INTERCOPE

BOX MESSAGING HUB

Version 3 Release 23 (V3R23)

Security Implementation Guide

Revision 2.0

Table of Contents

1 Introduction in Security

1.1 A Very Brief Introduction to Cryptography

This chapter gives an insight into some major cryptography terms and their meanings that will help you to understand the **BOX** security concepts and their technical implementation and configuration.

1.1.1 Information Security Services

Cryptography is a common, widespread and fundamental aspect of information security. Generally, it is a method of storing and transmitting data - or to be more precise: 'information' - in a form, so that only those for whom it is intended can read and process it.

The primary objective of using cryptography is to provide the following four fundamental information `security services`:

Privacy/Confidentiality: It is a `security service` that keeps the information from an unauthorized person and ensures that no one else can read or process the information except the intended person.

Authentication: This `security service` is the process of proving one's identity. Authentication provides the identification of the originator. It confirms to the receiver that the information received has been sent only by an identified and verified sender.

Integrity: It is a `security service` that deals with identifying any alteration to the information. The integrity service confirms that whether the information is still intact or not since it was last created, transmitted, or stored by an authorized user.

Non-repudiation: It is a `security service` that ensures that the original creator of the information can't deny the creation or transmission of the said information to a recipient.

1.1.2 Cryptosystem

In cryptography, a `cryptosystem` is a suite of cryptographic algorithms needed to implement a particular `security service`. Typically, a `cryptosystem` takes the original information -the un-encrypted information-, transforms it into `cipher` information -the encrypted information- by using a `cipher` -an encryption algorithm or method- and a `key` -a value used in conjunction with a `cipher` to create `cipher` information, which will in turn usually be decrypted back into usable original information.

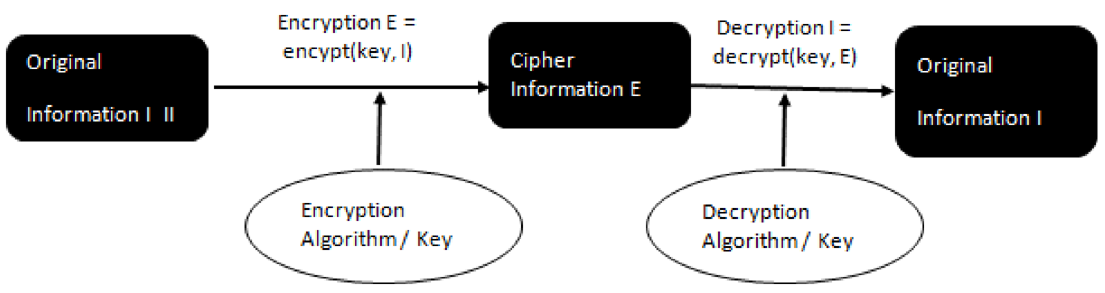

Original Information I II
Encryption E = encypt(key, I)
Cipher Information E
Decryption I = decrypt(key, E)
Original Information I
Encryption Algorithm / Key
Decryption Algorithm / Key

Figure 1. Encryption/Decryption

The method of disguising information in such a way as to hide its content is called **encryption**. Encrypting information results in unreadable gibberish called `cipher information`. The process of reverting `cipher` information to its original information is called `decryption`**.** A `cipher` is an algorithm (series of well-defined steps - a procedure) for performing encryption or decryption. Such procedure is depending on a piece of auxiliary information, called a `key`. At the end without knowledge of the `key`, it should be unfeasible to decrypt the resulting `cipher` information into its original. Generally, the longer the `key` size, the stronger the encryption. However, longer `keys` also result in longer processes of encryption/decryption.

1.1.3 Type of Cryptosystems

Cryptosystems can be broken down into a set of encryption categories:

Symmetric Encryption (Secret-key cryptography): This kind of `cryptosystem` uses a single `key` for both encryption and decryption. Symmetric-key systems are simple and fast, but their main drawback is that the two parties of the information transfer must somehow exchange the `key` in a secure way. Most of the symmetric `ciphers` used today are block `ciphers` which encrypt a fixed size of n-bits of information - known as a block - at one time.

Two of the most widely used block `ciphers` are:

DES (Data Encryption Standard). DES is a 64-bit `cipher` that works with a 64-bit `key`. 8 of the 64 bits in the `key` are parity bits, so the `key` size is technically 56 bits long.

AES (Advanced Encryption Standard), AES is arguably the most widely used block `cipher` in the world. It has a block size of 128 bits and supports three possible `key` sizes - 128, 192, and 256 bits.

Asymmetric Encryption (Public-key cryptography): This `cryptosystem` uses a mathematically related `key` pair for encryption and decryption. One `key` of the `key` pair is known as the `private key` which is kept secret, and other one is the `public key` which is shared. Though `private` and `public keys` are related mathematically, it is infeasible to calculate the `private key` from the `public key`. A user can encrypt information with another person's `public key` to ensure that only the person with the matching `private key` can decrypt it. Or they could encrypt information with their `private key` to mark their ownership of the information (Non-repudiation) and allow anyone with access to the `public key` to decrypt it.

Two of the most widely used asymmetric `key` algorithms are:

RSA (Rivest-Shamir-Adleman). RSA derives its security from the difficulty of factoring large integers that are the product of two large prime numbers. Multiplying these two numbers is easy but determining the original prime numbers from the total factoring is considered infeasible due to the time it would take. RSA `keys` can be typically 1024 or 2048 bits long.

DSA (Digital Signature Algorithm). DSA is designed as part of the Digital Signature Standard (DSS). DSA's security is based on the 'discrete logarithm problem'.

Hashing Encryption: This `cryptosystem` uses a mathematical process to perform a calculation against the information and returns a numeric or **hash value**. Unlike other `cryptosystem`s, hashing is a one-way system, this means that *it is impossible to use the hash value to deduce the original information*. Hashing is an ideal way to store passwords, as hashes are inherently one-way in their nature.

The `hash value` does serve as a highly effective **checksum (sometimes called information/message digest)**, meaning that it allows a way of seeing if the information has changed. If the checksum of information at the recipient is different to that of the originator, the information was manipulated. One of the `key` properties of hashing algorithms is determinism. The `hash value` of the same information is always the same. In this way hashing serves the purpose of ensuring integrity. Hashing also can be used to compare a large amount of information. Hash values can be created for different information, meaning that it is easier to compare hashes than the information itself.

Popular hashing algorithms are:

MD5. MD5 is the most widely known hashing algorithm. It produces a 16-byte hash value. Recently a few vulnerabilities have been discovered in MD5.

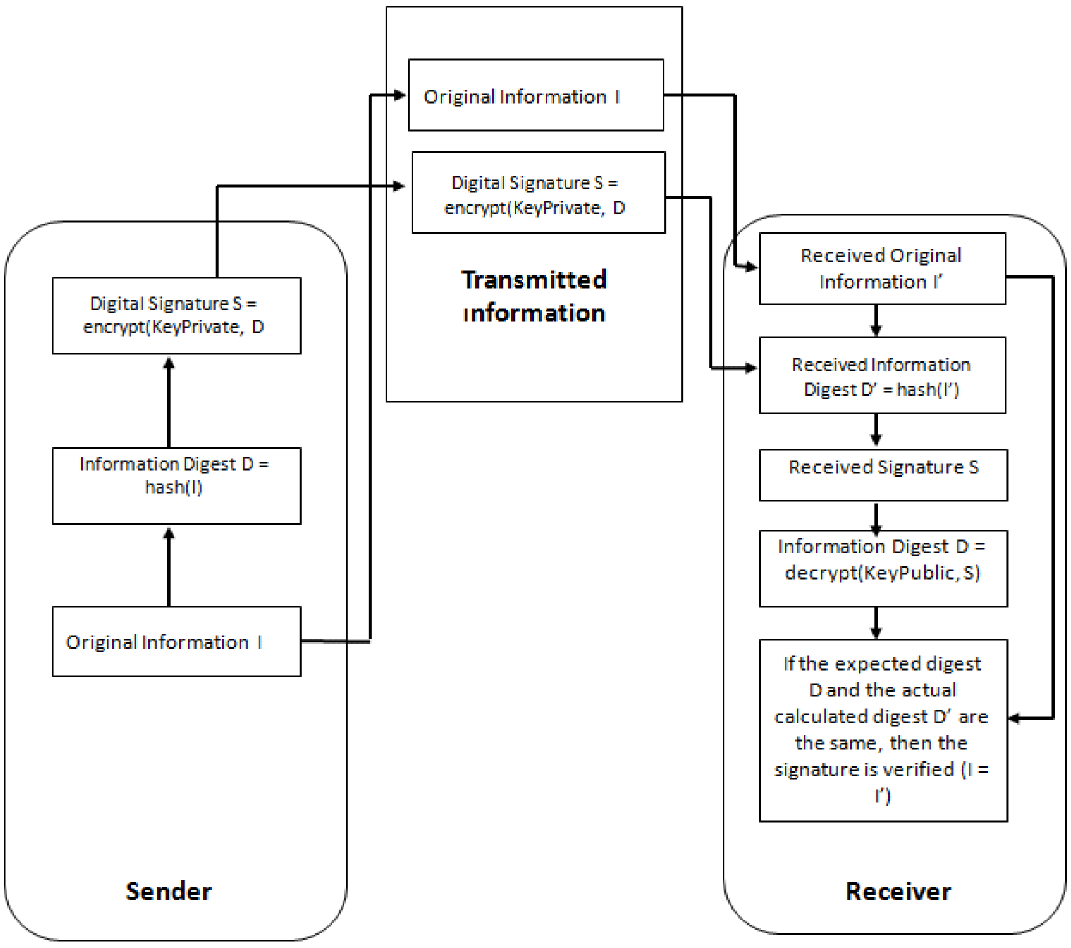
SHA (Secure Hash Algorithm). SHA is a family of cryptographic hash functions and comes in a variety of lengths, the most popular being 256-bit.

To defend against the so-called 'dictionary attacks', an additional information called **salt** is usually appended to the original information before the hashing process.

1.1.4 Digital Signatures

A **digital signature** uniquely identifies the originator of digitally signed information and ensures the integrity of the signed information. The most common types of digital signatures today are created by signing the digest of the information (hash value) with the originator's `private key` to create a digital thumbprint of the information which is attached to the original information itself.

To verify the contents of digitally signed information, the recipient generates a new information digest from the information that was received, decrypts the original information digest with the originator's `public key`, and compares the decrypted digest with the newly generated digest. If the two digests match, the integrity of the message is verified. The identity of the originator also is confirmed because the `public key` can decrypt only data that has been encrypted with the corresponding `private key` of the originator. Two of the most widely used digital signature algorithms today are the RSA digital signature process and the Digital Signature Algorithm (DSA).



Original Information I
Digital Signature S = encrypt(KeyPrivate, D
Transmitted information
Digital Signature S = encrypt(KeyPrivate, D
Information Digest D = hash(I)
Original Information I
Sender
Received Original Information I'
Received Information Digest D' = hash(I')
Received Signature S
Information Digest D = decrypt(KeyPublic, S)
If the expected digest D and the actual calculated digest D' are the same, then the signature is verified (I = I')
Receiver

Figure 2. Digital Signature Process

1.1.5 Digital Certificates

`Digital certificates` (also known as `public key` certificates) are usually based on the ITU standard **X.509** which defines a standard certificate format for `public key` certificates and certification validation. Hence digital certificates are sometimes also referred to as X.509 certificates.

`Certificate authorities (CAs)` are entities that validate identities, sign and issue certificates. The certificate issued by the CA binds a particular `public key` to the name of the entity that the certificate identifies. Only the `public key` that is certified by the certificate will work with the corresponding `private key` that is owned by the entity identified by the certificate.

`X.509 certificates` can also be signed by its owner. This kind of certificates are known as `Self-signed certificates` and are considered as untrusted.

A X.509 certificate normally contains the following information:

Subject identification information	Information such as name, unique identifier of the certificate's owner.
Subject public key **value**	The public key of the certificate's owner
Certification authority's information	Name of the CA that issued the certificate, a serial number, and other information such as expiration date.
Certification authority's digital signature	The digital signature for the CA, which is obtained by hashing all the information in the certificate together and encrypting it with the CA private key.

X.509 certificates are available in various formats. The most widely used formats are:

PEM (Privacy Enhanced Mail)	PEM formats file have Base64(ASCII) encoded ASN.1 DER (Distinguished Encoding Rules) certificate, enclosed between the tags "BEGIN CERTIFICATE" and "END CERTIFICATE".
P12 (PKCS#12), PFX (Personal Information Exchange)	This is an archive file format that includes both the private key and the X.509 certificate itself. It is protected by password.

1.1.6 How to Convert X.509 Certificate in PFX Format into PEM Format

OpenSSL is a full-featured toolkit for the Transport Layer Security (TLS) and Secure Sockets Layer (SSL) protocols.

Additionally, OpenSSL includes the program ‘openssl’ for requesting, generating and administrating certificates via command line. The program can be downloaded from www.openssl.org.

For converting an `X.509 certificate` in PFX format into PEM format enter

```
openssl pkcs12 -in certificate.pfx -passin pass:certificatepassword
-clcerts -nokeys -out publiccertificate.pem
```

1.1.7 How to Extract the Private Key in PEM Format from X.509 Certificate in PFX Format

For extracting the `Private Key` in PEM format from X.509 certificate in PFX format enter

```
openssl pkcs12 -in certificate.pfx -passin pass:certificatepassword
-passout  -nodes -nocerts -out privatekey.pem
```

1.1.8 How to Create a Self-signed Certificate

Generate a new certificate to be signed by Intercope.

Create self-signed certificate [Output, Private-Key: mynewprivatekey.pem, X509-Certificate: mynewpublicx509cetificate.pem]

```
openssl req -newkey rsa:2048 -nodes -keyout mynewprivatekey.pem -x509
-days 365 -out mynewpublicx509certificate.pem
```

1.2 INTERCOPE Security Concept

The INTERCOPE security concept rests upon some basic principles.

According to these principles the code provided by INTERCOPE is signed before being handed out to the customer. Furthermore, each configuration (file) must be signed, as well as each record in the database. Accordingly, all configuration changes as well as modified database records must be signed. Any missing signature will be alerted.

Additionally, database integrity is ensured by regular checking of the database at record level to verify that no gaps exist in sequential numbering of the Message Processing Sequences (MPS), see 6.2.

Another basic principle rests on both the INTERCOPE customer as well as INTERCOPE using their own `private key` to encrypt sensitive information such as configurations, data in database, passwords, and so on. The main model terms are:

`Signed Configuration Certificate (SCC)`: The Signed Configuration Certificate is simply a piece of information that contains the `customer X.509 certificate` and its digital signature signed by INTERCOPE. The `Signed Configuration Certificate` will normally be used by BOX for the information verification process to ensure that the owner of the encrypted information is authorized by INTERCOPE in addition to the normal digital signature verification.

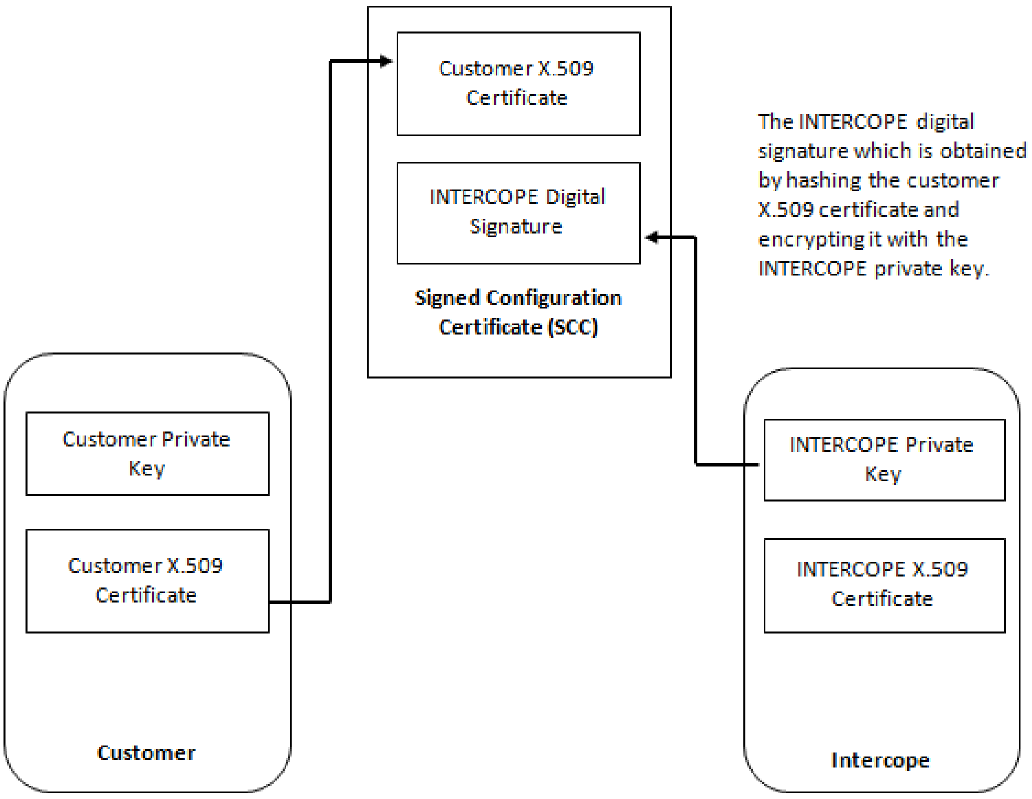

Customer X.509 Certificate
INTERCOPE Digital Signature
Signed Configuration Certificate (SCC)
The INTERCOPE digital signature which is obtained by hashing the customer X.509 certificate and encrypting it with the INTERCOPE private key.
Customer Private Key
Customer X.509 Certificate
Customer
INTERCOPE Private Key
INTERCOPE X.509 Certificate
Intercope

Figure 3. Signed Configuration Certificate

Master Encryption Key (MEK):

The master encryption `key` is an `AES-256`-bit `cipher`. Its purpose is to ensure secure access to other sensitive data, i.e. to the `BSL` (see below). Customers should generate and digitally sign their own `MEK`.

Base Secrets List (BSL):

The base secrets list represents a secure storage of customer defined secrets and contains a list of symmetric `keys` and their associated information such as used encryption algorithm, PHASH-Salts.

Please note, The BSL is deployed with the product software in encrypted format.

2 Onboarding Process

One part of the Onboarding Process is creating a Signed Customer Certificate.

This begins with the Customer sending an X509 certificate to INTERCOPE.

1. INTERCOPE then creates a Digital Certificate Envelope (XML) and signs it with the `INTERCOPE private signing key`.
2. This Signed Customer Certificate is then handed out to the Customer together with an `INTERCOPE eMEK (encrypted Master Encryption Key)` and `INTERCOPE eBSL (encrypted Base Security List)`
3. After this it is up to the Customer to generate an own `eMEK` for the BOX instance and then to re-encrypt and reinitialize the standard eBSL with the own `eMEK`.
4. Generating the eMEK, as well as re-encrypting and reinitializing the eBSL is done with the INTERCOPE Security Tool `(icopesecurity.jar)` which is part of the delivery of BOX

   (see below, chapter 3).
5. ‘Grep’ and ‘unzip’ is required for the user to perform certain steps within this document.

Additionally, INTERCOPE will deliver scripts (see 5.4) that will facilitate the onboarding process for the customer.

IMPORTANT:

The Customer must make sure that test and production systems use different eMEKs and eBSLs and take care of the Private Key not being stored on the BOX system for production.

