

Integrated Technical Services



Safety First, Service Always



S-124 Service - User Guide

September 2025

Record of Amendments

Version	Date	Description	Signature
1.0	2025-08-22	Polygon coordinates updated, section about signature explanation added and results limit changed to 2000.	M.R.C
1.1	2025-09-03	Convert section 2.5 into section 3, correct some steps and pictures in subsection 3.3	M.R.C

Table of Contents

Section 1	INTRODUCTION	4
1.1	Context	4
	Objective	4
	IMPORTANT	4
Section 2	How to use the CCG S-124 service	5
2.1	GET Detailed Request	5
2.2	GET Detailed Request Parameters	7
2.2.1	productVersion	7
2.2.2	containerType	7
2.2.3	dataReference	7
2.2.4	geometry	8
2.2.5	validFrom	8
2.2.6	validTo	8
2.2.7	page	8
2.2.8	pageSize	9
2.3	How to convert the GET Detailed Response	10
2.4	GET Summary Request	12
2.4.1	dataReference	13
2.4.2	info_identifier	13
2.4.3	Info_name	13
2.4.4	info_status	13
2.4.5	info_description	13
2.4.6	info_productVersion	14
Section 3	Signature Validation	15
3.1	Validation of the S-100 exchange set signatures	15
3.2	Validation of the request response signatures (in the exchangeMetaData field)	22
3.3	Validation of the catalog.sign signature (in the S-100 exchange set)	30

Section 1 INTRODUCTION

1.1 CONTEXT

The S-124 service has been designed to distribute the Navigational Warnings (NAVWARNs) produced by the Canadian Coast Guard, based on the S-124 standard and following the Secure Exchange of Maritime Information (SECOM) security protocol.

OBJECTIVE

The objective of this document is to describe how to use the S-124 service for end users such as S-100 data distributors and system developers. This service ***is not about downloading selected files or charts***; it implies the use of a REST API where HTTP methods are used to request data, the output is subject to further post-processing.

This document includes examples you can follow if you wish to discover this S-124 service and its output.

IMPORTANT

This S-124 service is still under development and represents our best efforts to meet the actual S-124 / S-100 specifications. Until advised otherwise, users should use this service for testing purposes only. Changes to this service may occur without prior notice.

The 'exchangeMetadata' field contains information such as the digital signature of the data field content, the public certificate to validate the signature and the signature algorithm used (see figure 2.1).

The request returns the S-124 NAVWARNs as an S-100 Exchange set (.zip file), when the parameter containerType=1 is used (see below for more information).

Example:

The request below will retrieve the active NAVWARNs for the Canadian S-100 Sea trials area, the output being packaged as a S-124 2.0.0 exchange set :

[https://s124.ccg-gcc.gc.ca/api/secom/v1/object?geometry=POLYGON\(\(-73.6%2045.5,%20-73.5%2045.5,%20-73.2%2045.8,%20-73.2%2046.0,%20-72.6%2046.0,%20-69.8%2047.2,%20-71.0%2047.2,%20-73.5%2046.1,%20-73.5%2045.7,%20-73.6%2045.7,%20-73.6%2045.5\)\)&containerType=1&productVersion=2.0.0](https://s124.ccg-gcc.gc.ca/api/secom/v1/object?geometry=POLYGON((-73.6%2045.5,%20-73.5%2045.5,%20-73.2%2045.8,%20-73.2%2046.0,%20-72.6%2046.0,%20-69.8%2047.2,%20-71.0%2047.2,%20-73.5%2046.1,%20-73.5%2045.7,%20-73.6%2045.7,%20-73.6%2045.5))&containerType=1&productVersion=2.0.0)

The output contains useful information, when you wish to check the syntax or validity of your requests. Simply scroll to the very end of the output block to find lines that look like:

```
"responseText": "Get Request of NWs returned 1 results with the following filters: dataReference=null, containerType=S100_ExchangeSet , validFrom=null, validTo=null, dataProductType=null, productVersion=2.0.0, geometry=POLYGON((-73.6 45.5, -73.5 45.5, -73.2 45.8, -73.2 46.0, -72.6 46.0, -69.8 47.2, -71.0 47.2, -73.5 46.1, -73.5 45.7, -73.6 45.7, -73.6 45.5)), page=null, pageSize=null, max pageSize is 2000"
```

2.2 GET DETAILED REQUEST PARAMETERS

The URL for the GET detailed request is <https://s124.ccg-gcc.gc.ca/api/secom/v1/object> and the following parameters can be used. Request parameters are separated with the '&' character.

