United Nations ECE/TRANS/WP.30/GE.1/2021/5

Economic and Social Council

Distr.: General
3 December 2020

Original: English

Economic Commission for Europe

Inland Transport Committee

Working Party on Customs Questions affecting Transport

Group of Experts on Conceptual and Technical Aspects of Computerization of the TIR Procedure

First session
Geneva, 27–29 January 2021
Item 6 (a) of the provisional agenda
eTIR international system:
Progress report on the development of the eTIR international system

eTIR web services – Glossary and technical procedures*

Note by the secretariat

I. Introduction - Mandate

1. The Inland Transport Committee during its eighty-second session (23–28 February 2020) approved (ECE/TRANS/294, para. 84[1]) the establishment of the Group of Experts on Conceptual and Technical Aspects of Computerization of the TIR Procedure (WP.30/GE.1) and endorsed its ToR[2] (ECE/TRANS/WP30/2019/9 and ECE/TRANS/WP.30/2019/9/Corr.1) pending approval by UNECE Executive Committee (EXCOM). EXCOM during its Remote informal meeting of members of the Executive Committee (20 May 2020) approved the establishment of the Group of Experts on Conceptual and Technical Aspects of Computerization of the TIR Procedure (WP.30/GE.1) until 2022, based on the terms of reference included in document ECE/TRANS/WP.30/2019/9 and Corr.1, as contained in document ECE/TRANS/294 (ECE/EX/2020/L.2, para. 5(b)).[3]

* This document was submitted late for processing since clearance in finalizing this document took longer than anticipated.

[1]Decision of the Inland Transport Committee para 84 / ECE/TRANS/294 www.unece.org/fileadmin/DAM/trans/doc/2020/itc/ECE-TRANS-294e.pdf

[2]Terms of reference of the newly established Group approved by the Inland Transport Committee and the Executive Committee (EXCOM) of UNECE www.unece.org/fileadmin/DAM/trans/bcf/wp30/documents/2019/ECE-TRANS-WP30-2019-09e.pdf and corrigendum www.unece.org/fileadmin/DAM/trans/bcf/wp30/documents/2019/ECE-TRANS-WP30-2019-09c1e.pdf

[3]Decision of EXCOM , ECE/EX/2020/L.2 / para 5(b) www.unece.org/fileadmin/DAM/commission/EXCOM/Agenda/2020/Remote_informal_mtg_20_05_2020/Item_4_ECE_EX_2020_L.2_ITC_Sub_bodies_E.pdf

2. The terms of reference of the Group stipulate that the Group should focus its work on preparing a new version of the eTIR specifications, pending the formal establishment of TIB. More specifically the Group should (a)prepare a new version of the technical specifications of the eTIR procedure, and amendments thereto, ensuring their alignment with the functional specifications of the eTIR procedure; (b)prepare a new version of the functional specifications of the eTIR procedure, and amendments thereto, ensuring their alignment with the conceptual specifications of the eTIR procedure; (c)prepare amendments to the conceptual specifications of the eTIR procedure, upon requests by WP.30.

3. The current document presents the glossary and the technical procedures describing how to create a X.509 certificate and test the connection to the eTIR web services. This document is valid for the eTIR international system version 1.0 based on the eTIR specifications version 4.3a.

II. Fallback procedures

4. 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.

III. Support and contact

5. 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

IV. 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.

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 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	
Version of the SOAP protocol	v1.2	
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	

Key	*Value*	*Comments*
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

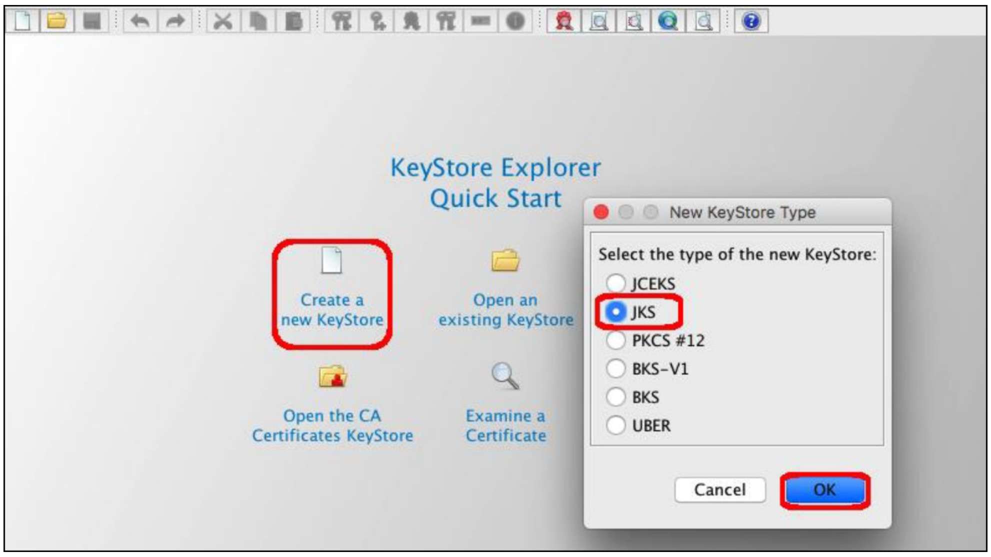
V. KeyStore: step by step generation of key pairs