3 INTERCOPE Security Tool

The delivered BOX software includes the **INTERCOPE Security Tool** (**icopesecurity.jar**) that can be used for security-relevant options. These options include:

- Checking if `icopesecurity.jar` is genuine
- Generating, signing and verifying Digital Certificate envelopes
- Generating and verifying Digital Signature files
- Encrypting passwords
- Generating and encrypting MEK files
- Encrypting and re-encrypting BSL
- Re-initializing encrypted BSL
- Verifying .jar and .war files
- Extracting data from the delivered `security.zip` file
- Inserting files into the `security.zip` file
- Upgrading an expired certificate´
- SSL JDBC Server Connection
- Possibility to append some meta information to the file signature

3.1 Checking if icopesecurity.jar is genuine

For every new icopesecurity.jar release the genuineness of the jar file should be verified. This can be done

a) utilising the fingerprint of the INTERCOPE-Code-Certificate (Hex String-Format), more specifically the SHA-256 digest of X509-Data in ASN.1 DER-representation.

```
[SHA256:03:A0:A3:C6:11:EA:8A:83:BC:2D:12:59:09:54:B8:30:D2:66:68:4C:B
E:8E:B7:E7:04:17:89:76:4D:21:3C:43]
```

b) the SHA-256-Digest of the icopesecurity.jar file itself (Hex String-Format).
To verify the authenticity of the jar file, the following steps must be performed:

Step1
Calculate the SHA-256-Digest of icopesecurity.jar using openssl and compare the resulting information with the SHA-256-Digest of the icopesecurity.jar file itself

Example

```
openssl dgst -sha256 icopesecurity-1.0.83.jar
SHA256(icopesecurity-1.0.83.jar)=
5d:dd:9e:36:42:17:3f:ce:a1:ac:fc:92:63:85:fb:08:70:5d:2e:96:40:ab:1f:
8b:04:97:a5:59:a4:5f:63:96
```

Step 2
Verify the fingerprint of icopesecurity.jar by using the standard Java-Tool 'jarsigner'

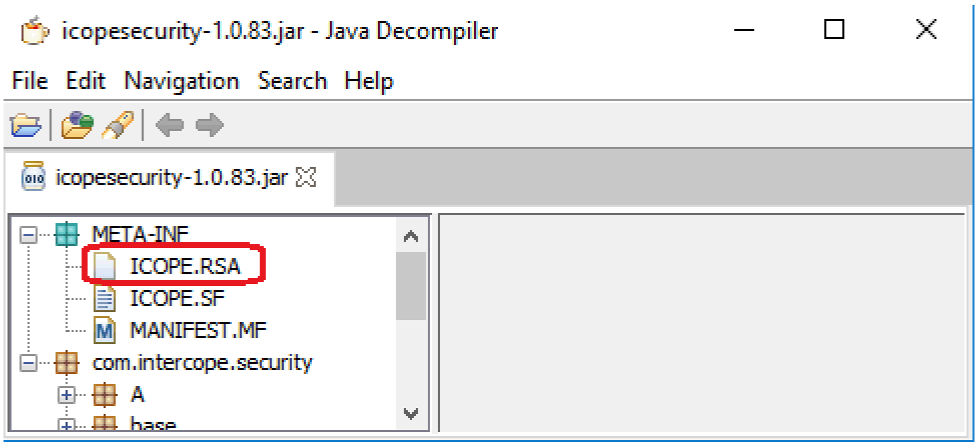
Example

```
jarsigner -verify -verbose -certs icopesecurity-1.0.83.jar
>icopesecurity-verification-report.txt
```

The resulting report `icopesecurity-verification-report.txt` states

a) icopesecurity.jar was not manipulated and
b) only the party owning the private key (INTERCOPE) which fits the Public-Key from the verification report, can generate the icopesecurity.jar

Step 3
To verify the fingerprint of the INTERCOPE-Code-Certificate (comparison with a)), the file 'ICOPE.RSA' (dir META-INF) must be extracted from the icopesecurity.jar.



ICOPE.RSA consist of X509 data in PKCS#7 format. To export a readable version, the standard Java-Tool 'keytool' is used:

```
keytool -printcert -file ICOPE.RSA > Intercope-X509-Info.txt
```

Excerpt

```
      .
Owner: EMAILADDRESS=info@intercope.com, CN=INTERCOPE International
Communication Products Engineering GmbH, O=INTERCOPE International
Communication Products Engineering GmbH, L=Hamburg, ST=Hamburg, C=DE
Issuer: CN=GlobalSign CodeSigning CA - SHA256 - G3, O=GlobalSign nv-
sa, C=BE
Serial number: 308e74d32456c789d5c2ad08
Valid from: Wed Oct 19 16:17:28 CEST 2016 until: Sun Jan 19 15:17:28
CET 2020
Certificate fingerprints:
       MD5:  CE:20:EB:CB:35:5D:70:81:70:61:C6:CF:51:E2:AC:5F
       SHA1:
4A:9B:55:6F:2B:F6:73:2A:7A:40:A6:3E:33:23:93:06:BD:07:BB:65
       SHA256:
03:A0:A3:C6:11:EA:8A:83:BC:2D:12:59:09:54:B8:30:D2:66:68:4C:BE:8E:B7:
E7:04:17:89:76:4D:21:3C:43
       Signature algorithm name: SHA256withRSA
```

3.2 Verifying Digital Certificate Envelopes

After INTERCOPE has generated and signed the Digital Certificate Envelope and sent it to the customer, it is up to the customer to verify the Certificate Envelope.

For this purpose, the INTERCOPE Security Tool `icopesecurity.jar` must be run with the option

```
java -jar icopesecurity.jar -verifycertenvelope [-cfgcertfile <cert.
envelope file name>] [-securityzipfile <INTERCOPE security ZIP file
which contains the Cert. Envelope file>]
```

-cfgcertfile	Certificate Envelope file name.	Not needed if parameter -**securityzipfile** is given.
-securityzipfile	INTERCOPE security ZIP file which contains the Certificate Envelope file.	Not needed if parameter -**cfgcertfile** is given

Examples:

```
java -jar icopesecurity.jar  -verifycertenvelope -cfgcertfile
SignedConfigurationCertificate.sig.xml
INTERCOPE Certificate Envelope file
'/opt/box/server22/security/SignedConfigurationCertificate.sig.xml' successfully
verified.
(Period Of Validity: Certificate Envelope[Not before: not given, Not on or
after: not given], Certificate[Not before: '2018-01-04T09:21:27Z', Not after:
'2019-01-04T09:21:27Z'], INTERCOPE Certificate[Not before: '2016-10-
19T14:17:28Z', Not after: '2020-01-19T14:17:28Z'])
```

```
java -jar icopesecurity.jar  -verifycertenvelope -securityzipfile
/opt/box/server22/security/security.zip
INTERCOPE Certificate Envelope file 'SignedConfigurationCertificate.sig.xml'
found in '/opt/box/server22/security/security.zip' successfully verified.
(Period Of Validity: Certificate Envelope[Not before: not given, Not on or
after: not given], Certificate[Not before: '2018-01-04T09:21:27Z', Not after:
'2019-01-04T09:21:27Z'], INTERCOPE Certificate[Not before: '2016-10-
19T14:17:28Z', Not after: '2020-01-19T14:17:28Z'])
```

3.3 Generating and Verifying File Signatures

Customized Configuration Files for the BOX Client, for the BOX Server and for Java Tools as well as BOX Replacement files must be digitally signed before the changes can take effect.

For this purpose, the **INTERCOPE Security Tool** `icopesecurity.jar` must be run with the option

```
java -jar icopesecurity.jar -genfilesignature [-certtype <Type of cert>]
[-keyfile <private key file name>] [-certpassword <cert. password>]
[-certalias <cert. alias>] [-certfile <cert. file name>]
[-file <name of file to be signed>] [-dir <name of directory which
contains the file(s) to be signed>] [-outdir <name of output directory>]
[-securityzipfile <INTERCOPE security ZIP>]
```

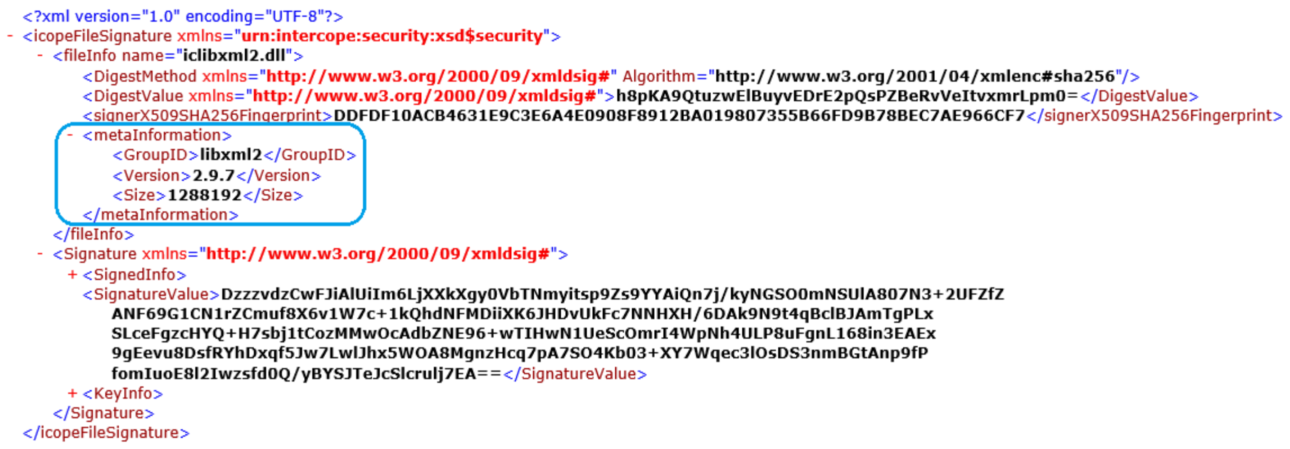
-certtype	Type of Certificate to be used for file signing. Must be one of PFX, P12, PEM or JKS.
-key**file**	Private key file name. Only needed in case of PEM.
-certpassword	Certificate password. Only needed in case of PFX, P12 or JKS.
-certalias	Certificate alias. Only needed in case of PFX, P12 or JKS.
-certfile	Name of Certificate file.
-file	Name of file to be signed.
-dir	Name of directory which contains the file(s) to be signed
-outdir	Name of output directory (optional, only used if parameter -securityzipfile is not given).
-securityzipfile	INTERCOPE security ZIP file which contains the file signature (optional).
-metaInfoKeyN	<key name of the meta information N (optional)>
-metaInfoKeyValueN	<key value of the meta information N(optional)>

Examples:

```
java -jar icopesecurity.jar -genfilesignature -certtype PEM -keyfile
myprivatekey.pem -certfile mypublicx509certificate.pem -file data.txt
java -jar icopesecurity.jar -genfilesignature -certtype PEM -keyfile
myprivatekey.pem -certfile mypublicx509certificate.pem -file data.txt -
securityzipfile security.zip

java -jar icopesecurity.jar -genfilesignature -certtype PFX -
certpassword mypwd -certalias myalias -certfile mypfxcert.pfx -file
data.txt

java -jar icopesecurity.jar -genfilesignature -certtype PEM -keyfile
devprivatekey.pem -certfile devpublicx509cetificat.pem -metaInfoKey1
GroupID -metaInfoKeyValue1 libxml2 -metaInfoKey2 Version -
metaInfoKeyValue2 2.9.7 -metaInfoKey3 Size -metaInfoKeyValue3 1288192 -
file iclibxml2.dll
```



```
  <?xml version="1.0" encoding="UTF-8"?>
- <icopeFileSignature xmlns="urn:intercope:security:xsd$security">
  - <fileInfo name="iclibxml2.dll">
      <DigestMethod xmlns="http://www.w3.org/2000/09/xmldsig#" Algorithm="http://www.w3.org/2001/04/xmlenc#sha256"/>
      <DigestValue xmlns="http://www.w3.org/2000/09/xmldsig#">h8pKA9QtuzwElBuyvEDrE2pQsPZBeRvVeItvxmrLpm0=</DigestValue>
      <signerX509SHA256Fingerprint>DDFDF10ACB4631E9C3E6A4E0908F8912BA019807355B66FD9B78BEC7AE966CF7</signerX509SHA256Fingerprint>
    - <metaInformation>
        <GroupID>libxml2</GroupID>
        <Version>2.9.7</Version>
        <Size>1288192</Size>
      </metaInformation>
    </fileInfo>
  - <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
    + <SignedInfo>
      <SignatureValue>DzzzvdzCwFJiAlUiIm6LjXXkXgy0VbTNmyitsp9Zs9YYAiQn7j/kyNGSO0mNSUlA807N3+2UFZfZ
        ANF69G1CN1rZCmuf8X6v1W7c+1kQhdNFMDiiXK6JHDvUkFc7NNHXH/6DAk9N9t4qBclBJAmTgPLx
        SLceFgzcHYQ+H7sbj1tCozMMwOcAdbZNE96+wTIHwN1UeScOmrI4WpNh4ULP8uFgnL168in3EAEx
        9gEevu8DsfRYhDxqf5Jw7LwlJhx5WOA8MgnzHcq7pA7SO4Kb03+XY7Wqec3lOsDS3nmBGtAnp9fP
        fomIuoE8l2Iwzsfd0Q/yBYSJTeJcSlcrulj7EA==</SignatureValue>
    + <KeyInfo>
    </Signature>
</icopeFileSignature>
```

File signatures can be verified with the option
`java -jar icopesecurity.jar -verifyfilesignature`

-datafile	Name of the file that contains data to be verified.
-signaturefile	Name of the file that contains data signature.
-cfgcertfile	Name of the Certificate Envelope file.

Example:
```
java -jar icopesecurity.jar -verifyfilesignature -datafile mydata.txt
-signaturefile mydata.txt.sig.xml -cfgcertfile
SignedConfigurationCertificate.sig.xml
```

3.4 Encrypting Passwords

BOX Database passwords and BOX UPM passwords must be encrypted.

For this purpose, the **INTERCOPE Security Tool** `icopesecurity.jar` must be run with the option

```
java -jar icopesecurity.jar -encryptpassword [-password <password to be
encrypted> [-emekfile <name of emek file>] [-encbslfile <name of eBSL
file>] [-bslsigfile <name of bsl signature file>] [-cfgcertfile <cert.
envelope file name>] [-securityzipfile <INTERCOPE security ZIP file>]
```

-password	Password to be encrypted.	
-emekfile	Name of the file that contains the Encrypted Master Encryption Key.	Not needed if parameter -securityzipfile is given.
-encbslfile	Name of the file that contains the encrypted Base Secret List.	Not needed if parameter -securityzipfile is given.
-bslsigfile	Name of the file that contains the signature of Base Secret List.	Not needed if parameter -securityzipfile is given.
-cfgcertfile	Name of the Certificate Envelope file. Not needed if parameter -securityzipfile is given.	
-securityzipfile	INTERCOPE security ZIP file which contains the information about E-MEK (-emekfile), E-BSL (-encbslfile), E-BSL signature (-bslsigfile) and Cert. Envelope (-cfgcertfile).	

Examples:

```
java -jar icopesecurity.jar -encryptpassword -password mypwd -emekfile
mek.enc -encbslfile BaseSecretList.xml.enc -bslsigfile
BaseSecretList.xml.sig.xml -cfgcertfile
SignedConfigurationCertificate.sig.xml

java -jar icopesecurity.jar -encryptpassword -emekfile mek.enc
-encbslfile BaseSecretList.xml.enc -bslsigfile
BaseSecretList.xml.sig.xml -cfgcertfile
SignedConfigurationCertificate.sig.xml

java -jar icopesecurity.jar -encryptpassword -password mypwd -
securityzipfile security.zip

java -jar icopesecurity.jar -encryptpassword -securityzipfile
security.zip
```

Also, other data than passwords can be encrypted. For this purpose, the **INTERCOPE Security Tool** `icopesecurity.jar` must be run with the option

```
java -jar icopesecurity.jar -encryptsecret [-secret <secret to be
encrypted> [-type <key type>] [-emekfile <name of emek file>] [-
encbslfile <name of eBSL file>] [-bslsigfile <name of bsl signature
file>] [-cfgcertfile <cert. envelope file name>] [-securityzipfile
<INTERCOPE security ZIP file>]
```

-secret	Secret to be encrypted.	
-type	Key type (optional, default type is 'password'. For other secrets than passwords, e.g. for LAU values, use type 'any').	
-emekfile	Name of the file that contains the Encrypted Master Encryption Key.	Not needed if parameter -securityzipfile is given.
-encbslfile	Name of the file that contains the encrypted Base Secret List.	Not needed if parameter -securityzipfile is given.
-bslsigfile	Name of the file that contains the signature of Base Secret List.	Not needed if parameter -securityzipfile is given.
-cfgcertfile	Name of the Certificate Envelope file.	Not needed if parameter -securityzipfile is given.
-securityzipfile	INTERCOPE security ZIP file which contains the information about E-MEK (-emekfile), E-BSL (-encbslfile), E-BSL signature (-bslsigfile) and Cert. Envelope (-cfgcertfile).	

Examples:

```
java -jar icopesecurity.jar -encryptsecret -secret myscrt -type any -
emekfile mek.enc -encbslfile BaseSecretList.xml.enc -bslsigfile
BaseSecretList.xml.sig.xml -cfgcertfile
SignedConfigurationCertificate.sig.xml
```

```
java -jar icopesecurity.jar -encryptsecret -type any -emekfile mek.enc -
encbslfile BaseSecretList.xml.enc -bslsigfile BaseSecretList.xml.sig.xml
-cfgcertfile SignedConfigurationCertificate.sig.xml
```

```
java -jar icopesecurity.jar -encryptsecret -secret myscrt -type any -
securityzipfile security.zip
```

```
java -jar icopesecurity.jar -encryptsecret -type any -securityzipfile
security.zip
```

3.5 Generating and Encrypting Master Encryption Keys

After having verified the Certificate Envelope received from INTERCOPE the Customer must generate an own Master Encryption Key (MEK) and encrypt it.

For generating the own MEK, the **INTERCOPE Security Tool** `icopesecurity.jar` runs with the option

```
java -jar icopesecurity.jar –generatemek [-mekfile <Encrypted-MEK file>]

-mekfile Name of Encrypted-MEK file (optional).
```

Example:

```
java -jar icopesecurity.jar -generatemek -mekfile mymek.mek
```

For encrypting the own MEK the **INTERCOPE Security Tool** `icopesecurity.jar` must be run with the option

```
java -jar icopesecurity.jar -encryptmek [-mek <Master Encryption Key>]
[-mekfile <Master Encryption Key file name>] [-keyfile <private key file
name>] [-certtype <Type of cert. to be used for encryption>]
[-certpassword <cert. password>] [-certalias <cert. alias>] [-certfile
<cert. file name>] [-emekfile <name of Encrypted-MEK file>]
```

-mek	Master Encryption Key. Optional, if both -mek and -mekfile parameters are not given, then a new MEK will be automatically generated.
-mekfile	Name of file that contains the Master Encryption Key (optional)].
-key**file**	Name of private key file. Only needed in case of PEM.
-certtype	Type of Certificate to be used for encryption. Must be one of PFX, P12, PEM or JKS.
-certpassword	Certificate password. Only needed in case of PFX, P12 or JKS.
-certalias	Certificate alias. Only needed in case of PFX, P12 or JKS.
-certfile	Name of Certificate.
-emekfile	Name of Encrypted-MEK file (optional).

Examples:

```
java -jar icopesecurity.jar -encryptmek -certtype PEM -keyfile
myprivatekey.pem -certfile mypublicx509certificate.pem

java -jar icopesecurity.jar encryptmek -mek
E44E312A321EAC837C5B55DDCC70D9BF1717DFCDA63A5B915A081B80AAD3C748A4B48C48
F864353F0783A26EBBF7C240 -certtype PEM -keyfile myprivatekey.pem -
certfile mypublicx509certificate.pem

java -jar icopesecurity.jar -encryptmek -mekfile mymek.mek -certtype PEM
-keyfile myprivatekey.pem -certfile mypublicx509certificate.pem
-emekfile myemek.enc
```

3.6 Extracting Entries from Security Zip File

The onboarding process requires that some entries are extracted from the `security.zip` file delivered by INTERCOPE and then replaced by corresponding, customized entries.

Extracting entries from the `security.zip` File is done by running the **INTERCOPE Security Tool** `icopesecurity.jar` with the option

```
java -jar icopesecurity.jar -extractseczipentries [-securityzipfile
<security ZIP file>] [-outdir <name of output directory>]
[-zipentrynames <list of entry names>]
```

-securityzipfile	Name of INTERCOPE security ZIP file.
-outdir	Name of output directory (optional).
-zipentrynames	List of entry names in security-zip to be extracted

Example:

```
java -jar icopesecurity.jar -extractseczipentries -securityzipfile
security.zip -zipentrynames ProductSecurityCfgDefinitions.xml
```

3.7 Re-encrypting and Initializing encrypted BSL

As mentioned above the onboarding process requires that some entries in the `security.zip` file delivered by INTERCOPE must be replaced by corresponding, customized entries.

One of these entries is the standard encrypted Base Secret List (eBSL). Before the eBSL can be inserted into the `security.zip`, it must be reencrypted and reinitialized.

3.7.1 Re-Encrypting BSL

For re-encrypting the eBSL the **INTERCOPE Security Tool** `icopesecurity.jar` must be run with the option

```
java -jar icopesecurity.jar –reencryptbsl [-encbslfile <name of eBSL
file>] [-emekfile <name of EMEK file>] [-cfgcertfile <cert. envelope
file name>] [-newemekfile <name of file with new EMEK>]
[-newcfgcertfile <NEW cert. envelope file>]
```

-encbslfile	Name of the file that will contain the encrypted Base Secret List.
-emekfile	Name of the file that contains the Encrypted Master Encryption Key.
-cfgcertfile	Name of the Certificate Envelope file.
-newemekfile	Name of the file that contains the new Encrypted Master Encryption Key.
-newcfgcertfile	Name of the NEW Certificate Envelope file.

