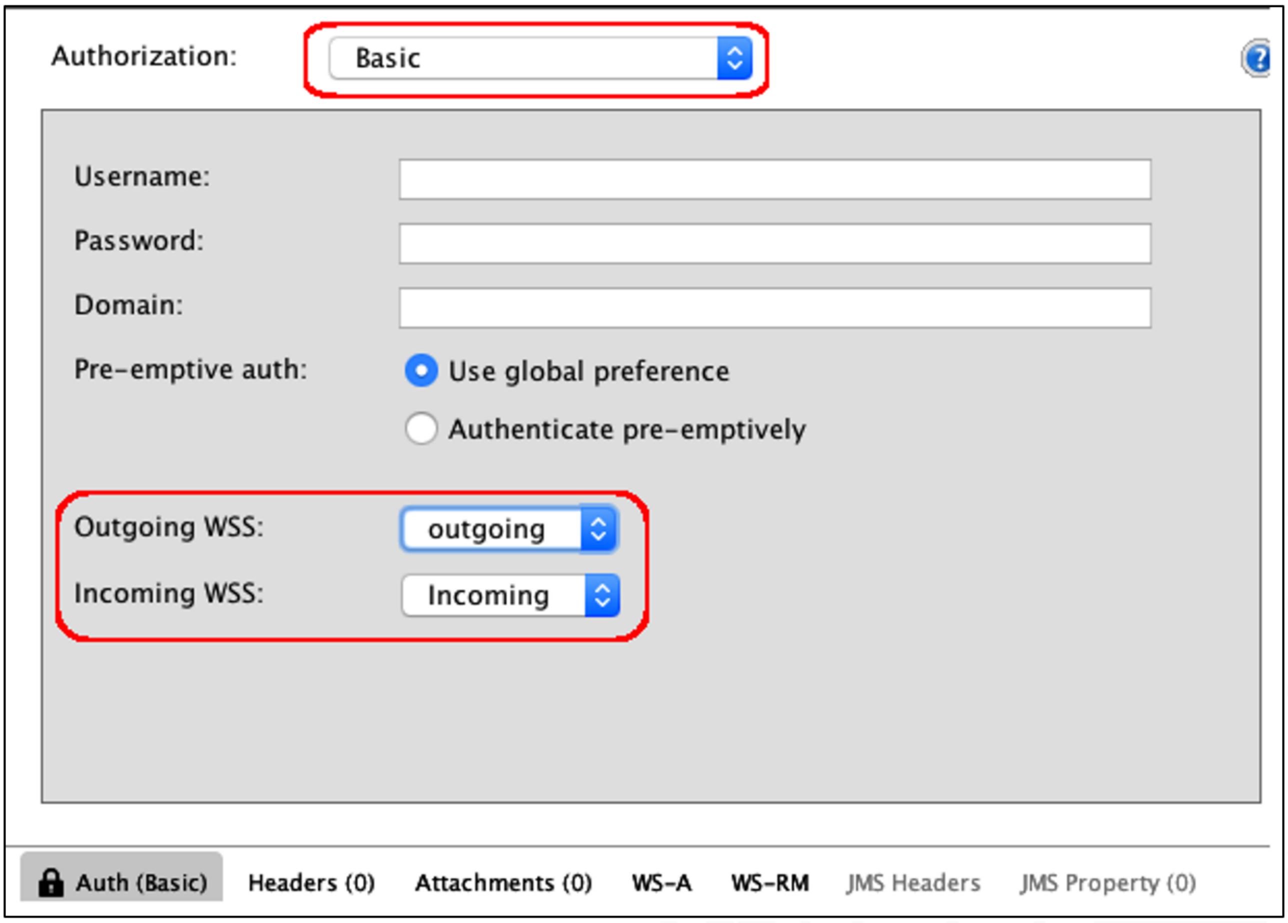
Name	Decrypt Keystore	Signature Keystore	Password
Incoming		etir_user.keystore	••••••••

8. Open the SOAP request by right clicking on the **queryGuarantee** request in the treeview on the left-hand side of the application and then select **New request** and give it a name.
9. Click the **Auth** tab in the bottom left corner and in the Authorization drop down list, select **Add New Authorization**… and then select **Basic** for a basic authorization.