A. 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/

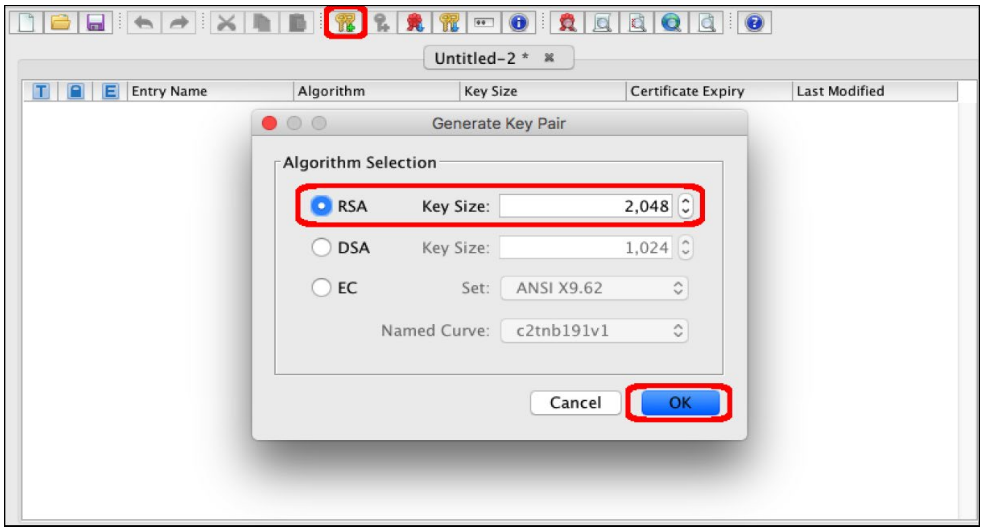
Key pair generation

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

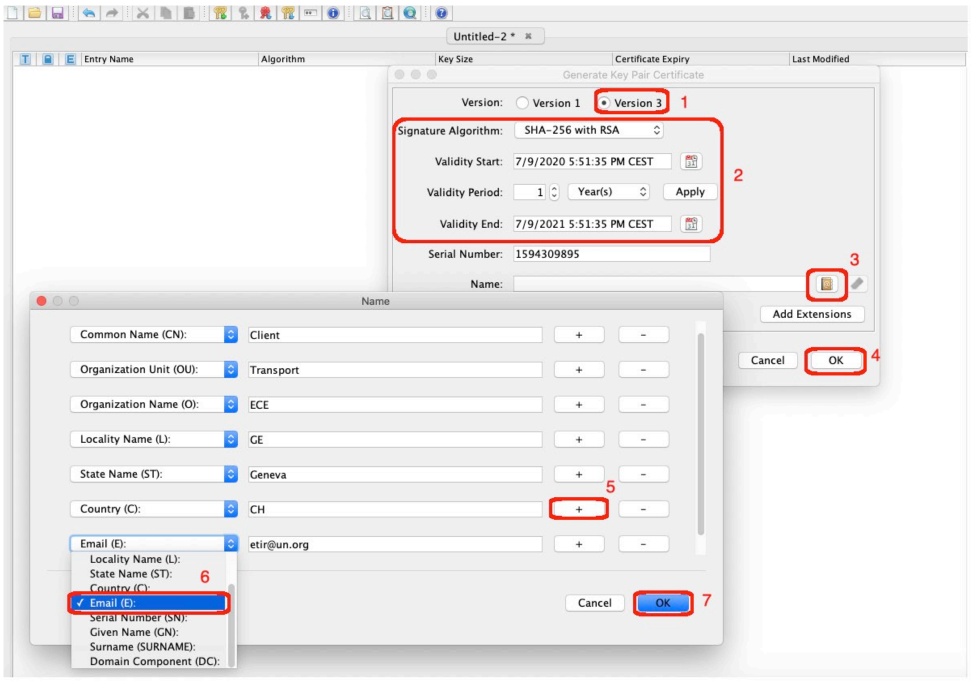
2. Generate Key Pair: choose RSA as algorithm with 2048 key size



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.