Example:

```
java -jar icopesecurity.jar -reencryptbsl -encbslfile
BaseSecretList.xml.enc -emekfile mek.enc -cfgcertfile
SignedConfigurationCertificate.sig.xml -newemekfile newmek.enc
-newcfgcertfile mynewcert.pem.env.xml
```

3.7.2 Re-Initializing BSL

For reinitializing the eBSL the INTERCOPE Security Tool `icopesecurity.jar` must be run with the option

```
java -jar icopesecurity.jar –reinitializeencryptbsl [-encbslfile <eBSL
file name>] [-bslsigfile <eBSL sign. file name>] [-emekfile <EMEK file
name>] [-cfgcertfile <cert. envelope file name>] [-comment <comment
text>] [-reinitencryptionkeylist (flag whether the encryption key also
should be reinitialized] [-certtype <Type of cert>] [-keyfile <private
key file name>][-certpassword <cert. password>] [-certalias <cert.
alias>] [-certfile <cert. file name>]
```

-encbslfile	Name of the file that will contain the encrypted Base Secret List.
-bslsigfile	Name of the file that contains the signature of Base Secret List.
-emekfile	Name of the file that contains the Encrypted Master Encryption Key.
-cfgcertfile	Name of the Certificate Envelope file.

-comment	Comment text. Must be given!
-reinitencryptionkeylist	Flag indicating whether the Encryption Key also should be reinitialized. Optional
-certtype	Type of Certificate. To be used for the reinitialized eBSL signing. Must be one of PFX, P12, PEM or JKS.
-keyfile	Name of private key file. Only needed in case of PEM.
-certpassword	Certificate password. Only needed in case of PFX, P12 or JKS.
-certalias	Certificate alias. Only needed in case of PFX, P12 or JKS.
-certfile	Name of Certificate file.

Example:

```
java -jar icopesecurity.jar -reinitializeencryptbsl -encbslfile
BaseSecretList.xml.enc -bslsigfile BaseSecretList.xml.sig.xml
-emekfile mek.enc -cfgcertfile SignedConfigurationCertificate.sig.xml
-comment \"Reinit by John Doe\" -reinitencryptionkeylist -certtype PEM
-keyfile myprivatekey.pem -certfile mypublicx509certificate.pem
```

3.8 Inserting Entries Into security.zip

As mentioned above the onboarding process requires that some entries in the `security.zip` file delivered by INTERCOPE must be replaced by corresponding, customized entries.

Such entries are for instance the new reencrypted and reinitialized Base Secret List, the encrypted Master Encryption Key file and the Signed Configuration Certificate.

For inserting files into the `security.zip`, the **INTERCOPE Security Tool** `icopesecurity.jar` must be run with the option

```
java -jar icopesecurity.jar -insertseczipentries [-securityzipfile
<INTERCOPE security ZIP file>] [-files <file list>]
```

-securityzipfile	Name of INTERCOPE security ZIP file.
-files	List of files to be inserted into security-zip.

Example:

```
java -jar icopesecurity.jar -insertseczipentries -securityzipfile
security.zip -files mek.enc SignedConfigurationCertificate.sig.xml
origsecurity/BaseSecretList.enc origsecurity/BaseSecretList.xml.sig.xml
```

3.9 Upgrading an Expired Certificate

If the respective certificate is expired, then it should be upgraded. In this case a new option '-renewcertificate' is available support the certificate upgrade process.
In case of certificate upgrade option '-renewcertificate' should be used to update the following entries within the file security.zip:

```
- mek.enc
- SignedConfigurationCertificate.sig.xml
- BaseSecretList.enc
- BaseSecretList.xml.sig.xml
- all file signatures with name subifx '.sig.xml'
```

```
java -jar icopesecurity.jar -renewcertificate [Upgrade INTERCOPE
security ZIP to new certificate][-securityzipfile <INTERCOPE security
ZIP file>][-newcfgcertfile <cert. envelope file name (the new one)>]
[-certtype <Type of cert. to be used for the BSL signing: must be one of
PFX,P12,PEM or JKS>][-keyfile <private key file name, only needed in
case of PEM>][-certpassword <cert. password, only needed in case of
PFX,P12 or JKS>][-certalias <cert. alias, only needed in case of PFX,P12
or JKS>][-certfile <cert. file name>]
```

-renewcertificate	Upgrade of expired certificate

3.9.1 Steps to Upgrade Expired Certificate

For the item security.zip, a X509 Certificate is required, preferably in PEM Format.
Supported formats are: `PFX,P12,PEM,CER,CRT,DER,P7B,P7C or JKS`.
If the format of your X509 Certificate is PFX, 'ASN.1- DER Format 'Stricted' is required, otherwise there might be problems extracting keys when signing config files.
A PFX file can be converted from `'Unstricted' to 'Stricted'` Format with `openssl tools`.
If not sure about the format, the pfx can be converted if necessary, at a later stage.

3.9.1.1 Send a new certificate to be signed by Intercope

- Send a (self-signed) customer signed certificate [Output, Private-Key: mynewprivatekey.pem, X509-Certificat: mynewpublicx509cetificate.pem] to Intercope

- Verify the received certificate envelope newSignedConfigurationCertificate.sig.xml (only for checking purpose) sent by Intercope

  ```
  java -jar icopesecurity.jar -verifycertenvelope -cfgcertfile
  newSignedConfigurationCertificate.sig.xml
  ```

3.9.1.2 Upgrade the security.zip with the newly signed certificate

- Upgrade security.zip [input/output: security.zip]

  ```
  java -jar icopesecurity.jar -renewcertificate -securityzipfile
  security.zip -newcfgcertfile
  newSignedConfigurationCertificate.sig.xml -certtype PEM -keyfile
  mynewprivatekey.pem -certfile mynewpublicx509certificate.pem
  ```

- Check whether the security.zip is OK or not (only for checking purpose)

```
java -jar icopesecurity.jar -testbsl -securityzipfile security.zip
```

Example:

```
java -jar icopesecurity.jar -renewcertificate -securityzipfile
security.zip -newcfgcertfile
newSignedConfigurationCertificate.sig.xml -certtype PEM -keyfile
mynewprivatekey.pem -certfile mynewpublicx509certificate.pem
```

IMPORTANT

If deployment scripts (please refer to chapter 5.4) are used, the `../security` structure needs to be updated with the new elements to generate the respective security.zip with the private– and public key and SignedConfigurationCertificate.sig.xml.

- Copy your new

  → public key `customer_publicx509certificate.pem` to folder `certs`
  → certificate envelope `SignedConfigurationCertificate.sig.xml` to folder `certs`
  → private key `customer_privatekey.pem` to folder `priv`

Also, copy the changed certificate and security.zip file to the for respective parameters configured path.

3.10 Supporting of IBM Common Cryptographic Architecture (CCA) interface for IBM z Platform

3.10.1 IBMJCECCA

The IBMJCECCA provider extends Java Cryptography Extension (JCE) and Java Cryptography Architecture (JCA) seamlessly to add the capability to use hardware cryptography using IBM Common Cryptographic Architecture (CCA) interfaces on both Linux for IBM z® and z/OS operating systems.

IBM CCA is a set of software elements that provide common application interfaces to secure, high-speed cryptographic services on various platforms using hardware cryptographic devices. On the z/OS platform, access to hardware cryptographic devices is controlled by the Integrated Cryptographic Service Facility (ICSF).

On IBM z® servers running Linux, access to hardware cryptographic devices is controlled by the IBM PCIe Cryptographic Coprocessor software.

More details about IBMJCECCA, please see
ftp://public.dhe.ibm.com//software/Java/Java80/IBMJCECCA/zOSHWCryptoRefGuide.html

3.10.2 Configuring and using hardware cryptographic devices on IBM z® Platform

To use a hardware cryptographic device, the appropriate card must be installed and configured according to the specifications that are provided with the card. On some platforms it is also necessary for the user or application to set up the cryptographic environment and provide access control (log into the card).

3.10.3 z/OS specifics

On the z/OS platform, access to hardware cryptographic devices is controlled by the Integrated Cryptographic Service Facility (ICSF). ICSF must be configured and running before the hardware cryptographic device is accessed.

3.10.4 Linux for IBM z® specifics

On the Linux for IBM z® platform, access to hardware cryptographic devices is controlled by the IBM PCIe Cryptographic Coprocessor software. IBM PCIe Cryptographic Coprocessor software must be configured and running before the hardware cryptographic device is accessed.

3.10.5 Configuring and using IBMJCECCA

To use the IBMJCECCA provider, you must add it to the 'java.security' file in the $JAVA_HOME/lib/security directory on z/OS and the $JAVA_HOME/jre/lib/security directory on Linux. If you add it as the first JCE provider in the list, it will be selected automatically for any service and algorithm that it supports if no provider (option -ksprovider) is specified. To add the IBMJCECCA provider as the first JCE provider, add the following to the java.security file:

```
#
# List of providers and their preference orders:
#
security.provider.1=com.ibm.crypto.hdwrCCA.provider.IBMJCECCA
security.provider.2=com.ibm.crypto.provider.IBMJCE
```

3.10.6 Testing the configuration of IBMJCECCA

To check whether the IBMJCECCA environment is correctly installed/configured or not, please use the option -`testracfcca`

```
java -jar icopesecurity.jar -testracfcca -ringownerid MYDEVKEYRINGOWNER
-ringid mydevkeyring
```

Please note, if the exception 'Hardware error from call CSNDPKB returnCode 12 reasonCode 0' has been thrown then the ICSF is not available.

For more details, please refer to:
https://www.ibm.com/support/knowledgecenter/en/SSLTBW_2.3.0/com.ibm.zos.v2r3.csfb400/csfb4880.htm

Try to open RACF input stream 'safkeyring://TEST/devkeyring'...

```
Processing Error:
Reflection error[Details: Instance of class
'com.ibm.crypto.hdwrCCA.provider.RACFInputStream' can't be
instantiated<Parameter types[java.lang.String;java.lang.String;[C]>]!
Call Stack:
:: :::   :::  :::
Caused by: com.ibm.crypto.hdwrCCA.provider.JCECCARuntimeException:
Hardware error from call CSNDPKB returnCode 12 reasonCode 0
        at
com.ibm.crypto.hdwrCCA.provider.KeyPairUtils.a(KeyPairUtils.java:1208)
        at
com.ibm.crypto.hdwrCCA.provider.KeyPairUtils.generatePrivateHWKey(KeyPai
rUtils.java:252)
        at
com.ibm.crypto.hdwrCCA.provider.RACFInputStream.getEntry(RACFInputStream
.java:408)
```

3.10.7 Creation of signed certificate

Step 1: Create self-signed x509-certificate (incl. private key) by using the java z/OS tool hwkeytool [Information: ftp://public.dhe.ibm.com//software/Java/Java80/IBMJCECCA/hwkeytool.html].

(Must be done by our customer)

```
hwkeytool -genkeypair -dname "CN=Dev. CCA Testing, OU=BOX Development,
O=Development, L=Hamburg, S=Hamburg, C=DE"
-alias mydevkeyalias -keystore
safkeyring://MYDEVKEYRINGOWNER/mydevkeyring -keysize 2048 -storetype
JCECCARACFKS
-providerName IBMJCECCA -J-
Djava.protocol.handler.pkgs=com.ibm.crypto.hdwrCCA.provider
```

Step 2: Verify the self-signed certificate:

```
java -jar icopesecurity.jar -showkeystoreinfocca -kstype JCECCARACFKS -
ringownerid MYDEVKEYRINGOWNER -ringid mydevkeyring -keyalias
mydevkeyalias
```

Step 3: Export the x509-certificate generated by step 1 as pem-format [Output: mydevpublicx509certificate.pem].

```
hwkeytool -exportcert -alias mydevkeyalias -keystore
safkeyring://MYDEVKEYRINGOWNER/mydevkeyring -storetype JCECCARACFKS -
providerName IBMJCECCA
-J-Djava.protocol.handler.pkgs=com.ibm.crypto.hdwrCCA.provider -rfc -
file mydevpublicx509certificate.pem
```

Step 4: Generate certificate envelope of mydevpublicx509cetificat.pem [Output: mySignedConfigurationCertificate.sig.xml]

```
java -jar icopesecurity.jar -gencertenvelopecca -certtype PEM -certfile
custompublicx509certificate.pem -ringownerid MYDEVKEYRINGOWNER -ringid
mydevkeyring -keyalias CUSTOMKEYLABEL -cfgcertfile
mySignedConfigurationCertificate.sig.xml
```

Step 5. Sign the new certificate envelope mySignedConfigurationCertificate.sig.xml [Input/Output: mySignedConfigurationCertificate.sig.xml]

(Must be done by INTERCOPE)

```
java -jar icopesecurity.jar -signcertenvelope -certtype PFX -
certpassword ????????? -certalias ?????????? -certfile OS201610179119-
stricted.pfx -cfgcertfile mySignedConfigurationCertificate.sig.xml
```

Step 6. Verify the new certificate envelope mySignedConfigurationCertificate.sig.xml

(Can be done by INTERCOPE or customer)

```
java -jar icopesecurity.jar -verifycertenvelope -cfgcertfile
mySignedConfigurationCertificate.sig.xml
```

6. Testing of file signing

Step 1. Test file signing

(Can be done by INTERCOPE or customer)

```
java -jar icopesecurity.jar -genfilesignaturecca -kstype JCECCARACFKS -
cfgcertfile mySignedConfigurationCertificate.sig.xml -file test.txt
```

Step 2. Verify file signature

(Can be done by INTERCOPE or customer)

```
java -jar icopesecurity.jar -verifyfilesignature -datafile test.txt -
signaturefile test.txt.sig.xml -cfgcertfile
mySignedConfigurationCertificate.sig.xml
```

3.10.8 Reencryption of BSL with a new MEK

Step 1: Generate new encrypted MEK by using IBMJCECCA [Output: mymek.enc]

(Must be done by our customer)

```
java -jar icopesecurity.jar -encryptmekcca -cfgcertfile
mySignedConfigurationCertificate.sig.xml -emekfile mymek.enc
```

Step 2: Extract current BaseSecretList.enc, mek.enc and SignedConfigurationCertificate.sig.xml from security.zip [Output: BaseSecretList.enc, mek.enc, SignedConfigurationCertificate.sig.xml]

(Must be done by our customer)

```
java -jar icopesecurity.jar -extractseczipentries -securityzipfile
security.zip -zipentrynames BaseSecretList.enc mek.enc
SignedConfigurationCertificate.sig.xml
```

Step 3: reenypt the BSL [input/output: BaseSecretList.enc, output: BaseSecretList.xml.sig.xml]

(Must be done by our customer)

```
java -jar icopesecurity.jar -reencryptbslcca -kstype JCECCARACFKS -
encbslfile BaseSecretList.enc -emekfile mek.enc -cfgcertfile
SignedConfigurationCertificate.sig.xml -newemekfile mymek.enc -
newcfgcertfile mySignedConfigurationCertificate.sig.xml
```

Step 4:

rename

a) mymek.enc -> mek.enc

b) mySignedConfigurationCertificate.sig.xml -> SignedConfigurationCertificate.sig.xml

Step 5: insert mek.enc, BaseSecretList.enc, BaseSecretList.xml.sig.xml, and SignedConfigurationCertificate.sig.xml to security.zip [input/output: security.zip]

```
java -jar icopesecurity.jar -insertseczipentries -securityzipfile
security.zip -files mek.enc BaseSecretList.enc
BaseSecretList.xml.sig.xml SignedConfigurationCertificate.sig.xml
```

Step 6: verify the reencrypted BSL

```
java -jar icopesecurity.jar -testbsl -securityzipfile security.zip
```

3.10.9 Reinitialization of BSL

Step 1: Extract current BaseSecretList.enc, BaseSecretList.xml.sig.xml, mek.enc and SignedConfigurationCertificate.sig.xml from security.zip [Output: BaseSecretList.enc, BaseSecretList.xml.sig.xml, mek.enc, SignedConfigurationCertificate.sig.xml]

(Must be done by our customer)

```
java -jar icopesecurity.jar -extractseczipentries -securityzipfile
security.zip -zipentrynames BaseSecretList.enc
BaseSecretList.xml.sig.xml mek.enc
SignedConfigurationCertificate.sig.xml
```

Step 2: Reinit. BSL by using IBMJCECCA

[Input/Output: BaseSecretList.enc/BaseSecretList.xml.sig.xml],

```
java -jar icopesecurity.jar -reinitializeencryptbslcca -encbslfile
BaseSecretList.enc -bslsigfile BaseSecretList.xml.sig.xml -emekfile
mek.enc -cfgcertfile SignedConfigurationCertificate.sig.xml -comment
"Reinit by John Doe" -reinitencryptionkeylist
-kstype JCECCARACFKS
```

Step 3: insert BaseSecretList.enc, and BaseSecretList.xml.sig.xml to security.zip again [input/output: security.zip]

```
java -jar icopesecurity.jar -insertseczipentries -securityzipfile
security.zip -files BaseSecretList.enc BaseSecretList.xml.sig.xml
```

Step 4: verify the reinitialized BSL

```
java -jar icopesecurity.jar -testbsl -securityzipfile security.zip
```

For all options of icopesecurity that need the information about custom's private key, there are corresponding IBMJCECCA-options (with subfix 'cca'):

-showkeystoreinfocca	**Show details about the given JCE keystore by using IBM Common Cryptographic Architecture (CCA) interface**
-gencertenvelopecca	Generate the INTERCOPE certificate envelope file for IBM Common Cryptographic Architecture (CCA) Environment
-genfilesignaturecca	Generate digital signature of the given file by using IBM Common Cryptographic Architecture (CCA) interface
-generatemekcca	Generate Master Encryption Key (MEK) by using IBM Common Cryptographic Architecture (CCA) interface
-encryptmekcca	Encrypt Master Encryption Key (MEK) by using IBM Common Cryptographic Architecture (CCA) interface
-reencryptbslcca	Reencrypt Base Secret List (BSL) by using IBM Common Cryptographic Architecture (CCA) interface
-reinitializeencryptbslcca	Reinitialize the Encrypted Base Secret List (E-BSL) by using IBM Common Cryptographic Architecture (CCA) interface
-testracfcca	Testing of RACF/CCA environment
-renewcertificatecca	Upgrade INTERCOPE security ZIP to new certificate by using IBM Common Cryptographic Architecture (CCA) interface (since version 1.0.88)

To obtain the private key information the following IBMJCECCA-specific option parameters must be specified:

```
[-kstype <type of keystore supported by IBMJCECCA e.g. JCECCARACFKS>
[-ksprovider <keystore provider (optional, default 'IBMJCECCA')>
[-ringownerid <ID of user that owns the RACF keyring>]
[-ringid <name of the RACF keyring>]
[-keyalias <alias of the key itself>]
```

3.11 Additional Options

3.11.1 Verify JCE files

To verify if 'JCE Unlimited Strength Jurisdiction Policy' is activated, check the `JAVA_HOME` of the used profile (echo `$JAVA_HOME`). The respective policy files (`local_policy.jar` and `US_export_policy.jar`) are usually located in `JAVA_HOME/lib/security`
you can open the jar file and check. the relevant line is

```
// There is no restriction to any algorithms. permission
javax.crypto.CryptoAllPermission;
```

For further information, please refer to

https://www.ibm.com/support/knowledgecenter/en/linuxonibm/liaag/wascrypt/l0wscry00_updatejcepolicyfile.htm
https://developer.ibm.com/answers/questions/178395/how-do-i-install-the-unrestricted-policy-files-in.html

3.11.2 Verify JAVA Settings With icopesecurity.jar Before Creating security.zip

icopesecurity.jar offers the possibility to check, whether the profile contains the necessary Java settings. Option `'-testcipher'` can be used to check whether the cipher algorithm defined by the custom certificate is supported by the JVM or not.

The cipher-tester generates a random string 'TEST_....' and encrypts this string using the custom private key. The encrypted string will be decrypted by using the public key specified in the X509 data found in configuration certificate (`SignedConfigurationCertificate.sig.xml` or `security.zip`).
The configuration certificate will also be verified.