Authorization: ✓ No Authorization / Basic / Add New Authorization...

Not Yet Configured
Authorization has not been set for protected services.
Use the *Authorization* drop down to configure.

10. New configuration window appears, select the preconfigured outgoing and incoming WSS.

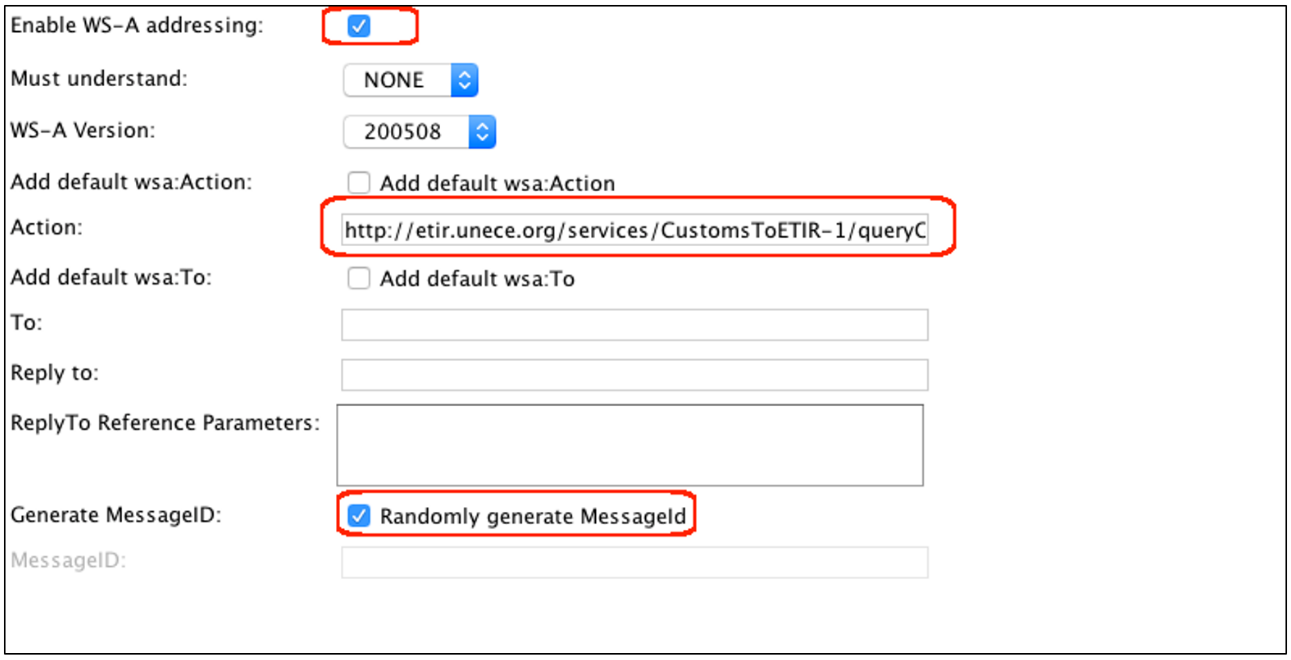

Authorization: Basic

Username:
Password:
Domain:
Pre-emptive auth: [✓ Use global preference] Authenticate pre-emptively
Outgoing WSS: outgoing
Incoming WSS: Incoming

11. Switch to the WS-A tab. Make sure that the following settings are properly configured:
    a. Enable WS-A addressing

b. Enable 'Randomly generated MessageId;

c. Make sure that the action field is correct and looks like the following: https://etir-uat-01.unece.org/etir/v4.3



Enable WS-A addressing:	✓
Must understand:	NONE
WS-A Version:	200508
Add default wsa:Action:	☐ Add default wsa:Action
Action:	http://etir.unece.org/services/CustomsToETIR-1/queryC
Add default wsa:To:	☐ Add default wsa:To
To:	
Reply to:	
ReplyTo Reference Parameters:	
Generate MessageID:	✓ Randomly generate MessageId
MessageID:	