Generate Key Pair Certificate

Version: Version 1 [✓ Version 3] 1

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: 3

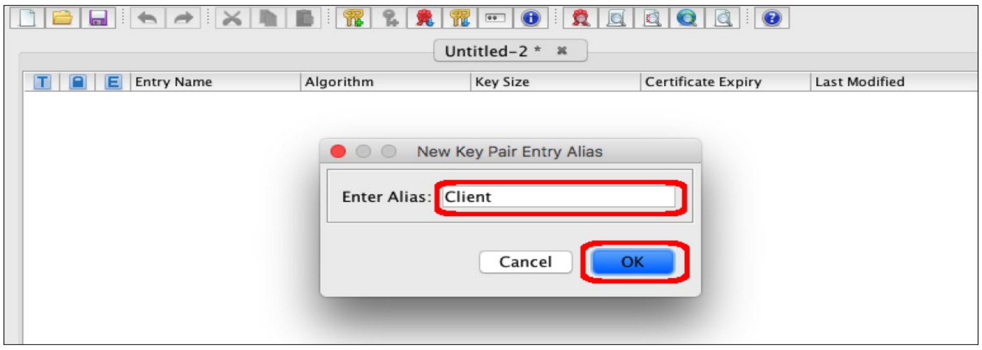
Add Extensions

Cancel OK 4

Name

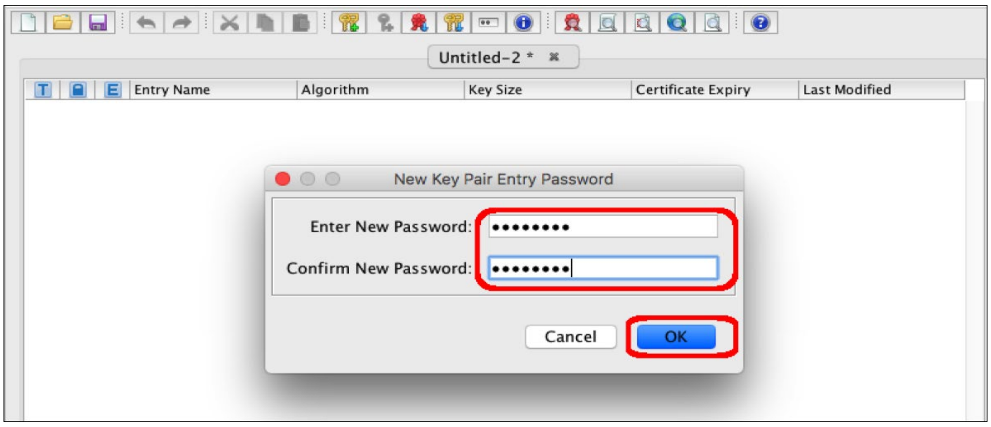
Common Name (CN):	Client	+	-
Organization Unit (OU):	Transport	+	-
Organization Name (O):	ECE	+	-
Locality Name (L):	GE	+	-
State Name (ST):	Geneva	+	-
Country (C):	CH	+ 5	-
Email (E):	etir@un.org	+	-

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


New Key Pair Entry Alias

Enter Alias: Client

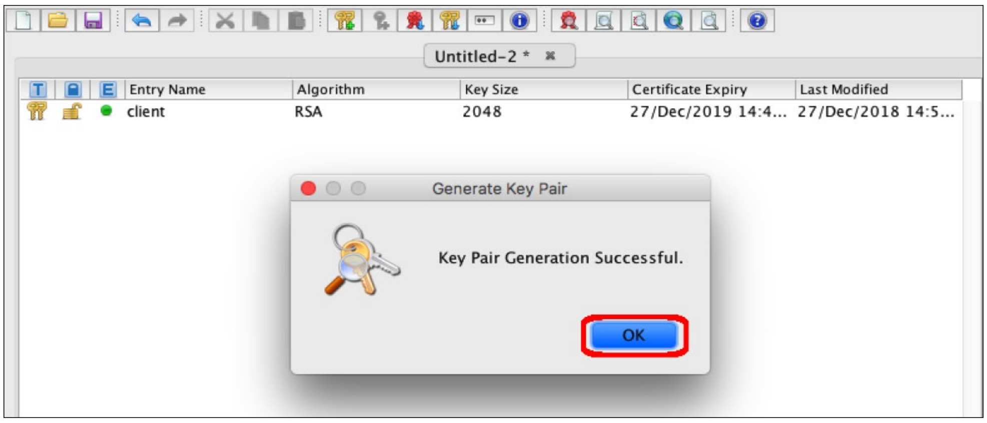
5. Another popup appears to define a key pair password.


New Key Pair Entry Password

Enter New Password: ••••••••

Confirm New Password: ••••••••

6. Key pair generation confirmed.



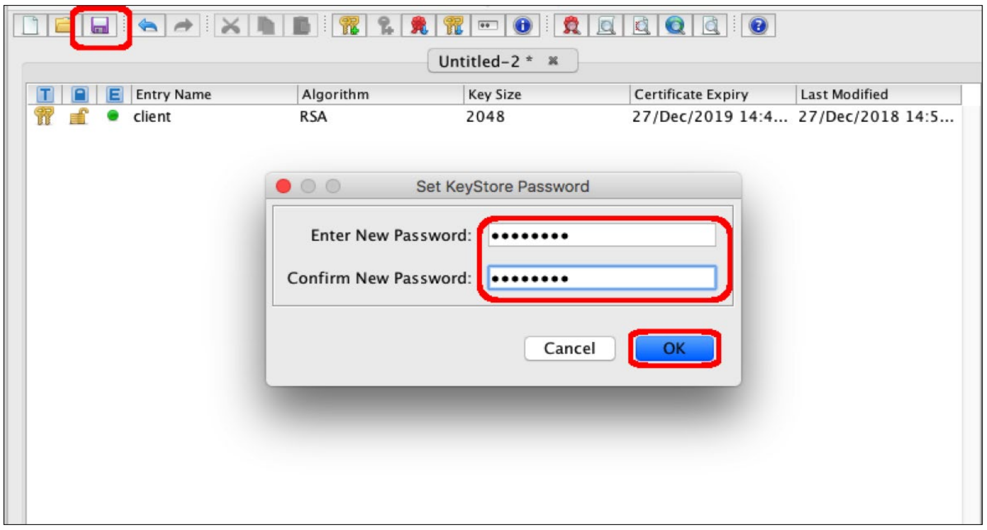
Untitled-2 *

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

Generate Key Pair

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).