2.2.1 productVersion

<https://s124.ccg-gcc.gc.ca/api/secom/v1/object?productVersion=2.0.0>

The **productVersion** parameter sets the version of the S-124 output. The default version is 2.0.0. The supported versions are 1.0.0, 1.5.0 and 2.0.0.

2.2.2 containerType

<https://s124.ccg-gcc.gc.ca/api/secom/v1/object?containerType=1>

The **containerType** parameter can be used if you want to get the results individually or packaged in a S-100 Exchange set (output is a .zip file).

When **containerType=1**, a single data field is created as a .zip file (encoded in base 64) containing the S-100 Exchange set.

If **containerType=0** (default value), you get one data field for each S-124 NAVWARN.

2.2.3 dataReference

<https://s124.ccg-gcc.gc.ca/api/secom/v1/object?dataReference=00000000-0000-0000-0000-000000171894>

The **dataReference** parameter can be used when you know the ID of the specific S-124 NAVWARN you want to get. The dataReference id is passed with the format XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX.

The GET Summary request (see section 2.4) outputs the dataReference value for each S-124 NAVWARN returned.

2.2.4 geometry

[https://s124.ccg-gcc.gc.ca/api/secom/v1/object?geometry=POLYGON\(\(-73.6%2045.5,%20-73.5%2045.5,%20-73.2%2045.8,%20-73.2%2046.0,%20-72.6%2046.0,%20-69.8%2047.2,%20-71.0%2047.2,%20-73.5%2046.1,%20-73.5%2045.7,%20-73.6%2045.7,%20-73.6%2045.5\)\)](https://s124.ccg-gcc.gc.ca/api/secom/v1/object?geometry=POLYGON((-73.6%2045.5,%20-73.5%2045.5,%20-73.2%2045.8,%20-73.2%2046.0,%20-72.6%2046.0,%20-69.8%2047.2,%20-71.0%2047.2,%20-73.5%2046.1,%20-73.5%2045.7,%20-73.6%2045.7,%20-73.6%2045.5)))

The **geometry** parameter can be used if you want to get the results from one particular area defined by a point, polyline or polygon. You can use a geometry defined in the WKT language (https://en.wikipedia.org/wiki/Well-known_text_representation_of_geometry).

We recommend using POLYGON, LINESTRING or POINT.

Note: The POLYGON example above defines the area for the Canadian S-100 Sea Trials.

2.2.5 validFrom

<https://s124.ccg-gcc.gc.ca/api/secom/v1/object?validFrom=20250101T000000>

The **validFrom** parameter can be used if you want to get the S-124 NAVWARNs published since the Timestamp value. When the **validFrom** parameter is used, the results will include the ('self-cancelled') canceller S-124 NAVWARNs. Also, when the **validFrom** parameter is used, the results will be returned as a page among pages.

2.2.6 validTo

<https://s124.ccg-gcc.gc.ca/api/secom/v1/object?validTo=20250101T000000>

The **validTo** parameter can be used if you want to get the S-124 NAVWARNs published before the Timestamp value. When the **validTo** parameter is used, the results will include the 'self-cancelled' S-124 NAVWARNs. Also, when the **validTo** parameter is used, the results will be returned as a page among pages.

2.2.7 page

<https://s124.ccg-gcc.gc.ca/api/secom/v1/object?pageSize=10&page=1>

The **page** parameter outputs the results as a page among pages.

To see how many pages are returned, the number of pages is shown in the *pagination* field at

2.3 HOW TO CONVERT THE GET DETAILED RESPONSE

As explained in section 2.1, **the content of the 'data' field is encoded in base 64.**

When **contentType=1** (S-100 Exchange set), you must follow the following steps to convert the result in a human readable text:

- 1) Copy the content (it is base64 encoded text) from the 'data' field. Omit the enclosing “.