Parameter list for the option `-testcipher`

-certtype	Type of private key file: must be one of PFX,P12,PEM or JKS
-keyfile	private key file name
-certpassword	password of the private key file, only needed in case of PFX,P12 or JKS
-certalias	alias of the private key file, only needed in case of PFX,P12 or JKS
-cfgcertfile	cert. envelope file name (not needed if -securityzipfile is given)
-securityzipfile	INTERCOPE security ZIP file which contains the Cert. Envelope file (not needed if -cfgcertfile is given)
-notverifycfgcert	flag indicates whether the verification of the cert. envelope file should be omitted or not (optional)

Examples:

```
-testcipher -certtype PEM -keyfile myprivatekey.pem -cfgcertfile
SignedConfigurationCertificate.sig.xml
```

```
-testcipher -certtype PEM -keyfile myprivatekey.pem -securityzipfile
security.zip
```

```
-testcipher -certtype PFX -keyfile mycert.pfx -certpassword mypwd -
certalias myalias -securityzipfile security.zip -notverifycfgcert
```

Example of test run:

```
java -jar icopesecurity.jar -testcipher -certtype PEM -keyfile
devprivatekey.pem -securityzipfile security.zip

Try to encrypt the test string 'TEST_APkebfFNgQDEtWAro4tQGGjmg'...

Encryption done, result(base64)
'uzvjsP0XLqRIVz29g93NYa60.....8EQ8AF2Q=='.

Try to decrypt the base64 encrypted string
'uzvjsP0XLqRIVz29g93NYa60.....8EQ8AF2Q=='...

Decryption done (OK), result 'TEST_APkebfFNgQDEtWAro4tQGGjmg'.
```

3.11.3 Verify Signed Configuration Certificate

To check the validity Period of the signed certificate, the following check can be run:

```
-verifycertenvelope -cfgcertfile SignedConfigurationCertificate.sig.xml
```

Example:

```
java -jar icopesecurity.jar  -verifycertenvelope -cfgcertfile
SignedConfigurationCertificate.sig.xml

INTERCOPE Certificate Envelope file
'/opt/box/bs_temp/sit_prod/SignedConfigurationCertificate.sig.xml'
successfully verified.
(Period Of Validity: Certificate Envelope[Not before: not given, Not on
or after: not given], Certificate[Not before: '2018-08-17T12:04:30Z',
Not after: '2019-09-21T12:04:30Z'], INTERCOPE Certificate[Not before:
'2016-10-19T14:17:28Z', Not after: '2020-01-19T14:17:28Z'])
```

3.11.4 Show Certificate Information

To obtain details about a given Certificate, the **INTERCOPE Security Tool** `icopesecurity.jar` can be run with the option

```
java -jar icopesecurity.jar -showcertinfo
```

-certtype	Type of Certificate to be used for encryption. Must be one of PFX, P12, PEM or JKS.
-certpassword	Certificate password. Only needed in case of PFX, P12 or JKS.
-certalias	Certificate alias. Only needed in case of PFX, P12 or JKS.
-certfile	Name of Certificate.

Examples:

```
java -jar icopesecurity.jar -showcertinfo -certtype PEM -certfile
mypublicx509certificate.pem

java -jar icopesecurity.jar -showcertinfo -certtype PFX -certpassword
mypwd -certalias myalias -certfile mypfxcert.pfx
```

3.11.5 Verify INTERCOPE .jar Files

INTERCOPE .jar files can be verified by running the **INTERCOPE Security Tool** `icopesecurity.jar` with the option

```
java -jar icopesecurity.jar –verifyicopejars
```

-jarfiles	List of INTERCOPE jar-files that shall be verified.

Example:

```
java -jar icopesecurity.jar -verifyicopejars -jarfiles mpoapi.jar
mpowi.jar
```

3.11.6 Verify BOX .war Files

BOX .war files can be verified by running the **INTERCOPE Security Tool** icopesecurity.jar with the option

```
java -jar icopesecurity.jar -verifyboxmhwar
```

-verifyboxmhwar	Verify the box-mh.war issued by INTERCOPE

Examples:

```
java -jar icopesecurity.jar -verifyboxmhwar -warfile box-mh.war

java -jar icopesecurity.jar -verifyboxmhwar -warfile box-mh.war -
securityzipfile security.zip
```

3.11.7 Verify the Content of Given Base Secret List (BSL)

This option should be used to verify and test the content of the given base secret list (BSL). The newly created security.zip is checked against all necessary components, such as MEK.enc and BSL. All keys and the hashing algorithms found in BSL will be tested against some test data.

```
java -jar icopesecurity.jar -testbsl
```

-testbsl	Verify and test the content of the BSL

Example

```
java -jar icopesecurity.jar -testbsl -securityzipfile security.zip
```

3.11.8 SSL JDBC Server Connection

Using a respective replacement token, the following line may be added to the replace.rpl file :

```
DB_DATABASE_T4URL=jdbc:db2://<db2 server address>:<port>/<database
name/location>:sslConnection=true;sslTrustStoreLocation=/opt/box/server/
SSL/JKS/db2.jks;sslTrustStorePassword=PASWD;
```

3 parameters are used to establish the SSL connection

Parameter	Value	Explanation
`sslConnection`	`true/false`	Sets connection as SSL
`sslTrustStoreLocation`	`path`	Path and file name to keystore, which must contain all necessary certificates
`sslTrustStorePassword`	`password`	Key store password

4 Software Integrity

BOX features an embedded function to perform regular software integrity checks, providing a detective control against unexpected modification to the operational software.

All BOX binaries are digitally signed with INTERCOPE's X509 signature. The signatures of the box-mh.war and jar files can be verified using the standard java tool, 'jar signer', or with the INTERCOPE Security Administration Tool (`icopesecurity.jar` file, see above, 3).

The BOX server code contains signature files for all server binaries (including third party libraries), and the digital signatures are verified at module start-up.

Any resultant mismatch between the expected and received Message Digest of the Digital Signature will trigger a BOX Start up Failure.

The integrity of production system software is therefore assured upon start-up of the application. Furthermore, and to conform with the Control implementation guidelines, the BOX Server is configured to automatically run the software integrity check once per day. The starting hour for the daily check can be configured with the parameter `CODE_SECURITY_CHECK_STARTING_HOUR` in section `[SECURITY]`. Default value for this parameter is 01:00am.

It is also possible to run the Software Integrity as a separate process using the `mpo_secchk` tool.

In addition to the embedded software integrity check, BOX supports the installation of third-party File Integrity Monitoring (FIM) tools. Therefore, as a further security measure, customers may deploy a FIM of their preference to validate the integrity of application software.

4.1 How to Verify Software Integrity

The verification of BOX software integrity is done by verifying Certificates and Signatures with the **INTERCOPE Security Tool** `icopesecurity.jar` as described above in chapter 3.

4.2 Security Check of Intercope JAVA Code

Due to the JAVA code being verified during start-up and with automatic regular code security checks Intercope jars are now signed. This applies to all Intercope jars delivered with the new release V3R22 and is planned for all third-party jars. All jars must be configured in the JAVA Classpath in parameter JAVA_OPTIONS.

High priority jars, which must be present, are

- → `MXTools.jar`
- → `RNITools.jar`
- → `icopesecurity.jar`
- → `icopeFinXml.jar`
- → `msgval-XXX.jar`
- → `box-servtools-XXX.jar`

IMPORTANT

Please be aware, that the above jars require additional jars due to dependencies, such as rnitoolkit.jar, xom.jar, fontbox.jar and pdfbox.jar.

Please review the configuration of JAVA_OPTIONS, keep, apart from the high priority jars, all jars already configured and check the CLASSPATH for any obsolete old jars causing error messages during the server start-up.

Exemplary JAVA_OPTIONS string

```
SRV_JAVA_CLASSPATH=/opt/box/server/security/lib/icopesecurity.jar:/opt/b
ox/server/config/security.zip:/opt/box/server/extlib/openssl:/opt/box/se
rver/extlib/xmllint:/opt/box/server/extlib/xmlcatalog:/opt/box/server/ex
tlib/xsltproc:/opt/ibm/db2/V10.1/java/db2jcc4.jar:/opt/ibm/db2/V10.1/jav
a/db2jcc_license_cu.jar:/opt/ibm/db2/V10.1/java/db2jcc4.jar:/opt/box/ser
ver/icjtools/commons-pool.jar:/opt/box/server/icjtools/commons-
dbcp.jar:/opt/box/server/icjtools/msgval.jar:/opt/box/server/icjtools/bo
x-
servtools.jar:/opt/box/server/config/val:/opt/box/server/icjtools/log4j-
core.jar:/opt/box/server/icjtools/log4j-
api.jar:/opt/box/server/icjtools/xom.jar:/opt/box/server/icjtools/log4j.
jar:/opt/box/server/icjtools/simple-
jndi.jar:/opt/box/server/config/printing:/opt/box/server/icjtools/icopeF
inXml.jar:/opt/box/server/icjtools/commons-
logging.jar:/opt/box/server/icjtools/pdfbox.jar:/opt/box/server/icjtools
/fontbox.jar:/opt/box/server/icjtools/jempbox.jar:/opt/box/server/icjtoo
ls/xmpbox.jar/:/opt/box/server/config/:/opt/box/server/icjtools/
:/opt/box/server/icjtools/commons-
io.jar:/opt/box/server/icjtools/commons-
compress.jar:/opt/box/server/icjtools/bcprov-
jdk15on.jar:/opt/box/server/icjtools/IC_W2X.jar:/opt/box/server/icjtools
/joda-
time.jar:/opt/box/server/icjtools/jtp.jar:/opt/box/server/icjtools/log4j
-1.2-api.jar:/opt/box/server/icjtools/log4j-slf4j-
impl.jar:/opt/box/server/icjtools/MXTools.jar:/opt/box/server/icjtools/r
nitoolkit.jar:/opt/box/server/icjtools/RNITools.jar:/opt/box/server/icjt
ools/saxon-he.jar:/opt/box/server/icjtools/slf4j-api.jar -Xms256m -
Xmx724m -Dcom.ibm.crypto.provider.DoRSATypeChecking=false
```

5 Configuration Integrity

All BOX configuration files must be signed to ensure that all modifications are made by an authorized user.

5.1 Client Configuration

The BOX delivery includes a compressed Security file (`security.zip`) containing the following elements:

Encrypted Base Secret List (eBSL):

The Base Secret List (BSL) contains a list of symmetric `keys` and their associated information e.g. the encryption algorithm. The BSL is deployed with the product software in encrypted format.

In order to secure passwords and to avoid storage of plain text passwords in the BSL the Salted Password Hashing technique is used.

Encrypted Master Encryption Key File (eMEK file):

The `key` for the encrypted BSL is the MEK, which is an AES-256-bit `cipher`. Its purpose is to ensure secure access to other sensitive data.

Signature Files for all relevant client configuration files:

All changes in client configuration files made by the customer must be digitally signed; this task is done using the INTERCOPE Security Tool. A new Signature File is generated, verified and inserted into the `security.zip` file.

Please note, the original names of the Signature Files in `security.zip` (files with the extension `.sig.xml`) must not be changed!

Assuming that the name of the file `configuration.properties` has been changed to `myconfiguration.properties` (due to whatever reason), the name of the Signature File still must be **configuration.properties.sig.xml**.

When the system is re-started the `security.zip` file is read and searched for the required Signature Files.

Configuration changes will only take effect when a corresponding Signature File is found and in case of missing signatures the system will generate corresponding alerts (see below 5.3.1).

Signed Configuration Certificate:

The Signed Configuration Certificate (SCC) is simply a piece of information that contains the customer X.509 certificate and its digital signature signed by INTERCOPE. The SSC will normally be used by BOX during the information verification process to ensure that the owner of the encrypted information is authorized by INTERCOPE, in addition to the normal digital signature verification.

Web Application configuration file **SecurityCfgDefinitions.xml**.

This configuration file specifies the behaviour in case of security issues (see below, 5.3.1)

5.1.1 A Few Remarks on security.zip

The purpose of the compressed file `security.zip` is:

- At system delivery to provide the customer with all Security Implementation relevant items (the Encrypted Base Secret List [eBSL], the Encrypted Master Encryption Key File [eMEK file], Signature Files for all relevant client configuration files, the Signed Configuration Certificate and the Web Application configuration file SecurityCfgDefinitions.xml), see above, 5.1.
- Within a running system `security.zip` is the default container for the above-mentioned items, i.e. the location in which these items will be searched for.

The delivered default `security.zip` must be modified in order to generate the installation- and customer-specific zip file.

For the BOX Client, the `security.zip` must contain the signature files for ‘static’ Web Application configuration files, e.g. generic schema xml files (signed by INTERCOPE) and the signature files for installation-specific configuration files (signed by customer).

The ‘static’ Web Application configuration files (signed by INTERCOPE) are delivered in a default `security.zip` that must be updated with customer-specific eMEK, encrypted eBSL, SignedCustomerCertificate.sig.xml and the signature files for all customized Web Application configuration files (`configuration.properties, log4j2.xml, mpo.xml, customized schema files`).

Each time a new `.war` file is deployed, the default `security.zip` (containing the current INTERCOPE signature files) delivered along with it must be updated with the installation-specific elements (`eMEK, enc.BSL, SignedConfigurationCertificate.xml`) and the configuration signature files.

For the modification of `security.zip` INTERCOPE provides necessary tools (icopeSecurity.jar, see above 3) and scripts (mkBSL.sh, signClient.sh, see 5.4.1 and 5.4.2.)

For BOX Server and Java tools the `security.zip` does not have to contain the signature files for configuration files (mposerver.cfg, api-configuration.properties…).

Therefore, the scripts to sign client configuration files and server configuration files work differently.

5.1.1.1 JVM Option to Specify Path and Name of security.zip File

A new JVM-option 'com.intercope.security.SecurityZIP' has been made available, which can be used to specify the path and name of the 'security.zip' file to be used.

By default, the BOX Client tries to find the 'security.zip' in

a) classpath, if not found then
b) WEB-INF directory

This process can be overwritten by using this new JVM-option:

```
-com.intercope.security.SecurityZIP=FullPathAndNameOfSecurityZIPToBeUsed
```

In this case the full path to the security.zip must be given, e.g.

```
-Dcom.intercope.security.SecurityZIP=d:\security\mysecurity.zip
-Dcom.intercope.security.SecurityZIP=NameOfSecurityZIPToBeUsed
```

In this case the given file must be found in `CLASSPATH`, e.g.

```
-Dcom.intercope.security.SecurityZIP=mysecurity.zip
```

[where `CLASSPATH=d:\security`]

5.2 Server Configuration

The Server configuration files are located in the directory **MPO_BASE/config**.

Changes in these configuration files must also be digitally signed. The signing is done using the **INTERCOPE Security Tool** `icopesecurity.jar` (see 0).

The Signature Files must be located in the same directory as the configuration files.

Configuration changes will only take effect when a corresponding Signature File is found and in case of missing signatures the system will generate corresponding alerts (see below 5.3.2).

Other Server related configuration changes are stored in the BOX database.

The configuration integrity in this case is ensured by the database integrity (see 6), i.e. the changes in the configuration data is encrypted by means of salted hashes before being stored.

When configuration data is read, the hash value is compared and in case of any mismatch the system will generate corresponding alerts (see below 5.3.2).

5.3 Behaviour in Case of Security Issues

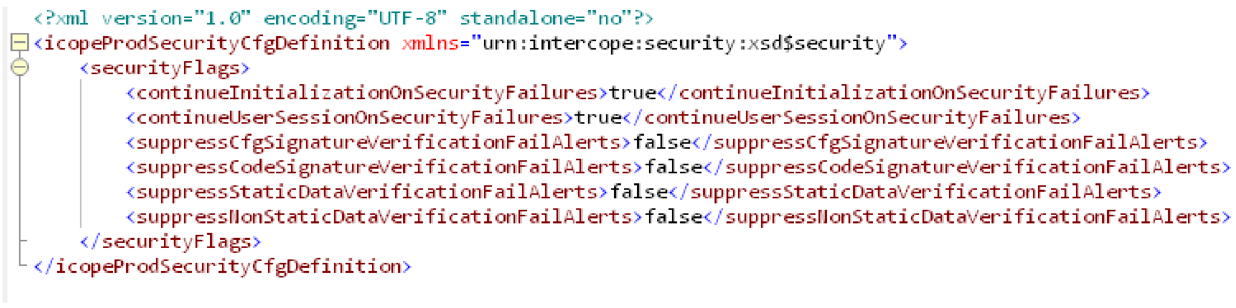
The behaviour on security issues is configured separately for the BOX Server and for the Web Application.

Configurable behaviour options in case of security issues are:

- Whether initialization shall be continued with or not.
- Whether user session shall be continued with or not.
- Whether alerts shall be generated or not in case of failed
  Configuration signature verification
  Code signature verification
  Static data verification
  Non-static data verification

5.3.1 Client Configuration

For the Web Application the configuration is done using the file SecurityCfgDefinitions.xml which is located in the `security.zip` file.



```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<icopeProdSecurityCfgDefinition xmlns="urn:intercope:security:xsd$security">
    <securityFlags>
        <continueInitializationOnSecurityFailures>true</continueInitializationOnSecurityFailures>
        <continueUserSessionOnSecurityFailures>true</continueUserSessionOnSecurityFailures>
        <suppressCfgSignatureVerificationFailAlerts>false</suppressCfgSignatureVerificationFailAlerts>
        <suppressCodeSignatureVerificationFailAlerts>false</suppressCodeSignatureVerificationFailAlerts>
        <suppressStaticDataVerificationFailAlerts>false</suppressStaticDataVerificationFailAlerts>
        <suppressNonStaticDataVerificationFailAlerts>false</suppressNonStaticDataVerificationFailAlerts>
    </securityFlags>
</icopeProdSecurityCfgDefinition>
```

5.3.2 Server Configuration

For the Server the configuration is done in all module configuration files (*.cfg), section `[SECURITY]`.

```
[SECURITY]
SECURITY_ZIP_FILE         security\basesec\security.zip
;BASE_SECRET_LIST          security/basesec
                          /customer_BaseSecretList.enc

;CUSTOMER_CONFIG_CERT      security/certs/
                          customer_SignedConfigurationCertificate.sig.xml

CONTINUE_INIT_ON_SECURITY_FAILURES                             NO
CONTINUE_PROC_ON_SECURITY_FAILURES                             NO
SUPPRESS_CONFIGURATION_SIGNATURE_VERIFICATION_ALERTS           NO
SUPPRESS_CODE_SIGNATURE_VERIFICATION_ALERTS                    NO
SUPPRESS_STATIC_DATA_VERIFICATION_ALERTS                       NO
SUPPRESS_NONSTATIC_DATA_VERIFICATION_ALERTS                    NO
```

There are two ways of configuring the security-relevant parameters in this section:

Either specify name and path to your `security.zip` via parameter `[SECURITY].SECURITY_ZIP_FILE` or specify name and path of the BSL and the Signed Configuration Certificate using the parameters

```
  BASE_SECRET_LIST             config\bsl-mk.enc
  CUSTOMER_CONFIG_CERT         config\devpublicx509certificate.env.xml
```

The default value for `[SECURITY].SECURITY_ZIP_FILE` is 'config/`security.zip`' but we recommend using another directory in order to avoid the BOX Server sourcing the delivered default `security.zip` file.

5.4 Deployment Scripts

In order to simplify the onboarding process and software updating INTERCOPE provides Deployment Scripts. These scripts can be found in the folder ‘security’.

Please note:

All scripts support the usage of a configuration file (sectool.conf). The configuration file must be in the security main folder (.../security/) to take effect. A preconfigured template is provided (../security/sectool.conf.dist) and can be configured to suit the specific environment. To activate the configuration file, it must be renamed in sectool.conf.

The provided deployment scripts mkBSL.sh, renewCert.sh, signclient.sh and signconfig.sh not only support Linux, but also CCA on zOS. A temporary java.security.zos is provided to configure JAVA for CCA, hence the jar utility is now supported on zOS.

5.4.1 mkBSL.sh:

Script mkBSL.sh is used for generating and encrypting a new MEK, for re-encrypting and re-initializing the BSL and for adding all customer-specific components to a new `security.zip` that can be used for the installation.

This script produces two zip files in the main folder:

`security.zip`	Contains also ‘static’ configuration file signatures for the Web Application.
`serversec.zip`	Contains only elements absolutely necessary for the BOX Server.

For the BOX Server either one zip file can be used.