Untitled-2 *

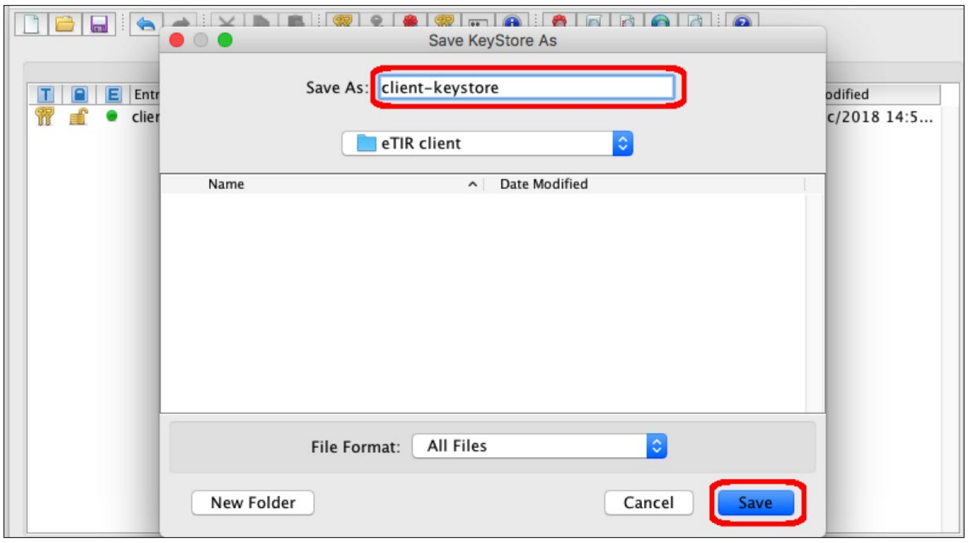
			Entry Name	Algorithm	Key Size	Certificate Expiry	Last Modified
		●	client	RSA	2048	27/Dec/2019 14:4...	27/Dec/2018 14:5...

Set KeyStore Password

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 KeyStore As

Save As: client-keystore

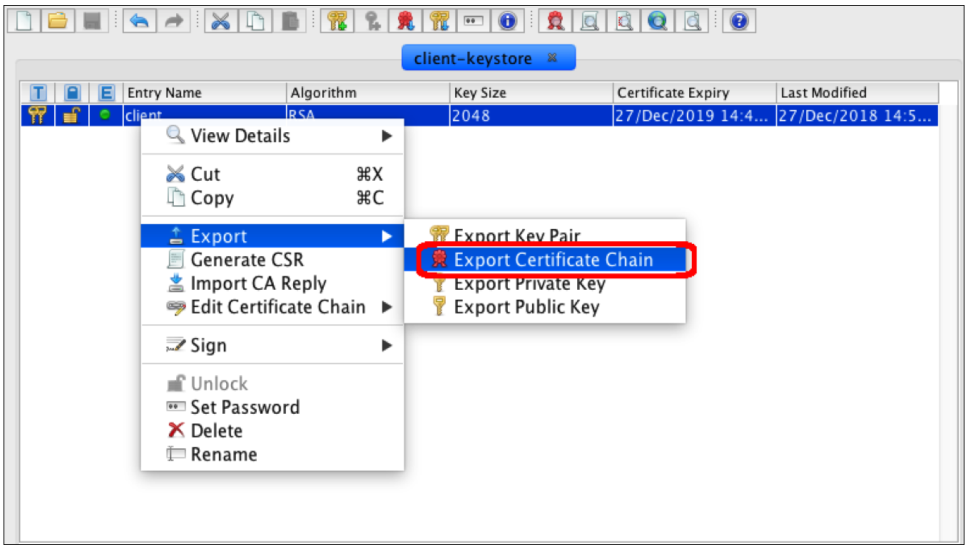
eTIR client

Name	Date Modified

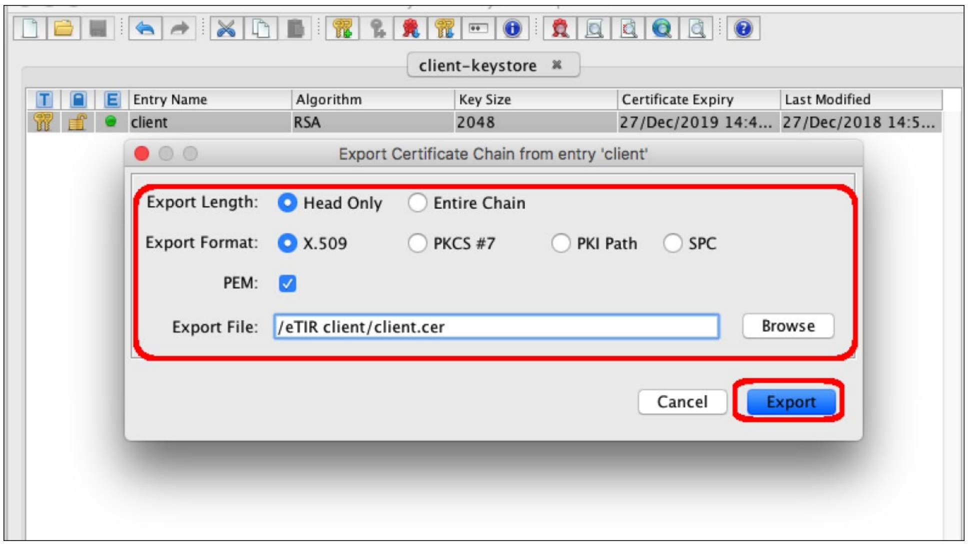
File Format: All Files

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 Certificate Chain from entry 'client'