Figure 2.3

- 2) Paste the base64 encoded text in the text box from the following web site (or any tool that can convert base64 text content into a file): <https://base64.guru/converter/decode/file> and click on the "Decode Base64 to File" button. Click on the application.zip link to download the .zip containing the S-100 Exchange set. See figure 2.4.

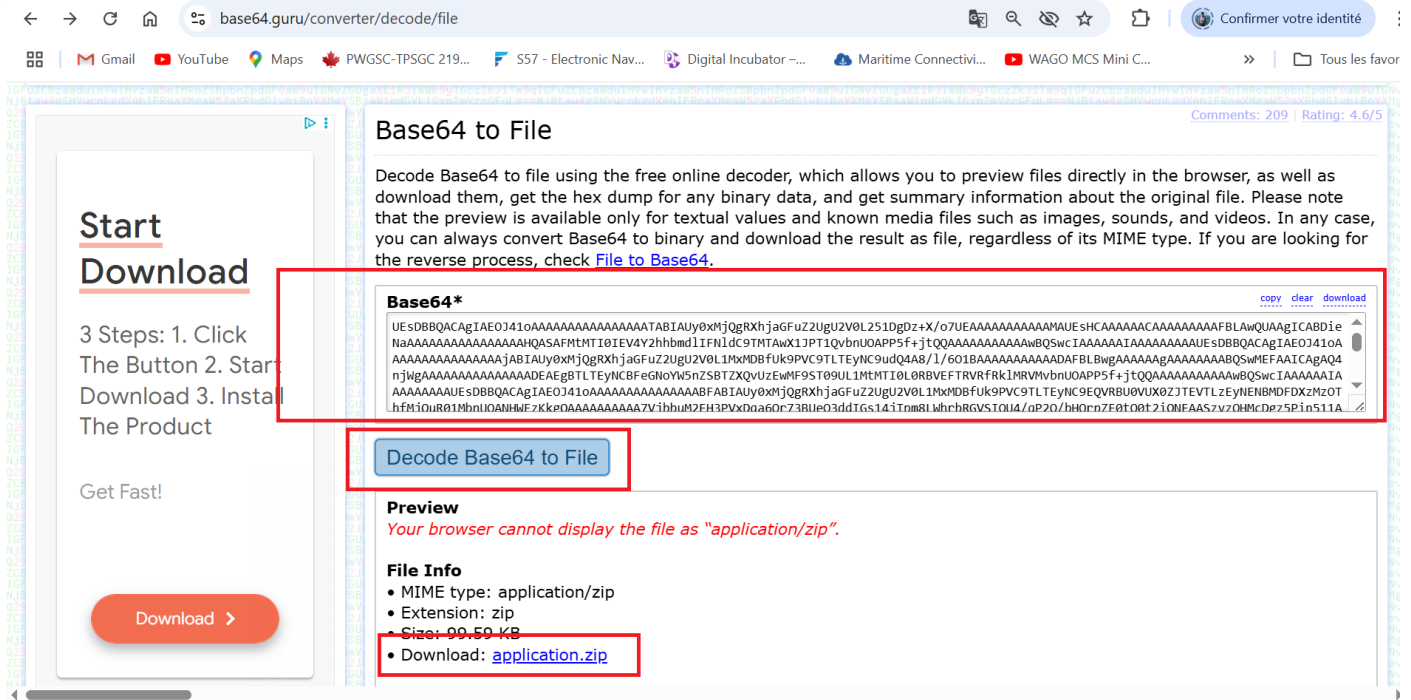


Figure 2.4

1) If the **containerType** parameter is not used, or is set to 0, you'll get individual S-124 NAVWARN records.

To convert them, follow these steps: For each 'data' field, copy and paste its value. See figure 2.5.



Figure 2.5

- 2) Copy and paste the base64 encoded text in the text box from the following web site (or a tool that can convert base64 text content into a file) <https://www.base64decode.org/> and click on the “Decode” button. The converted result will be displayed in the second text box. See figure 2.6.