The script copies the following files into the directory basesec:

```
  mek.enc (the new MEK)
  BaseSecretList.enc (the encrypted BSL)
  BaseSecretList.xml.sig.xml (signature file for the encrypted BSL)
  ProductSecurityCfgDefinitions.xml (
  ProductSecurityCfgDefinitions.xml.sig.xml
  SignedConfigurationCertificate.sig.xml
```

Do not change or delete these files. They are basic for your installation-specific `security.zip`. They are also used for other scripts (e.g. signclient.sh).

```
Usage:
./mkBSL.sh -s <secfile> -i <signconf> -p <certfile> -k <privkey>
```

where:

-s <secfile>	Path including filename of the security.zip file.
-p <certfile>	Path including filename of the Customer Certificate.
-I <signconf>	Path including filename of the Customer Certificate signed by Intercope.
-k <privkey>	Path including filename of the Private key of the customer certificate.
IBM Common Cryptographic Architecture (CCA) mode only parameters	
-Z	switch on the CCA mode
-C <kstype>	type of key store supported by IBMJCECCA e.g. JCECCARACFKS

Sample output:

```
./mkBSL.sh security.zip mypublicx509certificate.pem
SignedConfigurationCertificate.sig.xml myprivatekey.pem
Checking security.zip - passed
Checking Certificate - passed
Checking Intercope signed Certificate - passed
Checking key - passed
Please enter a description for the new BSL: created in April 2018 by ABC
MEK successfully encrypted.

BSL successfully re-encrypted.
INTERCOPE File Signature 'BaseSecretList.enc.sig.xml' successfully
created.

Encrypted BSL successfully reinitialized.

New security.zip created
Old renamed to security_180328-102830.zip
New serversec.zip created
```

5.4.1.1 mkBSL.sh on CCA/zOS

mkBSL.sh supports the parameters -Z and -C to reach the zOS KeyStore.

```
Usage
./mkBSL.sh -s <secfile> -i <signconf> -Z -C <kstype>
```

5.4.2 signClient.sh:

This script can in one run perform all the steps required at the time of deployment or when updates are deployed. The script can be called upon from a different location than `.../security`.

It is used for generating or updating the `security.zip` file, for automatically adding this `security.zip` file to the WAR or EAR archive files.

Usage:

```
./signclient.sh -m <mode> -s <security.zip location> -w <war file
location> -e <ear file location> -h
```

where:

-m	<mode>	CrtSZ CrtWAR CrtEAR UpdSZ UpdWAR UpdEAR	Create security.zip Create war file Create ear file Update security.zip Update war file Update ear file
-h			usage/help
-h	<mode>		usage parameters for this specific mode
-c	<path>		path to the adopted configuration files
-i	<Basesec> location		include configuration in war or ear file
-s	<security.zip>		location of the security.zip file, default is in the /opt/box22/csp/security/client /cltconf directory
-w	<war file location>		location of the war file
-e	<ear file location>		location of the ear file
-r	<replace.rpl location>		location of the replacement file if not part of the config directory.
-x	<extract security.zip>		extract security.zip only from new war/ear file.
-b	< basesec>		basesec directory
-o	<output directory>		output directory for the generated security.zip only in CrtSZ and UpdSZ mode
-p	< public Certificate>		path and name of the public Certificate
-k	< private key>		path and name of the private key

Modes

CrtSZ	-h	Mode to Create a new customized security.zip
CrtWAR	-h	Mode to Create a new customized security.zip in a war archive
CrtEAR	-h	Mode to Create a new customized security.zip in an ear archive
UpdSZ	-h	Mode to update a customized security.zip
UpdWAR	-h	Mode to update a customized security.zip
UpdEAR	-h	Mode to update a customized security.zip

Please note:

In CrtSZ or UpdSZ mode, the security.zip given as a parameter will not be changed, but copied to the destination folder, consequently changing a copy of the security.zip file. Deleting an adapted configuration file, for which a signature file is already contained in the security.zip file leads to a security error. In this case the original security.zip file taken from the client war file must be reinstated.

In CrtWAR, UpdWAR, CrtEAR and UpdEAR mode used with the parameter '- i' the updated security.zip as well as the configuration parameters are copied back to the WAR/EAR file. If the parameter '- i' is not given, the updated security.zip will be copied to the same directory as the updated configuration files.

Example:

To create `customer_security.zip` for BOX Web Application, run with -`m CrtSZ`

```
./signclient.sh -mCrtSZ
```

Sample output:

```
./signclient.sh -m CrtSZ
Checking public Certificate - passed
Checking private Certificate - passed
using client/cltconf/security.zip
INTERCOPE File Signature
'/opt/box22/csp/security/client/cltconf/configuration.properties.sig.xml
' successfully created in
'/opt/box22/csp/security/client/cltconf/security.zip'.
INTERCOPE File Signature
'/opt/box22/csp/security/client/cltconf/log4j2.xml.sig.xml' successfully
created in '/opt/box22/csp/security/client/cltconf/security.zip'.
INTERCOPE File Signature
'/opt/box22/csp/security/client/cltconf/log4j.properties.sig.xml'
successfully created in
'/opt/box22/csp/security/client/cltconf/security.zip'.
INTERCOPE File Signature
'/opt/box22/csp/security/client/cltconf/mpo.xml.sig.xml' successfully
created in '/opt/box22/csp/security/client/cltconf/security.zip'.
INTERCOPE File Signature
'/opt/box22/csp/security/client/cltconf/replace.rpl.sig.xml'
successfully created in
'/opt/box22/csp/security/client/cltconf/security.zip'.
/opt/box22/csp/security/client/cltconf/security.zip not part of the
signature list
Finished: added Signatures to the client/cltconf/security.zip
updating: ProductSecurityCfgDefinitions.xml.sig.xml
        zip warning: Local Entry CRC does not match CD:
ProductSecurityCfgDefinitions.xml.sig.xml
 (deflated 38%)
```

```
updating: ProductSecurityCfgDefinitions.xml
        zip warning: Local Entry CRC does not match CD:
ProductSecurityCfgDefinitions.xml
 (deflated 54%)
updating: BaseSecretList.enc
        zip warning: Local Entry CRC does not match CD:
BaseSecretList.enc
 (deflated 24%)
updating: SignedConfigurationCertificate.sig.xml
        zip warning: Local Entry CRC does not match CD:
SignedConfigurationCertificate.sig.xml
 (deflated 38%)
updating: BaseSecretList.xml.sig.xml
        zip warning: Local Entry CRC does not match CD:
BaseSecretList.xml.sig.xml
 (deflated 38%)
updating: mek.enc
        zip warning: Local Entry CRC does not match CD: mek.enc
 (deflated 16%)
added Base Security files to client/cltconf/security.zip
cp: `client/cltconf/security.zip' and
`/opt/box22/csp/security/client/cltconf/security.zip' are the same file
copied new security.zip to /opt/box22/csp/security/client/cltconf
finished
```

5.4.2.1 signclient on CCA/zOS

signclient.sh supports the parameters -Z and -C to reach the zOS KeyStore.

```
Usage
./signclient.sh -m <mode> -s <security.zip location> -w <war file
location> -Z -C <kstype>
```

5.4.3 signconfig.sh:

This script can be used for signing configuration files for BOX Server and Java Tools.

Run script signconfig.sh with path to files to be signed, enter file name for single file or * for all files. The script can be called upon from a different location than `.../security`.

Usage

```
./signconfig.sh [-p <public key>][-k <private key>][-s <sccfile>] -o
<output directory> -f <file> <file> ...
```

SignedConfigurationCertificate file <sccfile> is only used to verify that these certificates belong together.

-p	public key	public key	Path including filename of the public key of the Customer Certificate
-k	private key	private key	Path including filename of the Private key of the Customer certificate
-s	sccfile	SignedConfigurationCertificate file	(optional/mandatory in for CCA) Path including filename of the Customer Certificate signed by Intercope
-o	output	Path to output	all signature files will be

	directory	directory	created in that directory
-f	file/files	Path and files to be signed	List of files to be signed
IBM Common Cryptographic Architecture (CCA) mode only parameters			
-Z		switch on the CCA mode	
-C <kstype>		type of key store supported by IBMJCECCA e.g. JCECCARACFKS	

You can use a configuration file sectool.conf located in the main directory of the security tool to overwrite the default parameters listed at the top of signclient.sh.

Example

```
./signconfig.sh -f <path_to/mposerver.cfg
```

or

```
./signconfig.sh -f server/config/
```

Sample output:

```
./signconfig.sh -f ../server/config/mposerver.cfg
Checking java - passed
Checking Certificate - ./certs/devpublicx509certificate.pem - passed
Checking key - ./private/devprivatekey.pem - passed
certificate pair test: passed
used certificate:
        Subject: C=DE, ST=Hamburg, L=Hamburg, O=Development, INTERCOPE
GmbH, OU=BOX Development, CN=Dev
            Not After : Sep 27 09:33:04 2018 GMT
INTERCOPE File Signature
'/opt/box/security/../server/config/mposerver.cfg.sig.xml' successfully
created.
created signature file ../server/config/mposerver.cfg.sig.xml
```

When using `./signconfig.sh -f server/config/`, make sure the server config directory contains the signature files for all files, cfg files, xslt files, *.tpl files.

5.4.3.1 signconfig.sh on CCA/zOS

`signconfig.sh` supports the parameters -Z and -C to reach the zOS KeyStore.

```
Usage
./signconfig.sh -s <secfile> -i <signconf> -Z -C <kstype>
```

5.4.4 mpoenc.sh

This script creates password - and laukey hashes

-m	<mode> mandatory	LAU DB	Mode to use
-s	<security.zip> mandatory	security.zip	location of the security.zip file, default is in the .../security/client/cltconf directory
-p	"<password>"	Password	

	(optional)		
-l	"<laukey>" (optional)	Laukey	

Example

```
./mpoenc.sh -m DB -s security.zip
Checking security.zip - security.zip - passed
Enter Password:
Retype Password again:
Encrypted Password: '6886330FD1B89A954335527FBC82F2F7' successfully
created.
```

5.4.5 makepwd.sh

This script can be used for encrypting Database user and UPM user passwords.

Example:

```
./makepwd.sh <secfile>
```

Sample output:

```
box@icim02:/opt/box22/csp/security> ./makepwd.sh customer_security.zip
Checking security.zip - passed
Enter Password:
Retype Password again:
Encrypted Password: '65D63A224195FF57EF177259ED39A524' successfully
created.
```

5.4.6 renewCert.sh

Once a certificate is expired and replaced with a newly issued one, it is necessary to replace the certificate and update the Base Security List (BSL). The script to do this expects a new public and private certificate as well as the signed files '/SignedConfigurationCertificate.sig.xml' created for the new certificates.
The script will extract all files from 'Basesec' in a temporary security.zip (tempsecurity.zip) archive, update these files by using the icopesecurity.jar and create a new mek. With the new mek, the script will finally re-encrypt the BSL

```
Usage:
-------

./renewCert.sh -p <certfile> -c <signconf> -k <privkey>
```

-c <path/signconf>	Path including filename of the new Customer Certificate signed by Intercope
-p <certfile>	Path including filename of the Customer Certificate.
-k <privkey>	Path including filename of the Private key of the customer certificate.
IBM Common Cryptographic Architecture (CCA) mode only parameters	
-Z	switch on the CCA mode
-C <kstype>	type of key store supported by IBMJCECCA e.g. JCECCARACFKS

```
Example
./renewCert.sh -p certs.new/scm-pub.pem -c
certs.new/SignedConfigurationCertificate.sig.xml -k private.new/scm-
priv.pem
Checking unzip toolpassed
Checking zip toolpassed
Checking javapassed
Checking Certificate - passed
Checking Intercope signed Certificate - passed
Checking private key passed
certificate pair test: passed
SCC file test: passed
used certificate: <certificate subject>
     Not After : Feb 20 14:59:06 2028 GMT
basesec security repository found.
Checking for BSL: passed
Checking for BSL Signature: passed
Checking for eMEK: passed
Checking for Signed Configuration Certificate: passed
creating temporary security.zip  in /opt/box/security from basesec:
passed
'/opt/box/security/tempsecurity.zip' successfully upgraded.
renewing MEK and BSL: passed
Archive:  tempsecurity.zip
  inflating: basesec/SignedConfigurationCertificate.sig.xml
  inflating: basesec/ProductSecurityCfgDefinitions.xml.sig.xml
  inflating: basesec/BaseSecretList.enc
  inflating: basesec/mek.enc
  inflating: basesec/BaseSecretList.xml.sig.xml
  inflating: basesec/ProductSecurityCfgDefinitions.xml
extracting tempsecurity.zip to basesec: passed
New /serversec.zip created
```

5.4.6.1 renewCert.sh on CCA/zOS

renewCert.sh supports the parameters -Z and -C to reach the zOS keystore.

```
Usage
./ renewCert.sh -s <secfile> -i <signconf> -Z -C <kstype>
```

5.5 Configuration of WEB Application Server

To configure the WEB Application Server, stop the BOX application in the Application Server.

Make sure that all configuration files have been updated and signed and that the customer-specific `security.zip` has been either added to classpath or inserted in the war/ear file.

Deploy your **box-mh.war** file in your Application Server and restart the BOX application.

By default, the BOX Client tries to find the customer-specific '`security.zip`' in the classpath or –if not found – the WEB-INF directory.

This default behaviour can be overwritten by using a (new) JVM-option 'com.intercope.security.SecurityZIP' that can be used to specify the path and name of '`security.zip`' to be used.

There are two options: Either you give the full path to the `security.zip` or the `security.zip` file must be found in the classpath.

Examples:

```
-Dcom.intercope.security.SecurityZIP=FullPathAndNameOfSecurityZIPToBeUsed
```

In this case the full path to the `security.zip` must be given, e.g.:

```
-Dcom.intercope.security.SecurityZIP=d:\security\mysecurity.zip
```

```
-Dcom.intercope.security.SecurityZIP=NameOfSecurityZIPToBeUsed
```

In this case the given file must be found in classpath, e.g.

```
 -Dcom.intercope.security.SecurityZIP=mysecurity.zip [where
CLASSPATH=d:\security]
```

5.6 Java Tools Configuration

All tool configuration files for BOX Java Tools require to be digitally signed.

For this purpose, the configuration files (e.g. configuration.properties, replace.rpl) must be signed with the **INTERCOPE Security Tool** `icopesecurity.jar` where after the signature files must be stored in the same directory in which the configuration files are located.

Example:

config/configuration.properties
config/configuration.properties.sig.xml

When a tool is used, the option **–sz** must be given.

This parameter specifies the `security.zip` file used for verifying the integrity of the file(s).

Example:

```
./changeset-exporter.sh -sz ../security.zip -rpl ../replace.rpl -api
config/configuration.properties
```

Note: There are two security-related parameters, the above mentioned **–sz** parameter and the **–mek** parameter specifying the file containing the Master Encryption Key (MEK). As the MEK is in the `security.zip` file and the **–sz** parameter **must** be given, the –mek parameter is not needed.

6 Database Integrity

BOX features an embedded function that ensures database integrity and provides a detective control against unexpected modification (insertion/deletion) to records stored within the BOX database.

A two-step approach is adopted utilizing

1) Hash Checker (Encrypted checksums at record level)

2) Gap detection in database records.

6.1 The Hash Checker

The Hash Checker provides a hash function on database row level, which, in general terms, takes an input of arbitrary (although bounded) length and outputs a fixed-length value. Common names for the output of a hash function include hash value, hash, message digest, and digital fingerprint. The maximum number of input and output bits is determined by the design of the hash function.

The hash function is a cryptographic hash function and the Hash Checker's data encryption technique is SHA256 + SALT.

The Hash Checker is a vital part of the **BOX migration procedure** and is delivered as part of release V3R22 and upwards. It is a standalone tool using its own libraries and configuration (configuration.properties).

The Hash Checker has to run, once the database has been successfully migrated, to read database entries on row level, to calculate for each specified table row its respective hash value and to write this value for each row to specific table columns (PHASH and DHASH) created during the migration.

It is imperative, that the Hash Checker is executed straight after the migration!

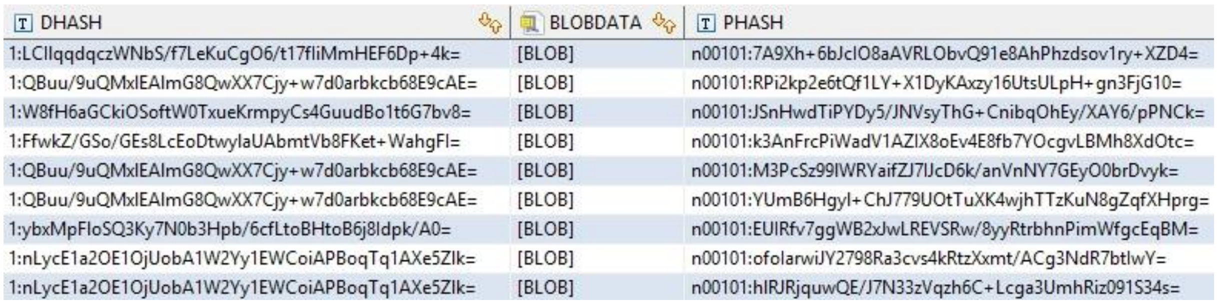
For the calculation of hash values to be successful, the `icopesecurity.jar` is used and contains necessary functions, which are called upon for the creation of the hash values and for writing the respective values to the database.

For every message in a V3R22 (and upwards) productive system these hash-values are automatically calculated and written to the database using the same functions implemented in the `icopesecurity.jar` as the Hash Checker uses.

Please note, that as long as an UPM user can logon to BOX, any user can use the Hash Checker tool. There is no specific need to run the Hash Checker with prefix SYS. In order to add hashes for more than one company, please use the parameter -cpl <prefix> (comma separated list of client prefixes).

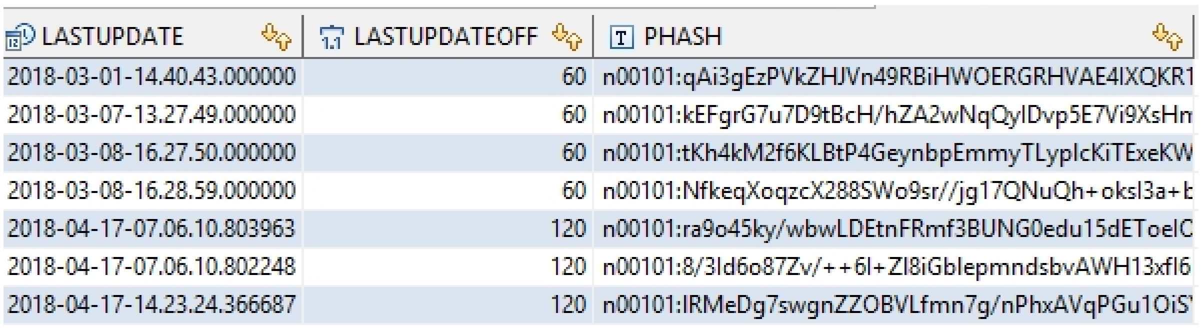
6.1.1 Difference Between DHASH and PHASH

DHASH and PHASH values will be calculated after the start of the HASH Checker. The difference between both hash-values is profound. Whilst the DHASH-value is calculated from the actual message contained in a CLOB or BLOB, the PHASH-value is calculated from the entire row **AND** the DHASH-value. Only certain tables, such as the CVMI contain a message CLOB or BLOB and therefore a DHASH-value



DHASH	BLOBDATA	PHASH
1:LCIlqqdqczWNbS/f7LeKuCgO6/t17fliMmHEF6Dp+4k=	[BLOB]	n00101:7A9Xh+6bJclO8aAVRLObvQ91e8AhPhzdsov1ry+XZD4=
1:QBuu/9uQMxlEAlmG8QwXX7Cjy+w7d0arbkcb68E9cAE=	[BLOB]	n00101:RPi2kp2e6tQf1LY+X1DyKAxzy16UtsULpH+gn3FjG10=
1:W8fH6aGCkiOSoftW0TxueKrmpyCs4GuudBo1t6G7bv8=	[BLOB]	n00101:JSnHwdTiPYDy5/JNVsyThG+CnibqOhEy/XAY6/pPNCk=
1:FfwkZ/GSo/GEs8LcEoDtwylaUAbmtVb8FKet+WahgFl=	[BLOB]	n00101:k3AnFrcPiWadV1AZlX8oEv4E8fb7YOcgvLBMh8XdOtc=
1:QBuu/9uQMxlEAlmG8QwXX7Cjy+w7d0arbkcb68E9cAE=	[BLOB]	n00101:M3PcSz99IWRYaifZJ7lJcD6k/anVnNY7GEyO0brDvyk=
1:QBuu/9uQMxlEAlmG8QwXX7Cjy+w7d0arbkcb68E9cAE=	[BLOB]	n00101:YUmB6Hgyl+ChJ779UOtTuXK4wjhTTzKuN8gZqfXHprg=
1:ybxMpFloSQ3Ky7N0b3Hpb/6cfLtoBHtoB6j8ldpk/A0=	[BLOB]	n00101:EUIRfv7ggWB2xJwLREVSRw/8yyRtrbhnPimWfgcEqBM=
1:nLycE1a2OE1OjUobA1W2Yy1EWCoiAPBoqTq1AXe5ZIk=	[BLOB]	n00101:ofolarwiJY2798Ra3cvs4kRtzXxmt/ACg3NdR7btlwY=
1:nLycE1a2OE1OjUobA1W2Yy1EWCoiAPBoqTq1AXe5ZIk=	[BLOB]	n00101:hIRJRjquwQE/J7N33zVqzh6C+Lcga3UmhRiz091S34s=

The following picture represents a table, here MWH-FIN, showing column ‘PHASH’ and created PHASH-values.