Export Length: [✓ Head Only] Entire Chain

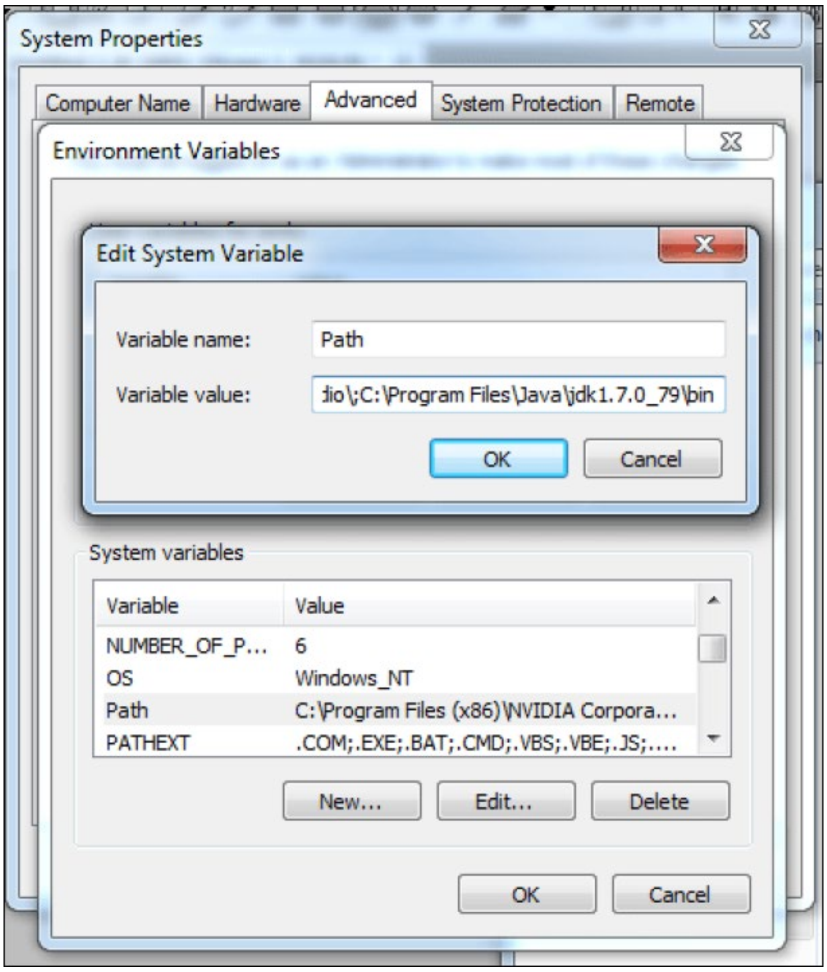
Export Format: [✓ X.509] PKCS #7 PKI Path SPC

PEM: ✓

Export File: /eTIR client/client.cer

B. 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:

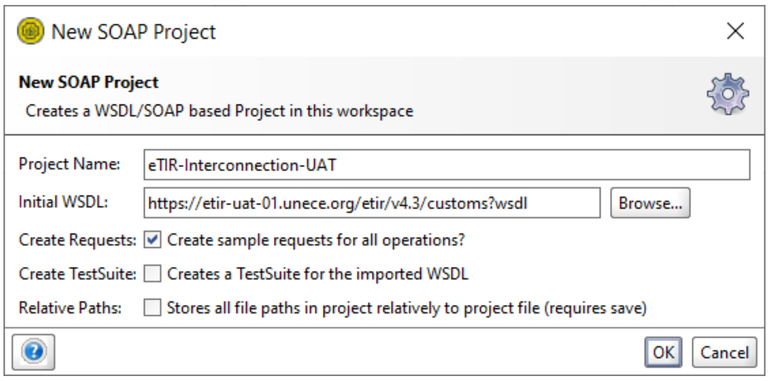
```
Certificate stored in file <client.cer>
```

VI. SOAP UI quick guide

Please find below the basic steps required to install and configure SOAP UI testing tool, and for sending an encrypting a SOAP request via SoapUI.

1. Download and install the SoapUI Open Source edition application from SOAP UI website download web page.

2. 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-Interconnection-UAT

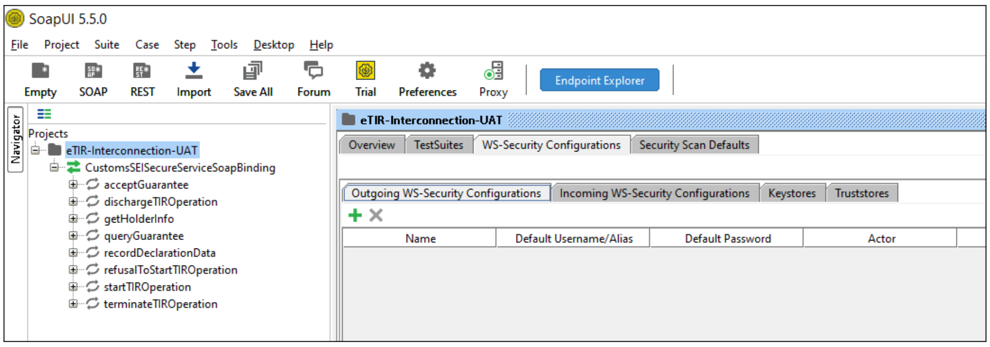
Initial WSDL: https://etir-uat-01.unece.org/etir/v4.3/customs?wsdl

Create Requests: [✓] Create sample requests for all operations?