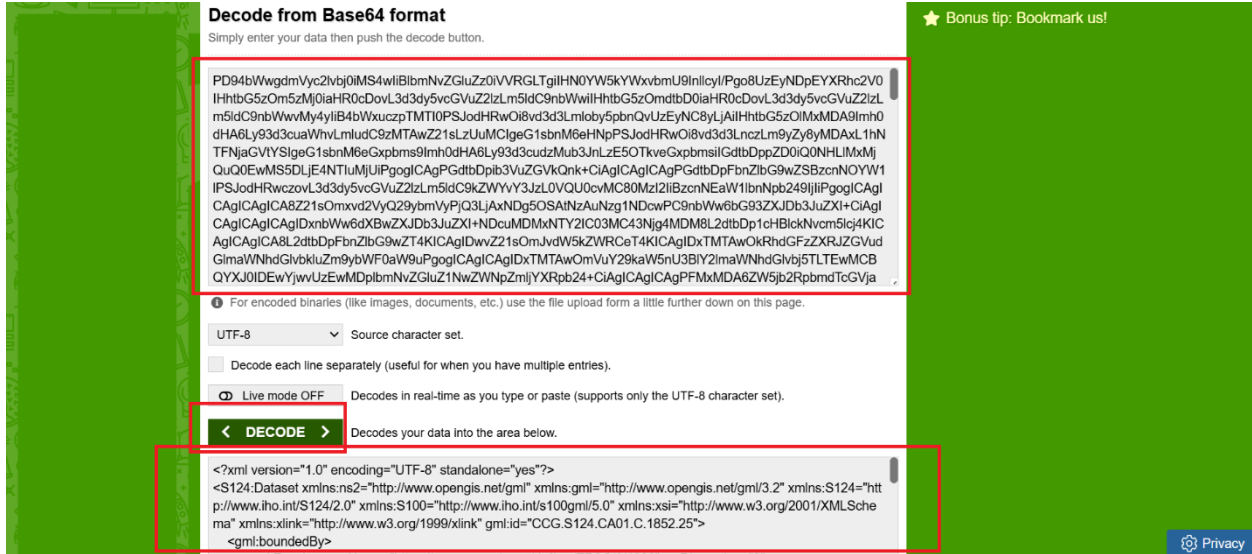


Figure 2.6

2.4 GET SUMMARY REQUEST

The GET Summary request returns the list of the current active S-124 NAVWARNs. The URL for the GET Summary request is <https://s124.ccg-gcc.gc.ca/api/secom/v1/object/summary> .

The fields returned by the GET Summary request are described below :

```

{
  "summaryObject": [
    {
      "dataReference": "00000000-0000-0000-0000-000000172455",
      "dataProtection": false,
      "dataCompression": false,
      "containerType": 0,
      "dataProductType": "S124",
      "info_identifier": "NW-C-2153-25",
      "info_name": "Offshore Works",
      "info_status": "PUBLISHED",
      "info_description": "Service barge Silt King at 42 51.252N 079 31.816W and will be flaring natural gas periodically.\nStay clear.",
      "info_lastModifiedDate": "20250718T123425Z",
      "info_productVersion": "1.5.0",
      "info_size": 6414
    },
    {
      "dataReference": "00000000-0000-0000-0000-000000172447",
      "dataProtection": false,
      "dataCompression": false,
      "containerType": 0,
      "dataProductType": "S124",
      "info_identifier": "NW-C-2150-25",
      "info_name": "Aids to Navigation",
      "info_status": "PUBLISHED",
      "info_description": "Unreliable, reduced intensity and improper characteristics.",
      "info_lastModifiedDate": "20250718T021224Z",
      "info_productVersion": "1.5.0",
      "info_size": 5278
    }
  ]
}

```

Figure 2.7

2.4.1 dataReference

The **dataReference** value can be used with the **dataReference** parameter in the GET Detailed request, to retrieve a specific S-124 NAVWARN.

2.4.2 info_identifier

The **info_identifier** field is associated with the S-124 NAVWARN series identifier.

2.4.3 Info_name

The **info_name** field is associated with the S-124 NAVWARN title.

2.4.4 info_status

The **info_status** field is associated with the status of the S-124 NAVWARN.

2.4.5 info_description

The **info_description** field is associated with the description of the S-124 NAVWARN.

2.4.6 info_productVersion

The **info_productVersion** is associated with the version of the S-124 product specification standard.