12. Run SoapUI against the Web Service by passing Soap Messages to the server. (Example below):

```
<soap:Envelope xmlns:cus="{link-etir-uat-url}"
xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
xmlns:urn="urn:wco:datamodel:WCO:I5:1">
<soap:Header/>
   <soap:Body>
      <cus:queryGuarantee>
         <urn:InterGov>
            <urn:FunctionCode>9</urn:FunctionCode>
            <urn:ID>uuid-query</urn:ID>
            <urn:TypeCode>I5</urn:TypeCode>
            <urn:ReplyTypeCode>00</urn:ReplyTypeCode>
            <urn:ObligationGuarantee>
               <urn:ReferenceID>XC95000003</urn:ReferenceID>
            </urn:ObligationGuarantee>
         </urn:InterGov>
      </cus:queryGuarantee>
   </soap:Body>
</soap:Envelope>
```

12.6. Example of SOAP request using WSS4J

The example describes how to use the apache Web Services Security for Java WSS4J to create the SOAP client. The WSS4J project provides a Java implementation of the primary security standards for Web Services, namely the OASIS Web Services Security (WS-Security) specifications from the OASIS Web Services Security TC. WSS4J provides an implementation of X.509 token profile. In this example the signature of the received responses from the eTIR international system can be checked and validated through interceptors, this part is explained in the next section.

First, you need to create your certificate. You can follow on of the two procedures detailed in the section KeyStore: step by step generation of key pairs: using KeyStore Explorer application or the Keytool command line interface.

Create the cryptographic property file

WSS4J requires some cryptographic properties. These properties are grouped in three sections as shown below:

```
#Crypto properties
#General properties:
#WSS4J specific provider used to create Crypto instances. Defaults to
"org.apache.ws.security.components.crypto.Merlin".
#org.apache.ws.security.crypto.provider=
#Keystore properties:
#The location of the keystore
org.apache.ws.security.crypto.merlin.keystore.file=keystoreFile
#The password used to load the keystore. Default value is "security".
org.apache.ws.security.crypto.merlin.keystore.password=password
#Type of keystore. Defaults to: java.security.KeyStore.getDefaultType()), normally it is JKS
#org.apache.ws.security.crypto.merlin.keystore.type=JKS
#The default keystore alias to use, if none is specified.
org.apache.ws.security.crypto.merlin.keystore.alias=aliasName
```

Password callback handler

The keystore password is specified in the cryptographic properties file as shown in the previous section. However the private key password was not provided yet. In the cryptographic property file there is a property to specify this i.e.

```
#The default password used to load the private key. Normally it is provided through a password-callback class
#org.apache.ws.security.crypto.merlin.keystore.private.password=
```

A good practice is to provide it through a password callback handler to ensure higher security. The password callback handler can fetch the password from a database, LDAP or from more secured places. In this example, we just display the private key password from the password callback handler class. Find below a basic implementation of the password callback handler:

```
import java.io.IOException;
import javax.security.auth.callback.Callback;
import javax.security.auth.callback.CallbackHandler;
import javax.security.auth.callback.UnsupportedCallbackException;
import org.apache.ws.security.WSPasswordCallback;

public class ServerPasswordCallback implements CallbackHandler {

  public void handle(Callback[] callbacks) throws IOException, UnsupportedCallbackException {

     for (int i = 0; i < callbacks.length; i++) {
      WSPasswordCallback pc = (WSPasswordCallback) callbacks[i];
      pc.setPassword("key-pass");
    }
  }
}
```

WSS4J Interceptors

The final step is to write the inbound and outbound WSS4J interceptors on the client side. Two interceptors are needed for the SOAP request. The interceptor beans configure the rest of the security settings like, whether the SOAP message will contain timestamp, signature, what parts need to be signed, what algorithm to use, what type of BinarySecurityToken and reference would be used, the pointer to Password Callback Handler class etc. All these settings are configured as key/value pairs in a map. The whole map is listed as WSHandler Tags on the WSS4J configuration page. Please note that many of the keys/settings have default values.