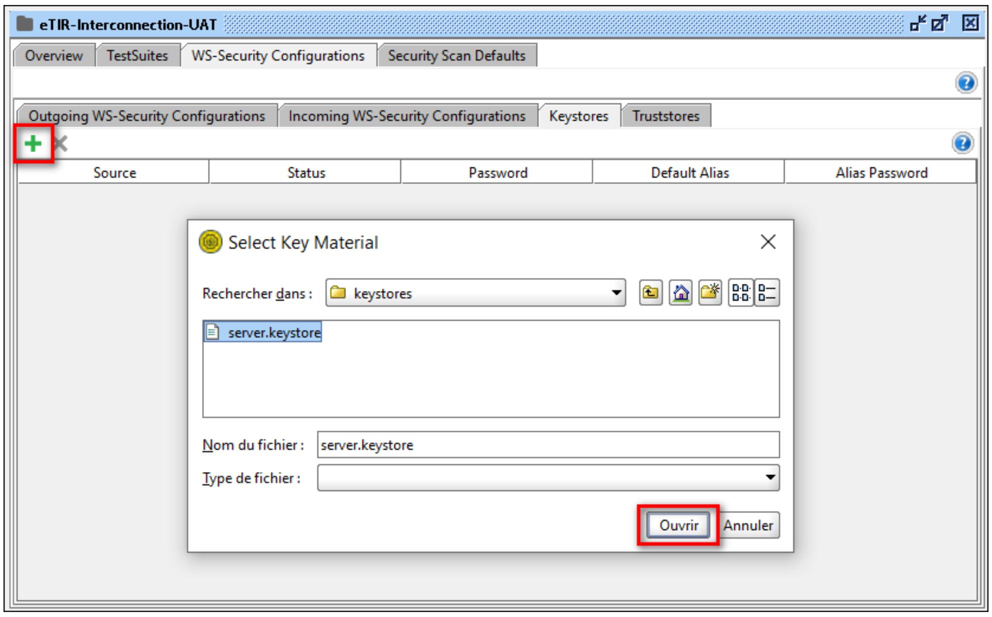
Create TestSuite: [ ] Creates a TestSuite for the imported WSDL

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

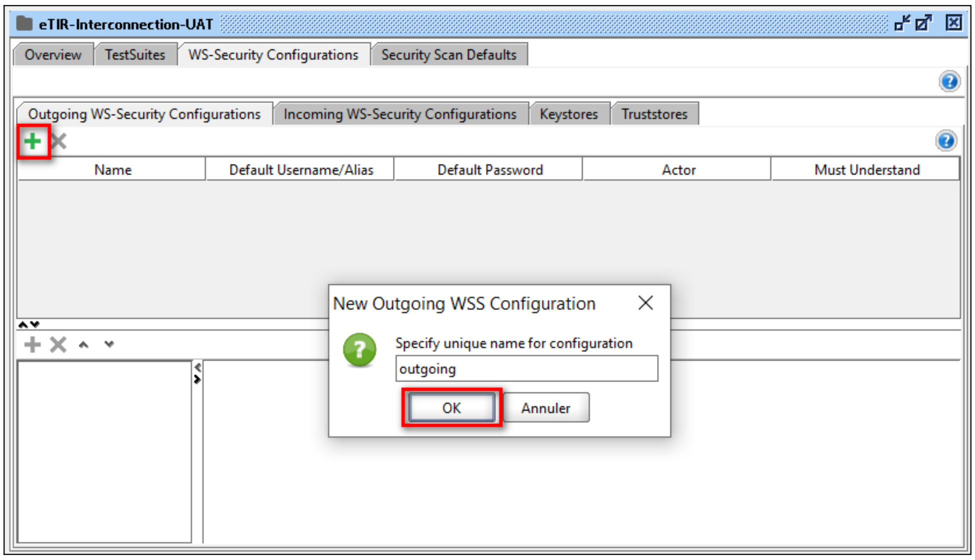
3. After creating a test project, go to the WS-Security Configurations tab inside the 'Show Project View' Menu by right clicking the project folder.


SoapUI 5.5.0
File Project Suite Case Step Tools Desktop Help
Empty SOAP REST Import Save All Forum Trial Preferences Proxy
Endpoint Explorer
Navigator
Projects
eTIR-Interconnection-UAT
CustomsSEISecureServiceSoapBinding
acceptGuarantee
dischargeTIROperation
getHolderInfo
queryGuarantee
recordDeclarationData
refusalToStartTIROperation
startTIROperation
terminateTIROperation
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

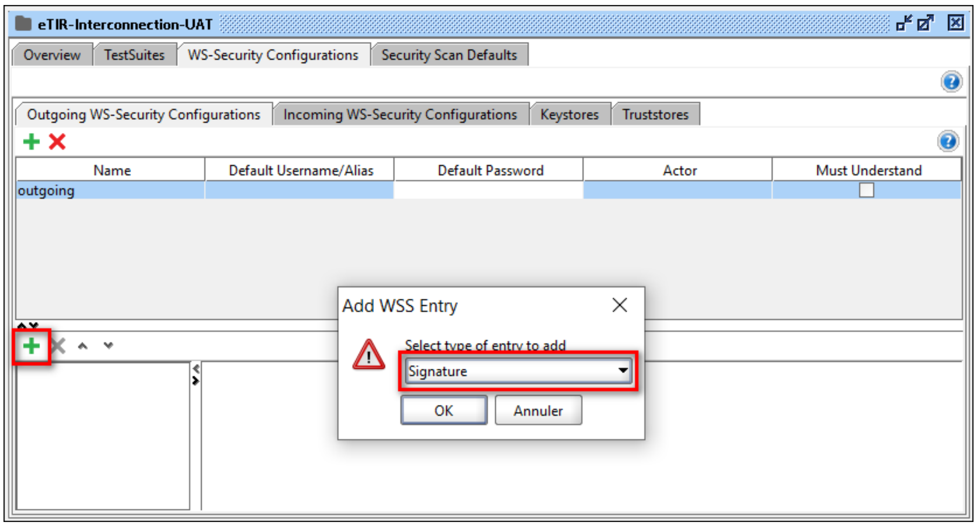
4. Add a KeyStore by clicking the add button and browsing to your keystore file, enter the password and click ok.


eTIR-Interconnection-UAT
Overview TestSuites WS-Security Configurations Security Scan Defaults
Outgoing WS-Security Configurations Incoming WS-Security Configurations Keystores Truststores
Source Status Password Default Alias Alias Password
Select Key Material
Rechercher dans : keystores
server.keystore
Nom du fichier : server.keystore
Type de fichier :
Ouvrir Annuler

5. Then click on the Outgoing WS-Security Configurations tab, to display the configurations that should be applied to outgoing messages, including requests. Add a new outgoing configuration.