Section 3 Signature Validation

3.1 VALIDATION OF THE S-100 EXCHANGE SET SIGNATURES

To validate the signature associated with a given S124 .gml file contained in a S-100 exchange, you have to follow these steps (Step 2 and step 3 are for intermediate/advanced users, we suggest to directly go to step 4 if you are not a developer or an experienced user):

- 1) extract the certificate from the catalog.xml file first. To do it, you have to copy the certificate value contained in the "<certificate>" tag (see figure 3.1).

```

<ns6:MD_CharacterSetCode codeList="http://www.iana.org/assignments/character-sets" codeListValue="UTF-8">UTF-8</ns6:MD_CharacterSetCode>
</ns6:characterEncoding>
</ns6:PT_Locale>
</ns2:defaultLocale>
▼<ns2:exchangeCatalogueDescription>
  <ns3:CharacterString>S124 Exchange Set of active canadian NAVWARNs</ns3:CharacterString>
</ns2:exchangeCatalogueDescription>
▼<ns2:exchangeCatalogueComment>
  <ns3:CharacterString/>
</ns2:exchangeCatalogueComment>
▼<ns2:certificates>
  <schemeAdministrator id="Canadian Coast Guard"/>
  <certificate id="cersccg124" issuer="1.2.840.113549.1.9.1=#161d696e666f406d61726974696d65636f6e6e65637469766974792e6e6574, CN=MCP Identity
  Registry, OU=MCP, O=MCP, L=Copenhagen, ST=Denmark, C=DK, UID=urn:mfn:mcp:ca:mcc:mcp
  idreg">TULJRCqQqNBNEInQxdJQkFnsVVKQTArYkRtMmJKLzVwNXLXQ2VGQmVUTJLVVV3Q2dZSutvWk16ajBFQXdNd2jd3hMREFxQndvSmtPUpRl0z1zkFFQkRceDFjbtQ2Y1hKdu9tMwpjRHBqWVRwdFkyTTZiV053TF
  </certificate>
</ns2:certificates>
</ns2:dataServerIdentifier>https://s124.ccg-gcc.gc.ca/</ns2:dataServerIdentifier>
▼<ns2:datasetDiscovery/Metadata>
  ▼<ns2:S100_DatasetDiscovery/Metadata>
    <ns2:fileName>file:/124CA01C_2446_25.GML</ns2:fileName>
    ▼<ns2:description>
      <ns3:CharacterString>Montréal to Trois-Rivières - Montreal to Sorel</ns3:CharacterString>
    </ns2:description>
    <ns2:datasetID>urn:mfn:ccg:s124:CA01:C.2446.25</ns2:datasetID>
    <ns2:compressionFlag>false</ns2:compressionFlag>
    <ns2:dataProtection>false</ns2:dataProtection>
    <ns2:protectionScheme>S100p15</ns2:protectionScheme>
    <ns2:digitalSignatureReference>ECCDSA-384-SHA2</ns2:digitalSignatureReference>
    ▼<ns2:digitalSignatureValue>
      <S100_SE_DigitalSignature id="file:/124CA01C_2446_25.GML"
      certificateRef="cersccg124">MGQCMG688Tpm2JCHAoPQtZTCqL4zRq0jIf7S8Cc0ora3DB8aokfTDG17Q5NXcnhCSQdgiWnJKp/jxfs2tq4EYXk65J0e/VJ3Br041fmIF7w8kH/ZsPy9cMmbT91b8XyW2wHuMg<
    </ns2:digitalSignatureValue>
    <ns2:copyright>true</ns2:copyright>
  ▼<ns2:classification>
    <ns14:MD_ClassificationCode codeList="https://standards.iso.org/iso/19115/resources/Codelists/cat/codelists.xml"
    codeListValue="1">unclassified</ns14:MD_ClassificationCode>
  </ns2:classification>

```

Figure 3.1

- 2) (Optional) Decode the certificate value because the certificate value in the catalog.xml is encoded in Base64. The 2 following options can be used:
 - a) with the following online tool <https://www.base64decode.org/> :

- 4) Validate the signature with the public key and the signature value. To validate the signature, you can use an online tool (option a) or a programming language like Java (option b).
- a) If you have skipped step 2 and step 3, copy the public key extracted from the CCG public certificate (the content of the whole following text).