Inbound and outbound WSS4J interceptors - XML-Spring configuration

```
<beans xmlns="http://www.springframework.org/schema/beans"
 xmlns:jaxws="http://cxf.apache.org/jaxws" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="
 http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd
 http://cxf.apache.org/jaxws http://cxf.apache.org/schemas/jaxws.xsd">

 <bean id="logInBound" class="org.apache.cxf.interceptor.LoggingInInterceptor" />
 <bean id="logOutBound" class="org.apache.cxf.interceptor.LoggingOutInterceptor" />
 <jaxws:client id="customsClient" serviceClass="org.unece.etir.ws.cusc.CustomsSEISecure"
 address="https://ece-dev-etir.unece.org/etir/services/CustomsToETIR-1">
 <jaxws:inInterceptors>
 <ref bean="logInBound" />
 <ref bean="inbound-security" />
 </jaxws:inInterceptors>
 <jaxws:outInterceptors>
 <ref bean="logOutBound" />
 <ref bean="outbound-security" />
 </jaxws:outInterceptors>
 </jaxws:client>

 <!-- WSS4JOutInterceptor for signing outbound SOAP messages -->
    <bean class="org.apache.cxf.ws.security.wss4j.WSS4JOutInterceptor" id="outbound-security">
        <constructor-arg>
            <map>
                <entry key="action" value="Signature"/>
                <entry key="user" value="client"/>
                <entry key="signaturePropFile" value="client-crypto.properties"/>
                <entry key="passwordCallbackClass" value="client.ClientPasswordCallback"/>
                <entry key="signatureParts"
                value="{Element}{http://schemas.xmlsoap.org/soap/envelope/}Body"/>
            </map>
        </constructor-arg>
    </bean>

    <!--  WSS4JInInterceptor for validating the signature of inbound SOAP messages -->
    <bean class="org.apache.cxf.ws.security.wss4j.WSS4JInInterceptor" id="inbound-security">
        <constructor-arg>
            <map>
                <entry key="action" value="Signature"/>
                <entry key="signaturePropFile" value="client-crypto.properties"/>
                <entry key="passwordCallbackClass" value="client.ClientPasswordCallback"/>
            </map>
        </constructor-arg>
    </bean>

</beans>
```

Inbound and outbound WSS4J interceptors - Java configuration

WSS4JOutInterceptor signing outbound SOAP messages:

```
Map<String, Object> propsOut = new HashMap<String, Object>();
propsOut.put(WSHandlerConstants.ACTION, WSHandlerConstants.SIGNATURE);
propsOut.put(WSHandlerConstants.SIGNATURE_PARTS, "{Element}{http://www.w3.org/2003/05/soap-envelope}Body;");
propsOut.put(WSHandlerConstants.PW_CALLBACK_CLASS, PasswordCallback.class.getName());
propsOut.put(WSHandlerConstants.USER, "client");
propsOut.put(WSHandlerConstants.SIG_PROP_FILE, "META-INF/client-security.properties");
```

WSS4JInInterceptor validating the signature of inbound SOAP messages:

```java
Map<String, Object> propsIn = new HashMap<String, Object>();
propsIn.put(WSHandlerConstants.ACTION, WSHandlerConstants.SIGNATURE);
propsIn.put(WSHandlerConstants.PW_CALLBACK_CLASS, PasswordCallback.class.getName());
propsIn.put(WSHandlerConstants.SIG_PROP_FILE, "META-INF/client-security.properties");

WSS4JOutInterceptor wssOut = new WSS4JOutInterceptor(propsOut);
WSS4JInInterceptor wssIn = new WSS4JInInterceptor(propsIn);
Client client = ClientProxy.getClient(port);
Endpoint endpoint = client.getEndpoint();
endpoint.getOutInterceptors().add(wssOut);
endpoint.getInInterceptors().add(wssIn);
endpoint.getInInterceptors().add(new LoggingInInterceptor());
endpoint.getOutInterceptors().add(new LoggingOutInterceptor());
```