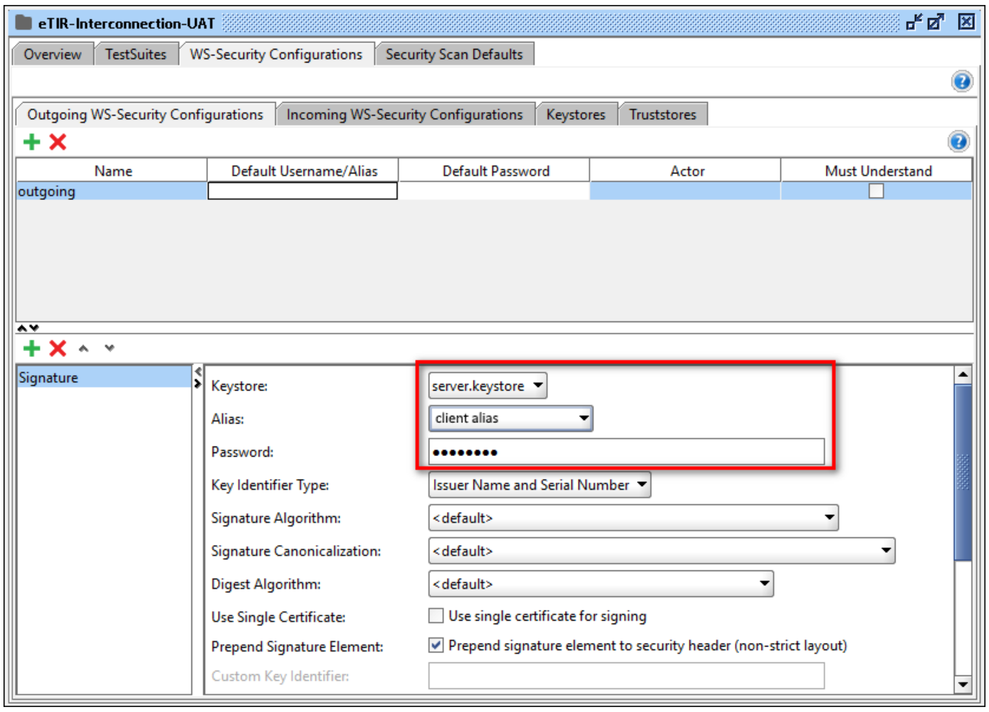

eTIR-Interconnection-UAT
Overview TestSuites WS-Security Configurations Security Scan Defaults
Outgoing WS-Security Configurations Incoming WS-Security Configurations Keystores Truststores
Name Default Username/Alias Default Password Actor Must Understand
New Outgoing WSS Configuration
Specify unique name for configuration
outgoing
OK Annuler

6. This configuration type is used for encryption, signing and adding SAML, timestamp and username headers. In this situation, only the signing part is used. Add a signature entry.



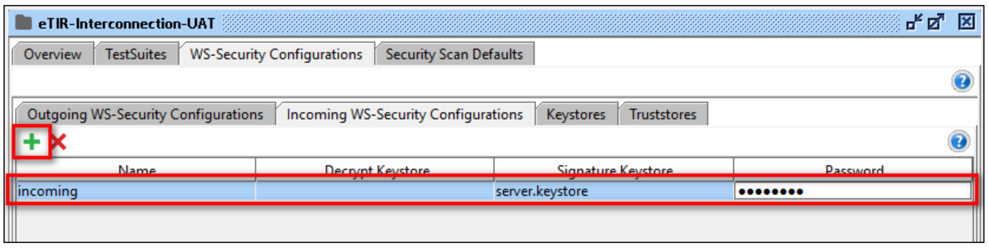
7. Select the keystore and key alias (key) that will be used along with the password for that alias. These are the configurable fields:

- **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.



Keystore: server.keystore
Alias: client alias
Password: ••••••••
Key Identifier Type: Issuer Name and Serial Number
Signature Algorithm: <default>
Signature Canonicalization: <default>
Digest Algorithm: <default>
Use Single Certificate: Use single certificate for signing
Prepend Signature Element: ✓ Prepend signature element to security header (non-strict layout)
Custom Key Identifier:

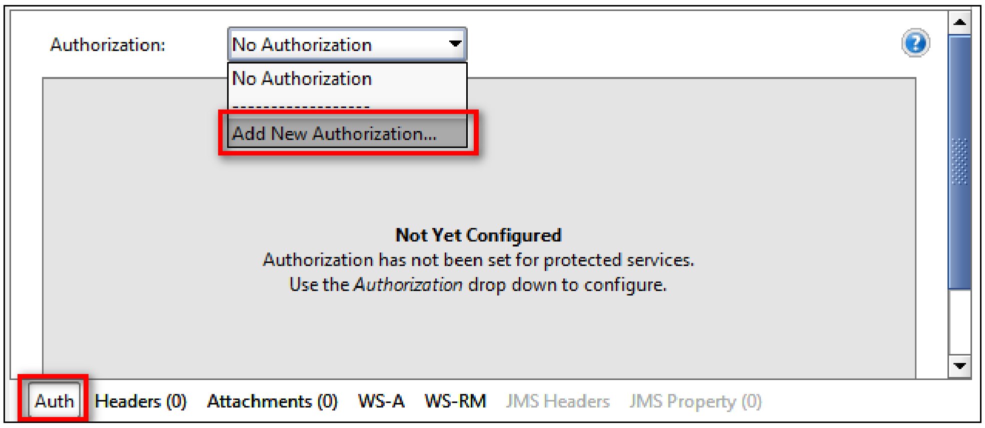
8. Click on the **Incoming WS-Security Configurations** tab, to configure what should be applied to incoming messages, including responses. Specify the keystore name and the password. This configuration will ensure the validation of the signature of the incoming responses from eTIR server.


eTIR-Interconnection-UAT
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		server.keystore	••••••••

9. 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.