LASTUPDATE	LASTUPDATEOFF	PHASH
2018-03-01-14.40.43.000000	60	n00101:qAi3gEzPVkZHJVn49RBiHWOERGRHVAE4IXQKR1
2018-03-07-13.27.49.000000	60	n00101:kEFgrG7u7D9tBcH/hZA2wNqQylDvp5E7Vi9XsHn
2018-03-08-16.27.50.000000	60	n00101:tKh4kM2f6KLBtP4GeynbpEmmyTLyplcKiTExeKW
2018-03-08-16.28.59.000000	60	n00101:NfkeqXoqzcX288SWo9sr//jg17QNuQh+oksl3a+b
2018-04-17-07.06.10.803963	120	n00101:ra9o45ky/wbwLDEtnFRmf3BUNG0edu15dEToelC
2018-04-17-07.06.10.802248	120	n00101:8/3Id6o87Zv/++6I+ZI8iGblepmndsbvAWH13xfl6
2018-04-17-14.23.24.366687	120	n00101:IRMeDg7swgnZZOBVLfmn7g/nPhxAVqPGu1OiS

Excerpt from MWH-FIN

6.1.2 Using the Hash Checker

The Hash Checker with all dependent files is delivered in a ZIP file, which needs to be extracted in a newly created directory.

After extraction you’ll find the following structure:

- hash-checker.sh (the tool)
- config (folder containing api-configuration.properties, log4j2.xml, replace.properties)
- lib (necessary lib files, e.g. icopesecurity.jar)

To configure the Hash-Checker, the api-configuration.properties file needs to be adapted. Good practice is to copy the provided api-configuration.properties file to configuration.properties and adapt the copied file, if and as required:

- `DB.username=$$R$DB_DATABASE_USER$`
- `DB.password=$$R$DB_DATABASE_ENC_PWD$`
- `DB.URL=$$R$DB_DATABASE_T4URL$`
- `DB.DBSchema=$$R$DB_DATABASE_QUALIFIER$`
- `DB.type=$$R$DB_DATABASE_TYPE$`

The parameter System.VMID=MPOAPIVM00HASHCHECK must not be changed, it is unique for the Hash Checker api instance. For this reason, it is not recommended to use any other configuration.properties files, e.g. client configuration.properties.

It is also possible to use a tool's specific replace.rpl containing the above parameters.

As a first and most important measure after adapting your configuration, the configuration files in the config folder and , if present, the tools own replace.rpl must be signed (Chapter 5.4.3).

In order to start the Hash Checker, you need to provide the path to the configuration file (api-)configuration.properties, the (server- or tools-) configuration file replace.rpl and the `security.zip` file, necessary for both, client and server. It will be started with the following exemplary command:

```
./hash-checker.sh -sz ../server/config/security.zip -api
config/configuration.properties -rpl config/replace.rpl -u <your-user> -
p <your-password> -g boxall -ph -dh -c demo -pl BOX
```

Pease note: The above script represents the basic command for running the Hash Checker. The following list does explain the full scope of possible values available for the Hash Checker, though the values used in the basic script are compulsory, additional values allow a more selective run.

Usage:

```
java [JAVA_OPTS] com.intercope.mpo.tools.hashchecker.Main [-?] [-lt] -sz
<FILE> [-mek <FILE>] [-api <file>] [-rpl <file>] -c <cltprefix>
-u <uname> -p <password> | -ep <encpwd>  [-g <groups>] [-in] -pl
<products> [-cpl <prefixes>] [-a <years>] [-parfile <file>] [-ph] [-dh]
```

-?	--help	print this help
-lt	--list-tables	lists table groups and their included table names
-sz	--security-zip <FILE>	security zip file used to verify integrity of various configuration files
-mek	<FILE>	file containing the encrypted MEK to use
-api	--apiconfig <file>	api configuration file
-rpl	--replacement-properties <file>	replacement configuration file
-c	--clientprefix <cltprefix>	client prefix
-ce	--check-existing	checks only existing row
-u	--user <uname>	username
-p	--password <password>	password
-ep	--encrypted-password <encpwd>	encrypted password
-g	--groups <groups>	comma separated list of table groups to process, optionally the group name can be followed by opening and closing brackets containing a comma separated list of table names, e.g. -g 'upm(GENACL,GENACLEA)'-in,--ignore-nonstatic ignore nonstatic data tables in mixed groups, only valid with option '-ph'
-pl	--product-list <products>	comma separated list of product identifiers to use: BC or BOX
-cpl	--clientprefixlist <prefixes>	comma separated list of client prefixes, defaults to login-

		clientprefix
-a	--annual <years>	comma separated list of years used for annual archive tables
-parfile	<file>	parameter file for command line arguments, each
-ph	--phash	calculate missing PHASHes
-dh	--dhash	calculate missing DHASHes

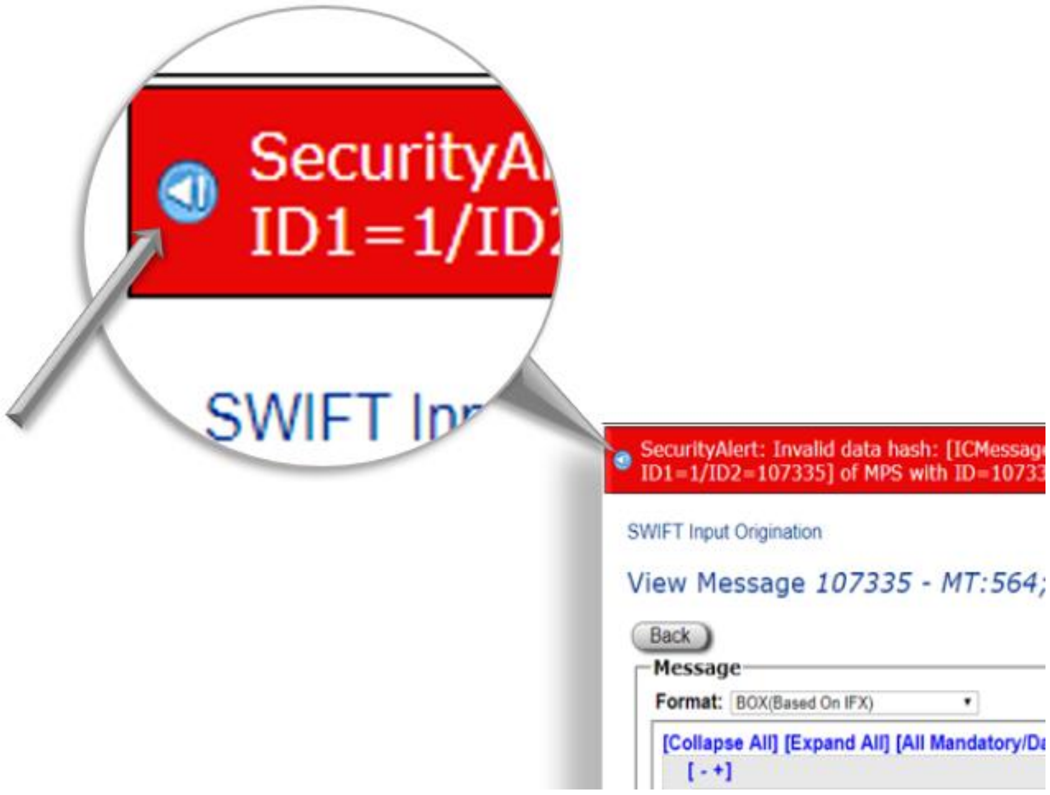
When checking the hash values or when calculating new hash values, the hash checker tool writes a log file (**mismatch.log**) in which any mismatches can be found.

Sample Output:

```
[…]
2018-01-11T14:44:59,006 # P-HASH mismatch for table 'SECPOL01', row '[SECPOLID=39]', old P-HASH 's00101:1ih45P5Lne55TlN1wdGw8Z2K+4e56oF+1vvNuXm4c2s='
2018-01-11T14:44:59,007 # P-HASH mismatch for table 'SECPOL01', row '[SECPOLID=44]', old P-HASH 's00101:iRrsYTn5Lky9sUDn/1ZFnht2sPoLpNobvxhG2iul36E='
2018-01-11T14:44:59,007 # P-HASH mismatch for table 'SECPOL01', row '[SECPOLID=47]', old P-HASH 's00101:xqnsFjKMY4cZGAfmWIg/QEbrtdgeJ5s63yBCeEEDmFg='
2018-01-11T14:44:59,008 # checked 'SECPOL01' with 16 rows and 3 P-HASH mismatches found
[…]
```

IMPORTANT

If a message has been compromised and the hashchecker has been run again, the warning must be deactivated, before a new message is authorized. If this is not done, the next incoming message will be shown as compromised. Deactivation is done by clicking on the blue sign.



SecurityAlert: Invalid data hash: [ICMessag
ID1=1/ID2=107335] of MPS with ID=10733

SWIFT Input Origination

View Message 107335 - MT:564;

SecurityAl
ID1=1/ID
SWIFT In
Back
Message
Format: BOX(Based On IFX)
[Collapse All] [Expand All] [All Mandatory/Da
[- +]

6.2 Gap Detection

Database integrity is ensured using a Gap Detection technique i.e. by regular checking of the database at record level to verify that no gaps exist in sequential numbering of the Message Processing Sequences (MPS) for all message types (e.g. FIN, FileAct, InterAct), therefore confirming that no modifications have been made to the stored data.

This checking of records is performed on the BOX MPS repository at the full system level, not at the level of an individual company within a corporate group-type implementation.

Gap detection is configured on one hand in the Security Check Tool - mpo_secchck (see 12.1) and on the other hand in the BOX Deleter module.

In the Central Server configuration, section `[DELETER]` contains parameters for scheduling the Deleter functionality. There is one parameter for each weekday, e.g. `[DELETER].<MONDAY>_INTERVAL`, `[DELETER].<TUESDAY>_INTERVAL`, etc.

These parameters specify the start and stop intervals for each day of the week.

The Deleter time interval syntax is:

```
<deleter command><start time 1> [- <end time 1>][,<deleter
command><start time 2> [- <end time 2>]][,<deleter command><start time
3> [- <end time 3>]]
```

where:

deleter command	'D': Run MPS 'D'eleter 'G': Perform MPS 'G'ap Detection 'S': Do database 'S'tatistics update 'U': Run 'U'ser delete ‘B’: Run deletion of expired text rebulk archive entries 'M': Run NCM deletion on expired 'M'essage audits 'T': Run NCM deletion on expired s'T'atic audits 'C': Run NCM deletion on expired 'C'onfiguration change sets
start time	Time format: 'HH:MM'
end time	Time format: 'HH:MM'.

Specifying a deleter command ‘G’ (Gap detection) for these parameters is the preferred way because it guarantees optimal gap checking.

We recommend using the default parameters or, respectively, not configuring the Gap Detection parameters, e.g. `GAPD_CHECK_STARTING_HOUR` in section `[GAP_DETECTION]`.

This results in an automatic, regular Gap check by the Deleter module before deletion

7 Backend Security

Communication between Back-end Applications and BOX is implemented via
MQ Communication Gateways (messages) or File Communication Gateways (files)

The MQ Gateway can

- Calculate a `LAU` value for data that is to be sent and send the calculated value with the message
- Extract a `LAU` value from a received message and verify the message using the extracted `LAU` value.

The File Gateway can

- Calculate a `LAU` value for files that are to be sent and send the calculated value in a specially created `LAU`-file together with the original file to the back-end application.
- Calculate a `LAU` value for a received original file and extract a `LAU` value from a received `LAU`-file. The file is the verified by comparing the extracted `LAU` value with the calculated `LAU` value.

For more detailed information, refer to the sections below.

7.1 Configuration

In the configuration of the MQ Gateway `[LCG<MEADOW>.F002]` (or File Gateway `[LCG<MEADOW>.F005]` the (optional) configuration parameter `LAU_KEY` activates the back-office security feature.

As value of the parameter the encrypted `LAU key` that is used for the `LAU` value calculation must be given.

The MQ Gateway configuration includes two additional (optional) parameters:

`LAU_CALCULATION_CODEPAGE`

Per default the `LAU` value for the message data is calculated from the format it is stored in the MQ message. With this parameter you can specify the codepage into which the message data is transformed for `LAU` value calculation before the message is sent.

`RFH2_LAU_KEY_MODE`

This parameter specifies for which section(s) of the message a `LAU` value shall be calculated/verified when MQ messages with RFH2 Header are sent/received.

Possible values are:

PAYLOAD	The LAU value is calculated/verified for the Payload (whereby the value of the parameter LAU_CALCULATION_CODEPAGE will be considered)
RFH2_HEADER	The LAU value is calculated/verified for the NameValues in the RFH2 Header.
BOTH	There is one LAU value calculated/verified for the Payload and one LAU value for the NameValues in the RFH2 Header. This is the default value.

The configuration parameter `SECURITY_FAILURE_SHORTLABEL` in section `[LCG<MEADOW>PEXA]` specifies a Failure Label for all received messages for which a Security Failure were determined (No `LAU` found, `LAU` mismatch, etc.). The processing will then be continued with this Pattern Label.

If no Failure Label has been specified, the message processing is handled as follows:

If an Exception Pattern Label (parameter `DEFAULT_EXCEPTION_SHORTLABEL)` has been specified, the Security Failure Pattern Label is set to the Exception Pattern Label.

If no Exception Pattern Label has been specified, the Security Failure Pattern Label is set to the Default Pattern Label (parameter `DEFAULT_IPS_SHORTLABEL`).

Example:

```
[LCG<MEADOW>.PEXA]

  DEVICE_TYPE                        0xF002

  SECURITY_FAILURE_SHORTLABEL        TagToSecurityFailure
  SECURITY_FAILURE_LABELPREFIX       demo

  IMPORT_CHECK_CYCLE                 5
  GENERATE_COMMAND_REPORT            NO
  DEFAULT_EXCEPTION_LABELPREFIX      demo
  DEFAULT_EXCEPTION_SHORTLABEL       ExceptionLabel
  DEFAULT_IPS_SHORTLABEL             DefaultLabel
  DEFAULT_IPS_LABELPREFIX            demo
  DEFAULT_MPS_INITMODE               2;    ; 1 - Instantiated, 2 - Pattern

[LCG<MEADOW>.F002]

  LAU_KEY
D0E71CD62033F4B489D6CFBB1F5D30CCA54F1441EC0F41548EB640E6C7917662036311A5
9F6CB85D3FB0F7BC7FE5FFF12D0612CB41435EDCD2A32737CC085FA7C2A87E339572ECC3
4B148F64D9251F48   ; LAU Key
1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF

  LAU_CALCULATION_CODEPAGE                EBCDIC
  RFH2_LAU_KEY_MODE                       BOTH    ; PAYLOAD   ; RFH2_HEADER

  PLUGIN_LIBRARY_NAME                     expgi_genflatbuf
  LOCAL_QUEUE_MANAGER                     QM_ichh2wk
  DEFAULT_OUTBOUND_QUEUE_MANAGER          QM_ichh2wk
  DEFAULT_OUTBOUND_QUEUE                  TO.BE1
  INBOUND_QUEUE                           TO.CGTWMEADOW
  DEFAULT_REPLY_QUEUE_MANAGER             QM_ichh2wk
  DEFAULT_REPLY_QUEUE                     TO.CGTWMEADOW
  TRASH_QUEUE_NAME                        TRASH
```

7.2 MQ Message Transmission with LAU Value (MQMD only Mode)

When a message in MQMD only mode is sent, and the Backoffice Security feature is active, a `LAU` block of length 192 bytes is inserted before the message content block. Below you can see a sample of such a message:

```
ICHH2WK0 MPO_SERVER     (37DC:6BC0) <EXIMF002>PutMessage2         17.11.14 15:23:23.482
        A: First MQMD
   000000 | 4D 44 20 20 02 00 00 00 00 00 60 03 08 00 00 00 | MD  ......`.....
   000016 | FF 00 00 00 00 22 02 00 00 B8 04 00 00 | ........".......
   000032 | 20 20 20 20 20 20 20 20 02 00 00 00 01 00 00 00 |         ........
   000048 | 41 4D 51 20 51 4D 5F 69 63 68 68 32 77 6B 20 20 | AMQ QM_ichh2wk
   000064 | 7A 80 05 5A 20 04 29 06 00 00 00 00 00 00 00 00 | z..Z .).........
   000080 | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | ................
   000096 | 00 00 00 00 54 4F 2E 43 47 54 57 4D 45 41 44 4F | ....TO.CGTWMEADO
   000112 | 57 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | W...............
   000128 | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | ................
   000144 | 00 00 00 00 51 4D 5F 69 63 68 68 32 77 6B 00 00 | ....QM_ichh2wk..
   000160 | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | ................
   000176 | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | ................
   000192 | 00 00 00 00 77 6F 6C 66 72 61 6D 20 20 20 20 20 | ....wolfram
   000208 | 16 01 05 15 00 00 00 C6 B1 0D F9 B0 45 8E F1 17 | ............E...
   000224 | E6 BB 82 74 04 00 00 00 00 00 00 00 00 00 00 0B | ...t............
   000240 | 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 |
   000256 | 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 |
   000272 | 0B 00 00 00 44 3A 5C 6D 70 6F 5F 6E 74 5C 6D 70 | ....D:\mpo_nt\mp
   000288 | 6F 5F 73 65 72 76 65 72 2E 65 78 65 20 20 20 20 | o_server.exe
   000304 | 32 30 31 37 31 31 31 34 31 34 32 33 32 33 34 38 | 2017111414232348
   000320 | 20 20 20 20 00 00 00 00 00 00 00 00 00 00 00 00 |     ............
   000336 | 00 00 00 00 00 00 00 00 00 00 00 00 01 00 00 00 | ................
   000352 | 00 00 00 00 00 00 00 00 FF xx | ............xxxx

ICHH2WK0 MPO_SERVER     (37DC:6BC0) <EXIMF002>PutMessage2         17.11.14 15:23:23.482
        A: MessageContent
   000000 | 4C 41 55 56 41 4C 3D 43 50 45 42 43 44 49 43 3A | LAUVAL=CPEBCDIC:
   000016 | 61 62 68 6B 61 49 72 67 4C 41 71 74 47 71 41 6A | abhkaIrgLAqtGqAj
   000032 | 49 47 33 6F 48 39 48 70 35 52 4D 77 36 4C 46 7A | IG3oH9Hp5RMw6LFz
   000048 | 37 6C 6E 6B 2F 35 77 65 79 70 66 62 39 4F 74 72 | 7lnk/5weypfb9Otr
   000064 | 71 48 49 50 67 46 49 4C 42 47 6E 6F 66 4D 67 6F | qHIPgFILBGnofMgo
   000080 | 67 59 34 61 30 41 63 41 6E 36 61 43 39 4C 56 48 | gY4a0AcAn6aC9LVH
   000096 | 32 77 4F 6E 67 65 6E 64 71 62 4D 58 4C 78 41 50 | 2wOngendqbMXLxAP
   000112 | 50 44 49 6E 46 66 71 38 71 31 77 3D 20 20 20 20 | PDInFfq8q1w=
   000128 | 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 |
   000144 | 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 |
   000160 | 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 |
   000176 | 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 |
   000192 | 7B 31 3A 46 30 31 50 54 53 41 44 45 53 41 41 58 | {1:F01PTSADESAAX
   000208 | 58 58 30 30 30 30 30 30 30 30 30 30 7D 7B 32 3A | XX0000000000}{2:
   000224 | 49 31 30 33 50 54 53 41 44 45 53 41 41 58 58 58 | I103PTSADESAAXXX
   000240 | 4E 32 7D 7B 33 3A 7B 31 30 38 3A 31 32 33 34 35 | N2}{3:{108:12345
   000256 | 36 37 38 39 30 31 32 33 34 35 36 7D 7B 34 3A | 67890123456}}{4:
   000272 | 0D 0A 3A 32 30 3A 32 30 53 78 78 78 78 78 78 78 | ..:20:20Sxxxxxxx
   000288 | 2E 54 0D 0A 3A 32 33 42 3A 53 50 52 49 0D 0A 3A | .T..:23B:SPRI..:
   000304 | 33 32 41 3A 30 31 31 31 31 37 45 55 52 31 32 33 | 32A:011117EUR123
   000320 | 2C 34 35 0D 0A 3A 33 33 42 3A 45 55 52 31 2C 30 | ,45..:33B:EUR1,0
   000336 | 0D 0A 3A 35 30 41 3A 2F 33 34 78 0D 0A 50 54 53 | ..:50A:/34x..PTS
   000352 | 41 44 45 53 41 58 58 58 0D 0A 3A 35 39 3A 2F 33 | ADESAXXX..:59:/3
   000368 | 34 78 0D 0A 34 58 33 35 78 0D 0A 3A 37 31 41 3A | 4x..4X35x..:71A:
   000384 | 4F 55 52 0D 0A 2D 7D xx xx xx | OUR..-}xxxxxxxxx
```

The `LAU` block begins with `LAUVAL`=, followed by an optional Prefix containing additional information that ends with a colon (‘**:**’) and by the actual `LAU` Value.