10. 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.

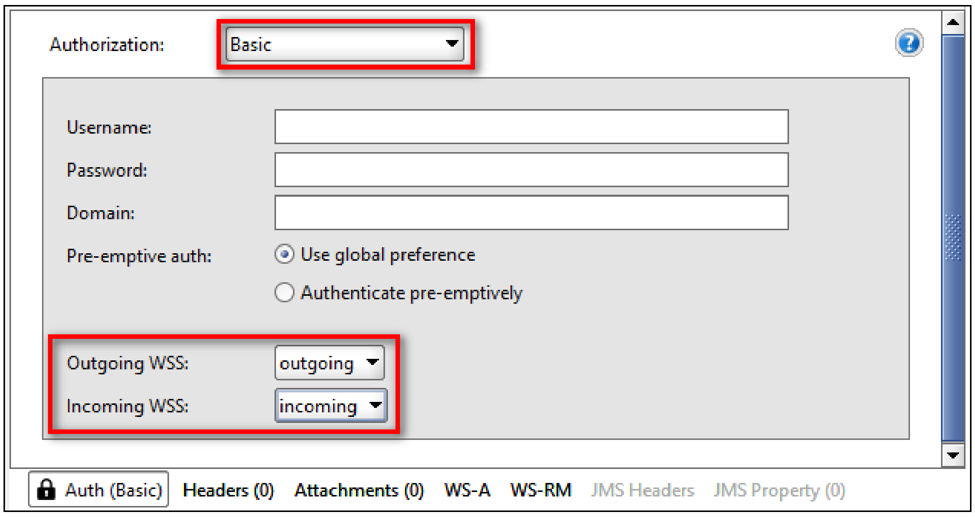

Authorization: No Authorization

No Authorization

Add New Authorization...

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

11. 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

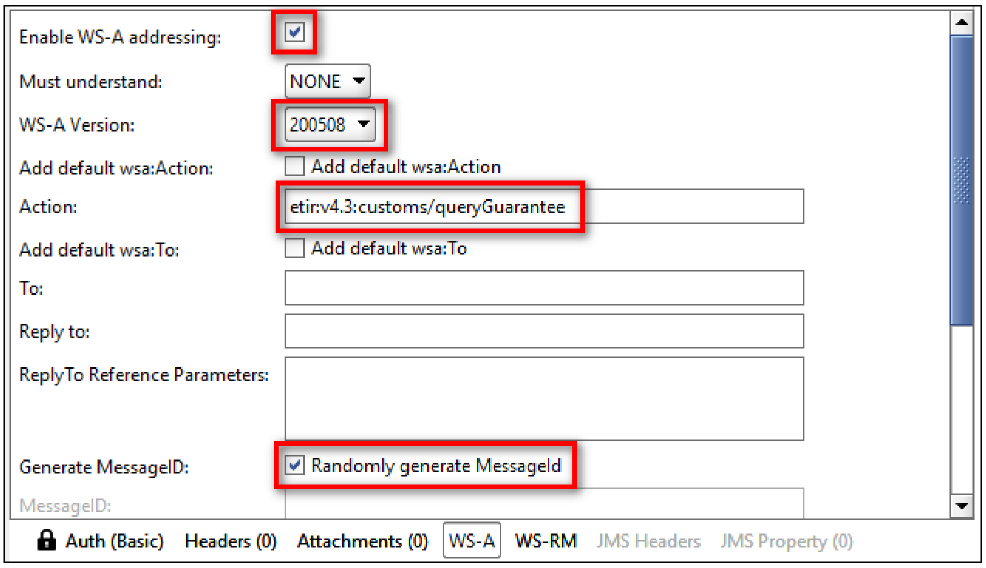
Outgoing WSS: outgoing
Incoming WSS: incoming

12. 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: etir:v4.3:customs/queryGuarantee

Add default wsa:To: Add default wsa:To

To:

Reply to:

ReplyTo Reference Parameters:

Generate MessageID: ✓ Randomly generate MessageId

MessageID:

13. Run SoapUI against the Web Service by passing Soap Messages to the server. You can use the example below:

```
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope" xmlns:etir="etir:v4.3:customs" xmlns:etir1=
"etir:I5:v4.3">
   <soap:Header/>
   <soap:Body>
      <etir:queryGuarantee>
         <etir1:InterGov>
            <etir1:FunctionCode>9</etir1:FunctionCode>
            <etir1:ID>e393d591-963a-4cd1-9d4a-08467afcf524</etir1:ID>
            <etir1:TypeCode>I5</etir1:TypeCode>
            <etir1:ReplyTypeCode>00</etir1:ReplyTypeCode>
            <etir1:ObligationGuarantee>
               <etir1:ReferenceID>XC95000003</etir1:ReferenceID>
            </etir1:ObligationGuarantee>
         </etir1:InterGov>
      </etir:queryGuarantee>
   </soap:Body>
</soap:Envelope>
```

VII. 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:

```
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());
```