The rest of the block is filled with blanks.

The `LAU` Value is calculated as follows:

1. An `SHA256` Digest is calculated for the Message Data.
2. The 32 bytes Digest resulting from step 1 above is coded in HexAscii (lower case).
3. This HexAscii string is encrypted with the `LAU` Key specified by the configuration parameter `LAU_KEY` (Encryption method is `MP_ENCRYPT_MTHD_AES256_ECB_PKCS5`).

4. The encrypted data in BASE64 encoding then represents the actual `LAU` Value.

The Prefix is a comma-separated list of options. There are three (3) options:

CP =	`Codepage. Specifies the code page used for LAU Value calculation.`
ALG =	`Algorithm. Consists of Digest and Encryption Algorithm. Currently only encryption method MP_ENCRYPT_MTHD_AES256_ECB is used.`
PLEN =	`Length of the Prefix (including the ending colon (':').`

For message transmission the codepage can be specified only if the configuration parameter `LAU_CALCULATION_CODEPAGE` has been set.

The codepage used for the `LAU` block is the same as the one specified for the Message Payload in the `MQMD` Header, i.e. if the MQ Plug-in sends the message in `EBCDIC`, also the `LAU` block will be coded in `EBCDIC`.

7.3 MQ Message Reception with LAU Value (MQMD only Mode)

At message reception, the MQ Gateway tries to analyse and to check the `LAU` block (Does the block begin with `LAUVAL=`, are the Prefix and the `LAU` Value present, is the block filled with blanks?).

If the `LAU` block is not valid or if it is missing, the MPS obtains the Security Failure Code `'LAU Missing Failure'`.

The message data is handed over to the Gateway Plug-in as received (in order to be further processed). If the processing is successful, a valid MPS is created. Otherwise an Exception MPS will be created and the processing will continue with the specified Security Failure Pattern Label (NOT with the Default Pattern Label or the Exception Pattern Label).

If a Prefix is found and the CP option has been specified, then the message data after the `LAU` block will be translated into the specified Codepage for `LAU` calculation.

The `LAU` block is then removed from the received message (i.e. it is not any more visible for the plug-in).

If there is a mismatch between the extracted and the calculated `LAU` Value, the received MPS obtains the Security Failure Code `'LAU Mismatch Failure'`.

If the extracted and the calculated `LAU` Values match, the received MPS obtains the Security Failure Code `'No Security Issue'`.

In case of processing errors during the `LAU` verification, the received MPS obtains the Security Failure Code `'LAU Processing Failure'`.

7.4 MQ Message Transmission with LAU Value (RFH2 Mode)

At message transmission with `LAU` Value an additional NameValue containing the `LAU` Value data is inserted into the RFH2 Header. Below you can see a sample of such a message:

```
CHH2WK0 MPO_SERVER     (7E34:473C) <EXIMF002>PutMessage2         17.11.14 15:33:57.136
        A: First MQMD
   000000 | 4D 44 20 20 02 00 00 00 00 00 60 03 08 00 00 00 | MD  ......`.....
   000016 | FF 00 00 00 00 22 02 00 00 B8 04 00 00 | ........".......
   000032 | 4D 51 48 52 46 32 20 20 08 00 00 00 01 00 00 00 | MQHRF2  ........
   000048 | 41 4D 51 20 51 4D 5F 69 63 68 68 32 77 6B 20 20 | AMQ QM_ichh2wk
   000064 | 7A 80 05 5A 20 04 4B 04 00 00 00 00 00 00 00 00 | z..Z .K.........
   000080 | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | ................
   000096 | 00 00 00 00 54 4F 2E 43 47 54 57 4D 45 41 44 4F | ....TO.CGTWMEADO
   000112 | 57 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | W...............
   000128 | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | ................
   000144 | 00 00 00 00 51 4D 5F 69 63 68 68 32 77 6B 00 00 | ....QM_ichh2wk..
   000160 | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | ................
   000176 | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | ................
   000192 | 00 00 00 00 77 6F 6C 66 72 61 6D 20 20 20 20 20 | ....wolfram
   000208 | 16 01 05 15 00 00 00 C6 B1 0D F9 B0 45 8E F1 17 | ............E...
   000224 | E6 BB 82 74 04 00 00 00 00 00 00 00 00 00 00 0B | ...t............
   000240 | 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 |
   000256 | 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 |
   000272 | 0B 00 00 00 44 3A 5C 6D 70 6F 5F 6E 74 5C 6D 70 | ....D:\mpo_nt\mp
   000288 | 6F 5F 73 65 72 76 65 72 2E 65 78 65 20 20 20 20 | o_server.exe
   000304 | 32 30 31 37 31 31 31 34 31 34 33 33 35 37 31 33 | 2017111414335713
   000320 | 20 20 20 20 00 00 00 00 00 00 00 00 00 00 00 00 |     ............
   000336 | 00 00 00 00 00 00 00 00 00 00 00 00 01 00 00 00 | ................
   000352 | 00 00 00 00 00 00 00 00 FF xx | ............xxxx
ICHH2WK0 MPO_SERVER     (7E34:473C) <EXIMF002>PutMessage2         17.11.14 15:33:57.136
        A: MessageContent
   000000 | 52 46 48 20 02 00 00 00 BC 01 00 00 22 02 00 00 | RFH ........"...
   000016 | B8 04 00 00 4D 51 53 54 52 20 20 20 00 00 00 00 | ....MQSTR   ....
   000032 | B8 04 00 00 64 01 00 00 3C 4C 61 75 56 61 6C 75 | ....d...<LauValu
   000048 | 65 44 61 74 61 3E 3C 52 46 48 32 4E 61 6D 65 56 | eData><RFH2NameV
   000064 | 61 6C 75 65 4C 61 75 3E 73 67 76 42 49 70 49 64 | alueLau>sgvBIpId
   000080 | 77 70 4A 51 65 4C 79 67 78 4E 36 74 51 30 73 46 | wpJQeLygxN6tQ0sF
   000096 | 63 50 64 61 51 50 31 72 46 48 4A 33 37 6E 66 35 | cPdaQP1rFHJ37nf5
   000112 | 37 54 7A 46 73 39 65 52 39 37 67 4B 2B 65 4E 42 | 7TzFs9eR97gK+eNB
   000128 | 63 2B 53 52 71 4A 6F 30 70 4E 34 70 7A 37 57 | c++SRqJo0pN4pz7W
   000144 | 42 7A 49 63 6C 78 63 6C 53 62 46 75 4E 65 6E 64 | BzIclxclSbFuNend
   000160 | 71 62 4D 58 4C 78 41 50 50 44 49 6E 46 66 71 38 | qbMXLxAPPDInFfq8
   000176 | 71 31 77 3D 3C 2F 52 46 48 32 4E 61 6D 65 56 61 | q1w=</RFH2NameVa
   000192 | 6C 75 65 4C 61 75 3E 3C 50 61 79 6C 6F 61 64 4C | lueLau><PayloadL
   000208 | 61 75 3E 61 62 68 6B 61 49 72 67 4C 41 71 74 47 | au>abhkaIrgLAqtG
   000224 | 71 41 6A 49 47 33 6F 48 39 48 70 35 52 4D 77 36 | qAjIG3oH9Hp5RMw6
   000240 | 4C 46 7A 37 6C 6E 6B 2F 35 77 65 79 70 66 62 39 | LFz7lnk/5weypfb9
   000256 | 4F 74 72 71 48 49 50 67 46 49 4C 42 47 6E 6F 66 | OtrqHIPgFILBGnof
   000272 | 4D 67 6F 67 59 34 61 30 41 63 41 6E 36 61 43 39 | MgogY4a0AcAn6aC9
   000288 | 4C 56 48 32 77 4F 6E 67 65 6E 64 71 62 4D 58 4C | LVH2wOngendqbMXL
   000304 | 78 41 50 50 44 49 6E 46 66 71 38 71 31 77 3D 3C | xAPPDInFfq8q1w=<
   000320 | 2F 50 61 79 6C 6F 61 64 4C 61 75 3E 3C 50 61 79 | /PayloadLau><Pay
   000336 | 6C 6F 61 64 4C 61 75 43 6F 64 65 70 61 67 65 3E | loadLauCodepage>
   000352 | 45 42 43 44 49 43 3C 2F 50 61 79 6C 6F 61 64 4C | EBCDIC</PayloadL
   000368 | 61 75 43 6F 64 65 70 61 67 65 3E 3C 2F 4C 61 75 | auCodepage></Lau
   000384 | 56 61 6C 75 65 44 61 74 61 3E 20 20 14 00 00 00 | ValueData>  ....
   000400 | 3C 74 65 73 74 31 3E 78 78 78 78 78 3C 2F 74 65 | <test1>xxxxx</te
   000416 | 73 74 31 3E 14 00 00 00 3C 74 65 73 74 32 3E 78 | st1>....<test2>x
   000432 | 78 78 78 78 3C 2F 74 65 73 74 32 3E 7B 31 3A 46 | xxxx</test2>{1:F
   000448 | 30 31 50 54 53 41 44 45 53 41 41 58 58 58 30 30 | 01PTSADESAAXXX00
   000464 | 30 30 30 30 30 30 30 30 7D 7B 32 3A 49 31 30 33 | 00000000}{2:I103
   000480 | 50 54 53 41 44 45 53 41 41 58 58 58 4E 32 7D 7B | PTSADESAAXXXN2}{
   000496 | 33 3A 7B 31 30 38 3A 31 32 33 34 35 36 37 38 39 | 3:{108:123456789
   000512 | 30 31 32 33 34 35 36 7D 7B 34 3A 0D 0A 3A 32 | 0123456}}{4:..:2
   000528 | 30 3A 32 30 53 78 78 78 78 78 78 78 2E 54 0D 0A | 0:20Sxxxxxxx.T..
   000544 | 3A 32 33 42 3A 53 50 52 49 0D 0A 3A 33 32 41 3A | :23B:SPRI..:32A:
   000560 | 30 31 31 31 31 37 45 55 52 31 32 33 2C 34 35 0D | 011117EUR123,45.
   000576 | 0A 3A 33 33 42 3A 45 55 52 31 2C 30 0D 0A 3A 35 | .:33B:EUR1,0..:5
   000592 | 30 41 3A 2F 33 34 78 0D 0A 50 54 53 41 44 45 53 | 0A:/34x..PTSADES
   000608 | 41 58 58 58 0D 0A 3A 35 39 3A 2F 33 34 78 0D 0A | AXXX..:59:/34x..
   000624 | 34 58 33 35 78 0D 0A 3A 37 31 41 3A 4F 55 52 0D | 4X35x..:71A:OUR.
   000640 | 0A 2D 7D xx xx xx xx | .-}xxxxxxxxxxxxx
```

The first NameValue in the RFH2 Header was created by the MQ Gateway and it contains all `LAU` relevant data.

The Root Node is `LauValueData`.

This Node may have the following Child Nodes:

RFH2NameValueLau	The LAU Value that was calculated for the NameValues of the RFH2 Header. For the calculation of the LAU Value the NameValues are concatenated as they occur in the RFH2 Header (including the padding blanks at the end but without length information).
PayloadLau	The LAU Value that was calculated for the payload. The value of the configuration parameter LAU_CALCULATION_CODEPAGE is considered (Message Transmission).
PayloadLauCodepage	Name of the Codepage (MPO naming) used when calculating the LAU Value for the payload (Messages Reception).
LauAlgorithm	Digest and Encryption Algorithm. Currently not evaluated!

7.5 MQ Message Reception with LAU Value (RFH2 Mode)

At message reception the Gateway tries to find and to analyse the LauValueData NameValue.

If LauValueData is not found or if RFH2NameValueLau or PayLoadLau are not found, even if required by the configuration parameter `RFH2_LAU_KEY_MODE`, then the MPS obtains the Security Failure Code `'LAU Missing Failure'`.

In case of failure during the verification of the `LAU`, the MPS obtains the Security Failure Code `'LAU Processing Failure'`.

If there is a mismatch between the extracted and the calculated `LAU` Value, the received MPS obtains the Security Failure Code `'LAU Mismatch Failure'`.

In case of a security failure the message processing will continue with the Security Failure Pattern Label (if specified). However, the MPS will be handed over to the plug-in, even if the security check fails.

If the extracted and the calculated `LAU` Values match, the received MPS obtains the Security Failure Code `'No Security Issue'`.

For verification of the RFH2 NameValue `LAU`, all NameValues, except for LauValueData, will be concatenated.

7.6 Transmission and Reception of Files

Transmission and Reception of Files from / to a back-end application is implemented via the BOX File (Communication) Gateway.

The Backoffice Security feature for the File Gateway works like the one for MQ Gateway in MQMD only Mode.

However, the `LAU` Value is not located in the files themselves.

At transmission the File Gateway creates for each file it writes an additional file (`LAU` File) that contains the `LAU` Value. The name of this file is the same as the original file with the additional extension `'.lau'.`

The content of the `LAU` File begins with `LAUVAL=` followed by the `LAU` Value. There is no Prefix.

At reception the Gateway expects for each received file a `LAU` File (same name as the received file, with an additional extension '**.lau**') that contains the `LAU` Value.

The `LAU` Value is calculated for each received original file and then compared with the `LAU` Value that was extracted from the `LAU` File.

If no respective *.lau file can be found for a received file or if the `LAU` Value cannot be extracted from the `LAU` File, then the MPS obtains the Security Failure Code `'LAU Missing Failure'.`

In case of failure during the verification of the `LAU`, the MPS obtains the Security Failure Code `'LAU Processing Failure'.`

If there is a mismatch between the extracted and the calculated `LAU` Value, the received MPS obtains the Security Failure Code `'LAU Mismatch Failure'.`

In case of a security failure the message processing will continue with the Security Failure Pattern Label (if specified). However, the MPS will be handed over to the plug-in, even if the security check fails.

If the extracted and the calculated `LAU` Values match, the received MPS obtains the Security Failure Code `'No Security Issue'.`

8 Authentication

8.1 Radius

BOX supports the Remote Authentication Dial-In User Service (RADIUS) protocol and that enables authentication of dial-in users and authorizing their access to the BOX system.

The settings on **BOX** side are configured within a **Security Policy**.

For more details on the configuration of the RADIUS-based Authentication, refer to the BOX User Profile Management documentation.

8.2 Multi-Factor Authentication

BOX provides embedded Dual-Factor Authentication for Operator PC login.

This feature provides a higher security level for user logins by providing the support for a second authentication factor.

The Dual-Factor Authentication requires an additional time-based one-time password (TOTP) to be supplied at login.

8.2.1 TOTP by RFC-6238 Compliant Authenticator

The TOTP can be generated and provided by an RFC-6238 compliant Authenticator app/program and then be sent to the user logging in.

Dual Factor Authentication configuration options allow for

- Specifying the number of digits of the TOTP used for authentication.
- Selecting the encryption algorithm for the TOTP. Possible algorithms are SHA-1, SHA-256 and SHA-512.
- Defining the period that TOTP code will be valid for.

For more details on the configuration of the Dual-Factor Authentication by RFC-6238 Compliant Authenticator, refer to the BOX User Profile Management documentation.

8.2.2 TOTP via SMTP

The TOTP can also be generated by BOX and be sent via SMTP to the user logging in.

Also, in this case Dual Factor Authentication configuration options allow for

- Specifying the number of digits of the TOTP used for authentication.
- Selecting the encryption algorithm for the TOTP. Possible algorithms are SHA-1, SHA-256 and SHA-512.
- Defining the period that TOTP code will be valid for.

For more details on the configuration of the Dual-Factor Authentication via SMTP, refer to the BOX User Profile Management documentation.

9 Change Management System of BOX

For all BOX modules that have a database connection (`Server, Messaging Interface; Communication Gateways`, etc) the configuration can be done using the `BOX GUI`.

The configuration via `GUI` is protected by the new `Configuration Change Management (CM)` that was presented with Version 3 Release 20 (V3R20).

This new CM concerns the configuration of all workflow-related objects `(Instruction Patterns, IPS, Analysis and Content Processing modules, Address Books`, etc.) and the UPM configuration, i.e. practically all aspects of configuration except for message objects (`Messages, Message History, Archives`) and `SWIFT Reference Data`, such as BIC Directory.

The new CM offers enhanced security options against unauthorized changes as all configuration changes are subject to 0 to n authorizations before being applied.

For detailed information on the `BOX Configuration Change Management`, refer to the BOX Administration Guide documentation.

10 Password Security Rules

Each user ID is attached to a Security Policy. Within this Security Policy general password security and login rules are specified.

With the password security rules you can specify

- The maximum password age, i.e. after how many days the password must be changed.
- Initial password lifetime, i.e. a time range after initial password allocation within which the initial password must be changed. It is possible to specify that the password must be changed by the user immediately on initial password allocation.
- Password encryption method.
- Password history length, i.e. a cycle or time range within which an already password cannot be re-used.

For the password itself you can specify constraints, such as:

- Minimum password length. Range is 0 – 99 characters.
- Minimum number of uppercase letters.
- Minimum number of lowercase letters.
- Minimum number of digits.
- Minimum number special characters:

Further rules for the password are:

- The number of occurrences of the same character in the password must be equal to or less than half the number of characters in the password minus one: ((L-1)/2) m-1.
- It must not be the same as the name of object protected by the password or the User ID.
- It must not contain a long sub-string of name of object protected by the password or of the User ID.
- It must not be an ‘Illegal pattern’, i.e. it must not be based on ordinary words.

For the login you can specify:

- Whether multiple login is possible or not.
- The maximum number of consecutive Login failures before the account will be locked. Only a user administrator can unlock the account. Failed attempts are logged.
- Lockout interval, i.e. the time the user will be locked out after too many consecutive login failures. Range is 1 minute – permanent lockout.
- A maximum inactivity period, i.e. a period after which a user will be blocked if he has not logged in to the system.
- A session timeout, i.e. a time after which a user is logged out automatically due to inactivity.

11 Logging and Monitoring

For convenient and reliable recording of security events and detection of anomalous actions and operations within the system, BOX can be configured to export alerts to `syslog` in CEF (Common Event Format) which is supported by various SIEM (Security Information and Event Management) products.

The configuration is done with the following parameter (in the monitor configuration file, mon.cfg) `USE_CEF_IN_SYSLOG` in section `[EXPORT0099]`.

Setting this parameter to YES results in alerts being exported to `syslog` in CEF format.

By setting parameters `MESSAGE_SECURITY_WARNING` and `SYSTEM_SECURITY_WARNING` to YES and all other parameters to NO the exporting of alerts can be filtered so that only security related alerts will be exported to `syslog`.

Example:

```
[EXPORT0099]
  USE_CEF_IN_SYSLOG             YES
  MESSAGE_SECURITY_WARNING      YES
  SYSTEM_SECURITY_WARNING       YES
```

Security related alerts are also written to each applicable log file starting with the log entry letter ‘Y’.

Example

```
MPO_IPNS       (00007f2d:47b87740) PSEC_CheckPSHInStructureData
18.12.14 17:43:20:824
        Y: Failure x004A0107 (W:Secured Object Data could not be
verified successfully.), cannot process hash version 0 for structure
'MP_DBTAS_KEYWORD' ID x6502 (Parameter
'SYSTEM.[ROOT_SYSTEM_INSTANCE].CM_DEVELOPMENT_MODE' for owner 'root
node')
```

For more details on alerts and the configuration, refer to the documents

BOX Configuration Guide and **BOX Alerts and SNMP Traps Reference Guide**.

12 Tools

12.1 Security Check Tool - mpo_secchck

The Security Check Tool can be used for checking the system regarding gaps in the MPS repository (Gap Detection) and for checking the code in regard missing signatures for code files (Code Check)

The tool delivers a security report file (name as specified for option –R, see below) and a log file.

Usage:

MPO_SECCHK

-c	ConfigFilename	Name of the configuration from which the security related configuration parameters (Section [SECURITY]) shall be read.
-X	E-MEK-Filename	Name of the file containing the encrypted Master Encryption Key.ConfigDirectory Directory where e-MEK file is located (default is ./config). If only filename was specified with parameters -c, -X, -r this is also the directory where configuration file, replacement file and e-MEK are located.
-C	ConfigDirectory	
-L	LogFile	
-S	LogFileSize	
-D	DebugFlags	
-r	ReplacementFilename	
-R	SecurityReportFilename	
-NoCodeCheck		Specifies whether code shall be checked regarding missing signatures or not.
-NoGapCheck		Specifies whether MPS repository shall be checked in regard gaps or not.

Examples:

LINUX Check

For convenience, create a script, such as mpo_secchck.sh containing the following exemplary script **without missing signatures and Gap detection**:

```
#!/bin/sh
MPO_BASE=/opt/box/server
$MPO_BASE/bin/mpo_secchk -c:$MPO_BASE/config/secchk.cfg -
X:$MPO_BASE/security/basesec/mek.enc -r:$MPO_BASE/config/replace.rpl -
R:report.txt -D:0xFFFF3B3F -L:$MPO_BASE/logs/secchk.log
```

Create a configuration the file secchk.cfg in ...server/config/ and make sure to sign it using the security tool ‘signconfig.sh’.

The configuration file should include:

```
[DB_INTERFACE]
   DATA_SOURCE_NAME                                $$R$DB_DATABASE_SOURCE$
   DATABASE_NAME                                   $$R$DB_DATABASE_NAME$
   DATABASE_QUALIFIER                              $$R$DB_DATABASE_QUALIFIER$
   DATABASE_USERNAME                               $$R$DB_DATABASE_USER$
   DATABASE_PASSWORD                               $$R$DB_DATABASE_ENC_PWD$
   DB_INTERFACE_LIBRARY                            $$R$DB_DATABASE_LIBRARY$
   MAX_CONNECTIONS                                 60
   CONNECTION_TIMEOUT                              2700
   CONNECTION_TIMEOUT_CHECK_INTERVAL               900
   EXECUTION_TRACE                                 3

[JAVA_ENVIRONMENT]
  JAVA_OPTIONS                                         --verbose -
Djava.class.path=$$R$SRV_JAVA_CLASSPATH$

[SECURITY]
  BASE_SECRET_LIST
  SECURITY_ZIP_FILE                                     security/security.zip
  CUSTOMER_CONFIG_CERT
  CONTINUE_INIT_ON_SECURITY_FAILURES                    YES
  CONTINUE_PROC_ON_SECURITY_FAILURES                    YES
  SUPPRESS_CONFIGURATION_SIGNATURE_VERIFICATION_ALERTS  NO
  SUPPRESS_CODE_SIGNATURE_VERIFICATION_ALERTS           NO
  SUPPRESS_STATIC_DATA_VERIFICATION_ALERTS              NO
  SUPPRESS_NONSTATIC_DATA_VERIFICATION_ALERTS           NO
  SUPPRESS_CODE_SECURITY_CHECK                          NO
  ARCHIVEDATA_PROTECTION_STARTDATE
$$R$ARCHIVEDATA_PROTECTION_STARTDATE$
  NONSTATICDATA_BOOTSTRAP_ENDDATE
$$R$NONSTATICDATA_BOOTSTRAP_ENDDATE$

[SESSION_COMM_INTERFACE]
   DISPATCHER_COUNT                                1

[SESSION_DISPATCHER01]
   CSSUBSYSTEM                                     TCP/IP
   MAXSESSIONS                                     27

[COMMSYS_T]
   OWN_IPADDRESS                                   $$R$SRV_HOST_NAME$
   DATA_ENCRYPTION_METHOD                          NOENCRYPT
   MPO_NAMESERVER                                  $$R$SRV_IPNS_HOST$
   MPO_NAMESERVER_PORT                             $$R$SRV_IPNS_PORT$
   SRV_USE_OWN_IPADDRESS_AS_ORIGINATION
$$R$SRV_USE_OWN_IPADDRESS_AS_ORIGINATION$
   LISTEN_ON_OWN_IPADDRESS_ONLY
$$R$SRV_LISTEN_ON_OWN_IPADDRESS_ONLY$
```

Please note, if the configuration contains the path to the security.zip file, the option '-X' is not needed within the script.

Resulting Report with found gaps

```
+++ Security Report: Security check tool started at Thu Jan 10 16:44:35 2019
>>> Security Report: Executable code and Java class security check started at
Thu Jan 10 16:44:38 2019
Executable code and Java class security checks finished successfully
<<< Security Report: Executable code security check completed at Thu Jan 10
16:44:53 2019
>>> Security Report: GAP detection check started at Thu Jan 10 16:44:53 2019
Gap detection data (20/MPS/MPS): 1. issue: Number 20 missing in data
Gap detection data (20/MPS/MPS): 2. issue: Number 40 missing in data
Gap detection data (20/MPS/MPS): 3. issue: Number 41 missing in data
Gap detection data (20/MPS/MPS): 4. issue: Number 42 missing in data
Gap detection for (20/MPS/MPS): Failure x004A0110 (W:Completeness check failed,
a (database) record is missing and might have been deleted.), 4 missing items, 0
items missing in repository
<<< Security Report: GAP detection check completed at Thu Jan 10 16:44:53 2019
+++ Security check completed successfully.
+++ Security Report: Security check tool completed at Thu Jan 10 16:44:53 2019
```

Code check only without missing signatures:

```
#!/bin/sh
MPO_BASE=/opt/box/server
$MPO_BASE/bin/mpo_secchk -c:$MPO_BASE/config/secchk.cfg -
X:$MPO_BASE/security/basesec/mek.enc -r:$MPO_BASE/config/replace.rpl -
R:report.txt -D:0xFFFF3B3F -L:$MPO_BASE/logs/secchk.log -NoGapCheck
```

Resulting Report

```
+++ Security Report: Security check tool started at Thu Jan 10 16:47:29 2019
+++ GAP checks suppressed.
>>> Security Report: Executable code and Java class security check started at
Thu Jan 10 16:47:31 2019
Executable code and Java class security checks finished successfully
<<< Security Report: Executable code security check completed at Thu Jan 10
16:47:47 2019
+++ Security check completed successfully.
+++ Security Report: Security check tool completed at Thu Jan 10 16:47:47 2019
```

WINDOWS check

Code check only without missing signatures:

```
mpo_secchk.exe -c:sec.cfg -X:mek.enc -R:report.txt -D:0xFFFF3B3F –NoGapCheck
```

Report text:

```
+++ Security Report: Security check tool started at Thu Mar 01 08:57:40 2018
+++ GAP checks suppressed.
>>> Security Report: Executable code security check started at Thu Mar 01
08:57:42 2018
Executable code security checks finished successfully
<<< Security Report: Executable code security check completed at Thu Mar 01
08:57:42 2018
+++ Security check completed successfully.
+++ Security Report: Security check tool completed at Thu Mar 01 08:57:42 2018
```

The error is logged and can be found in the log file:

```
ICHH2AK0 MPO_SECCHK     (2AA0:10EC) ValidateSignature              18.03.01
09:02:31.387
        Y: Failure x004A0109 (W:File digest mismatch) for
'D:\box\mp_upmsync.dll'
ICHH2AK0 MPO_SECCHK     (2AA0:10EC) PSEC_ValidateCodeFileSignature 18.03.01
09:02:31.387
        Y: Failure x004A0109 (W:File digest mismatch) validating XML signature
from 'D:\box\mp_upmsync.dll.sig.xml' for code file
'D:\akwork\test\secchk\mp_upmsync.dll'
```

Code check without missing signatures and Gap detection with found gaps:

```
mpo_secchk.exe -c:sec.cfg -X:mek.enc -R:report.txt -D:0xFFFF3B3F
Report text:
+++ Security Report: Security check tool started at Thu Mar 01 09:15:23 2018
>>> Security Report: Executable code security check started at Thu Mar 01
09:15:25 2018
```

```
Executable code security checks finished successfully
<<< Security Report: Executable code security check completed at Thu Mar 01
09:15:25 2018
>>> Security Report: GAP detection check started at Thu Mar 01 09:15:25 2018
Gap detection data (1/MPS/MPS): 1. issue: Number 448711 missing in data
Gap detection data (1/MPS/MPS): 1. issue: Number 448722 contained in data but
missing
in repository
Gap detection data (1/MPS/MPS): 2. issue: Number 448723 contained in data but
missing
in repository
Gap detection data (1/MPS/MPS): 3. issue: Number 448724 contained in data but
missing
in repository
Gap detection for (1/MPS/MPS): Failure x004A0110 (W: Completeness check failed,
a (database) record is missing and might have been deleted), 1 missing items, 3
items missing in repository
<<< Security Report: GAP detection check completed at Thu Mar 01 09:15:26 2018
+++ Security issue detected (x004A0110 (W: Completeness check failed, a
(database) record is missing and might have been deleted)), see logfile
'mpo_secchk.log'.
+++ Security Report: Security check tool completed at Thu Mar 01 09:15:26 2018
```

Again, the errors are logged and can be found in the configured log file.

12.2 Security Configuration Reporting Tool - mpo_secrep

The **Security Configuration Reporting Tool** analyzes the system configuration regarding possible security issues and delivers a report on the analysis result.

The checked security-relevant items are:

Internal Data Flow Security	SAG Connections secured by LAU Keys ? Web Application HTTPS configuration?
Backoffice Data Flow Security	Do defined LCGs use security features?
Security Policies	Enough secure settings?
Transaction Business Controls	CBT Session Schedules?
Logical Access Control	
Software Integrity	Integrity and integrity settings for Client and Server Software enough?
Database Integrity	Database Integrity Configuration, Gap Checking Configuration?
Additional Security Configuration	Behaviour on Security Issue (Client and Server)?

Usage:

```
mpoSecRep [<options>]
```

Options:

-SZ:<filepath>	Path to the security file security zip.
-API:<filepath>	Path to the client configuration file configuration.properties that should be used.
-APISIG:<filepath>	Path to the configuration.properties signature file hat should be used. Overrides the signature file in the security zip.
[-REPLFN:<filepath>]	Path to the replacement file that should be used.
[-SERVERCFG:<filepath>]	Path to the server main configuration file.
[-MPOBASE:<filepath>]	Path to the server root directory ($MPO_BASE). Must be defined if parameter [MPO_SERVER]/'MODULE_CONFIG' in the main server file is neither empty nor starts with '$DB:'
[-WEBSERVERXML:<filepath>]	Path to the server.xml of the Tomcat installation (not of the BOX web application!). (Other web application server not yet implemented)If omitted, all checks regarding the configuration will be skipped.

Below you can see an excerpt from a Security Configuration Report as delivered by the tool:

```
////////////////////////////////////////////////////////////////////
////     mpoSecRep - Security Configuration Reporting Tool      ////
////                  for BOX Messaging Hub                     ////
////     Version: V3R22                                         ////
////     Report Created: 2018-02-07 17:51:43                    ////
////         All checks are referenced according to the         ////
////    SWIFT Customer Security Controls Framework 1.0 (CSCF)   ////
////              published on 31st of March 2017               ////
////////////////////////////////////////////////////////////////////

Startup parameters:
  Used configuration.properties: ../mpoTransfer/configuration.properties.test
  Used configuration.properties signature file:
../mpoTransfer/configuration.properties.test.sig.xml
  Used replacement file: /opt/box/server22/config/replace.rpl
  Used security zip file: /opt/box/server22/security/security.zip
  Used server configuration cfg file: /opt/box/server22/config/mposerver.cfg
  Used web server configuration file: - none -
  Used MPO_BASE: - none -
_________________________________

Category 1: Internal Data Flow Security (Mandatory / CSCF Ref 2.1)
  Check 1.1: All SAG Connections must be protected by LAU Keys.
  Result: PASSED
  ------------------------------------
  Check 1.2: The web application can only be reached via HTTPS protected connections.
  Result: SKIPPED - Path to server.xml is not defined.
_________________________________

Category 2: Backoffice Data Flow Security (Advisory / CSCF Ref 2.4A)
  Check 2.1: Defined LCGs must use security features when applicable.
    - LAU Key for 'LCG<MQFINLLOOP>.F002' is not defined.
    - LAU Key for 'LCG<RAWWIRE>.F002' is not defined.
  Result: FAILED - 2/3 LCG are not using available security features.
```

```
    _________________________________

    Category 3: Prevent Compromise of Credentials (Mandatory / CSCF Ref 4.1 + 4.2)

      Check 3.1: Analyse Security Policies.

    [...]

        - Security Policy 'DefSecPol_BANK55' of Client Prefix 'BANK55' ( affecting 10
    user(s) ):
          - Authentication Method: MPOPASSWD
          - DFA Mode: NONE
            - [OK  ] Minimum password length is set to 1 character(s).
            - [OK  ] Abandoned password list length is set to 5.
            - [OK  ] Unique login is enabled.
            - [OK  ] Maximum number of consecutive login failures is set to 3.
            - [OK  ] Maximum password age is set to 30 day(s).
            - [WARN] Maximum inactivity period before a user is locked out on next login
    attempt is not set.
            - [OK  ] Session Timeout is set to 9000 second(s).
            - [WARN] Password complexity is not/only partially set. Passwords must
    contain at least: 0 uppercase letters, 0 lowercase letters, 0 special characters, 0
    digits.

        - Security Policy 'ldap opts' of Client Prefix 'Bank1' ( affecting 1 user(s) ):
          - Authentication Method: LDAP
            - [OK  ] This policy uses attributes outside of MPO scope.
    [...]

        12/15 policies are using MPO Password authentication.
        0/15 policies are using SSO or MPO Password authentication.
        2/15 policies are using RADIUS Authentication.
        0/15 policies are using Single-Sign-On.
        1/15 policies are using LDAP.

        Of all non-SSO security policies, 0 are using Dual Factor Authentication.

      Result: WARNINGS - 12/15 Security Policies have warnings that require Customer
    inspection.

    _________________________________

    Category 4: Transaction Business Controls (Advisory / CSCF Ref 2.9A)
      Check 4.1: Report CBT Session Schedules.
        - Session scheduling for LT Session of channel 'ZYSAZYSA':
          - [INFO] Mode: UNDEFINED
        - Session scheduling for LT Session of channel 'PTSADESA_RMA!X':
      - [INFO] Mode: UNDEFINED
    - Session scheduling for LT Session of channel 'PTSADESB_FIA':
      - [INFO] Mode: UNDEFINED
    - Session scheduling for LT Session of channel 'PTSADESB_MX':
      - [INFO] Mode: UNDEFINED
  Result: INFO - Customer is advised to inspect the connection schedules.
_________________________________

Category 5: Logical Access Control (Mandatory / CSCF Ref 5.1)
  Check 5.1: Report Logical Access Control.
    - [INFO] Customer is advised to generate an UPM Report with mpoTransfer to get a
detailed overview of logical access control.
  Result: SKIPPED - Nothing to check
_________________________________

Category 6: Software Integrity (Mandatory / CSCF Ref 6.2)
    Check 6.1: Check Server Software Integrity Settings.
    - [INFO] Parameter 'BASE_SECRET_LIST' is not set. The BSL will be read from the
security zip file defined by server config parameter 'SECURITY_ZIP_FILE'.
    - [OK  ] Parameter 'SECURITY_ZIP_FILE' points to a zip file located at:
security/security.zip.
    - [OK  ] Parameter 'SUPPRESS_CODE_SECURITY_CHECK' is not defined. Default value is:
FALSE. Code integrity check will be executed by the server.
    - [OK  ] Parameter 'CODE_SECURITY_CHECK_STARTING_HOUR' is not defined. Default value
is: 12. Code integrity check will be executed in the hour after 01:00.
    - [OK  ] Parameter 'SUPPRESS_CODE_SIGNATURE_VERIFICATION_ALERTS' is not defined.
Default value is: FALSE. Alerts regarding 'Code Signature Verification' will be generated.
    - [OK  ] Parameter 'SUPPRESS_CONFIGURATION_SIGNATURE_VERIFICATION_ALERTS' is not
defined. Default value is: FALSE. Alerts regarding 'Configuration Signature Verification'
will be generated.
```

```
    - [OK  ] Parameter 'SUPPRESS_NONSTATIC_DATA_VERIFICATION_ALERTS' is not defined.
Default value is: FALSE. Alerts regarding 'Non-Static Data Verification' will be
generated.
    - [OK  ] Parameter 'SUPPRESS_STATIC_DATA_VERIFICATION_ALERTS' is not defined. Default
value is: FALSE. Alerts regarding 'Static Data Verification' will be generated.
  Result: PASSED - All relevant security alerts will be generated.
  ------------------------------------
  Check 6.2: Check Server Software Integrity.
    - [INFO] Customer is advised to use mpo_secchk tool to verify the code integrity of
server binaries.
  Result: SKIPPED - Nothing to check
  ------------------------------------
  Check 6.3: Check Client Software Integrity Settings.
    - [OK  ] The web application will generate alerts regarding 'Configuration Item
Signature Verification'.
    - [OK  ] The web application will generate alerts regarding 'Code Signing
Verification'.
    - [OK  ] The web application will generate alerts regarding 'Non-static Data
Verification'.
    - [OK  ] The web application will generate alerts regarding 'Static Data
Verification'.

  Result: PASSED - All relevant security alerts will be generated.
 ------------------------------------
  Check 6.4: Check Client/Tools Software Integrity.
    - [INFO] Customer is advised to use the icopesecurity.jar tool to verify the code
integrity of client/tool jars.
  Result: SKIPPED - Nothing to check
_________________________________

Category 7: Database Integrity (Mandatory / CSCF Ref 6.3)
  Check 7.1: Check Database Integrity Configuration.
    - [OK  ] Static Data will be protected by database row-level guarding.
    - [OK  ] Non-Static Data will be protected by database row-level guarding.

  Result: PASSED - Database row-level guarding is enabled for static and non-static data.
  ------------------------------------
  Check 7.2: Check Gap Checking Configuration.
    - [OK  ] Parameter 'DISABLE_GAP_DETECTION' is not defined. Default value is: FALSE.
Gap Detection will be executed by the BOX server.
    - [INFO] Gap Detection check is enabled and will always be performed on server module
startup.
    - [WARN] Gap Detection is activated but not scheduled in the Deleter config section!

  Result: WARNINGS - Gap Detection is not properly configured.
_________________________________

Category 8: Additional Security Configuration Checks
  Check 8.1: Behaviour on Security Issue (Client).
    - [OK  ] The web application will continue startup even if a security issue occurs.
    - [OK  ] UPM Users can login even if a security issue occurs.

  Result: PASSED - Data/Configuration security issues will be reported but do not
interfere with web application operability/user login.
  ------------------------------------
  Check 8.2: Behaviour on Security Issue (Server).
    - [OK  ] Parameter 'CONTINUE_INIT_ON_SECURITY_FAILURES' is not defined. Default value
is: TRUE. Server will continue initialization on configuration security issue.
    - [OK  ] Parameter 'CONTINUE_PROC_ON_SECURITY_FAILURES' is not defined. Default value
is: TRUE. Server will continue message processing on data security issue.

  Result: PASSED - Data/Configuration security issues will be reported but do not
interfere with module operability.
___________________________________

Report Summary
1. Internal Data Flow Security (Mandatory / CSCF Ref 2.1)
   [OK    ] 1.1 All SAG Connections secured by LAU Keys
   [SKIP  ] 1.2 Web Application HTTPS configuration
2. Backoffice Data Flow Security (Advisory / CSCF Ref 2.4A)
   [FAILED] 2.1 Defined LCGs must use security features when applicable
3. Prevent Compromise of Credentials (Mandatory / CSCF Ref 4.1 + 4.2)
   [WARN  ] 3.1 Analyse Security Policies
4. Transaction Business Controls (Advisory / CSCF Ref 2.9A)
   [INFO  ] 4.1 Report CBT Session Schedules
5. Logical Access Control (Mandatory / CSCF Ref 5.1)
   [SKIP  ] 5.1 Report Logical Access Control
6. Software Integrity (Mandatory / CSCF Ref 6.2)
```

```
   [OK    ] 6.1 Check Server Software Integrity Settings
   [SKIP  ] 6.2 Check Server Software Integrity
   [OK    ] 6.3 Check Client Software Integrity Settings
   [SKIP  ] 6.4 Check Client/Tools Software Integrity
7. Database Integrity (Mandatory / CSCF Ref 6.3)
   [OK    ] 7.1 Check Database Integrity Configuration
   [WARN  ] 7.2 Check Gap Checking Configuration
8. Additional Security Configuration Checks
   [OK    ] 8.1 Behaviour on Security Issue (Client)
   [OK    ] 8.2 Behaviour on Security Issue (Server)

------ End Of Report ------
```

13 Disclaimer

INTERCOPE GmbH
Himmelstrasse 12-16,
22299 Hamburg,
Germany

+49 40 514 52 0
info@intercope.com

https://www.intercope.com