-----BEGIN PUBLIC KEY-----

```
MHYwEAYHKoZlZjOCAQYFK4EEACIDYgAENyUVuPgUthLoOrY8BcSBcVrKyQmSRlpQ
5vnKN3aFq8WBKQ9cEVWltXeA79vk6Q531cxtDWcaS28dDFtsd7Od+QJaUeS5GPwr
E4GCSNe9XX85H4Y7+RXqzrCWac1j6n0/
```

-----END PUBLIC KEY-----

Go to the following web site <https://emn178.github.io/online-tools/ecdsa/verify/> and select ECDSA->Verify Signature

- i) Select UTF-8 for the Input Encoding (see figure 3.8);
- ii) Add the content of the file for which you want to validate the signature in the Input text area (see figure 3.8). In this example, we take the textual content of the file 124CA01C_2446_25.GML (see figure 3.9);
- iii) Select SECG secp384r1 / X9.63 ansip384r1 / NIST P-384 for the Curve option (see figure 3.8);
- iv) Select SHA384 for the Signature Algorithm (see figure 3.8);
- v) Select Pem Text for the Public Key Type (see figure 3.10);
- vi) Add the Public Key content (surrounded by -----BEGIN PUBLIC KEY----- and -----END PUBLIC KEY-----) in the Public Key Data text area (see figure 3.10);
- vii) Select Base64 for the Signature Type (see figure 3.11);
- viii) Add the content of the “<S100_SE_DigitalSignature>” tag associated with the file signature to validate (124CA01C_2446_25.GML in this example, see figure 3.12) in the Signature Data text area (see figure 3.11);
- ix) Click on verify button (see figure 3.8).

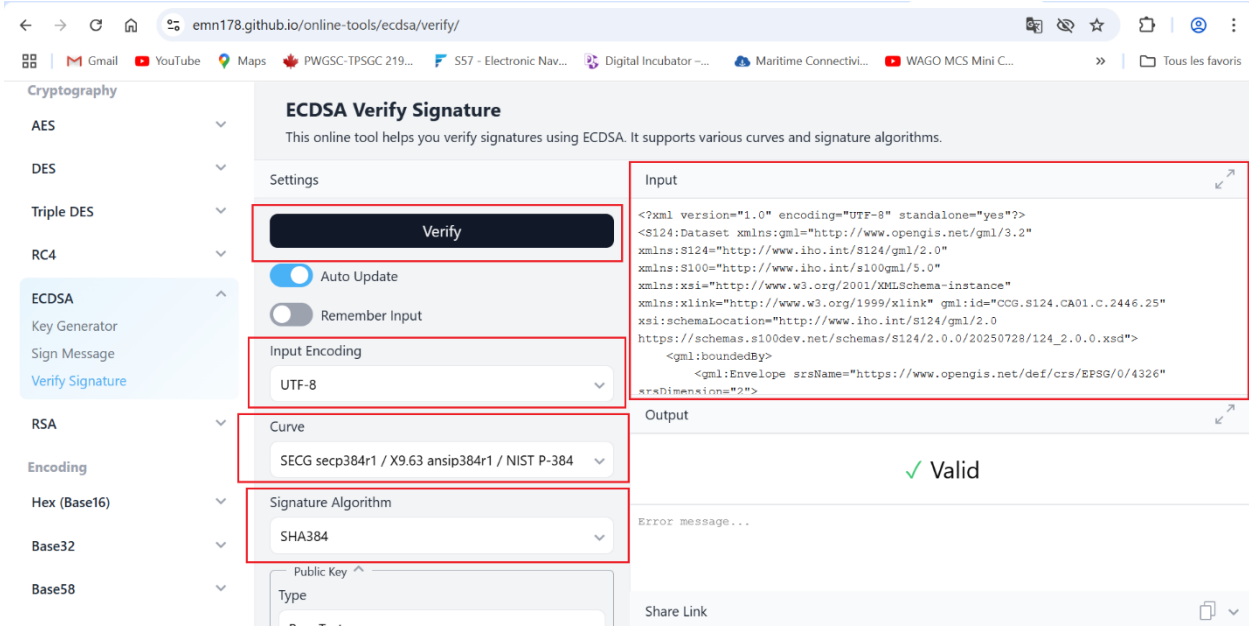


Figure 3.8

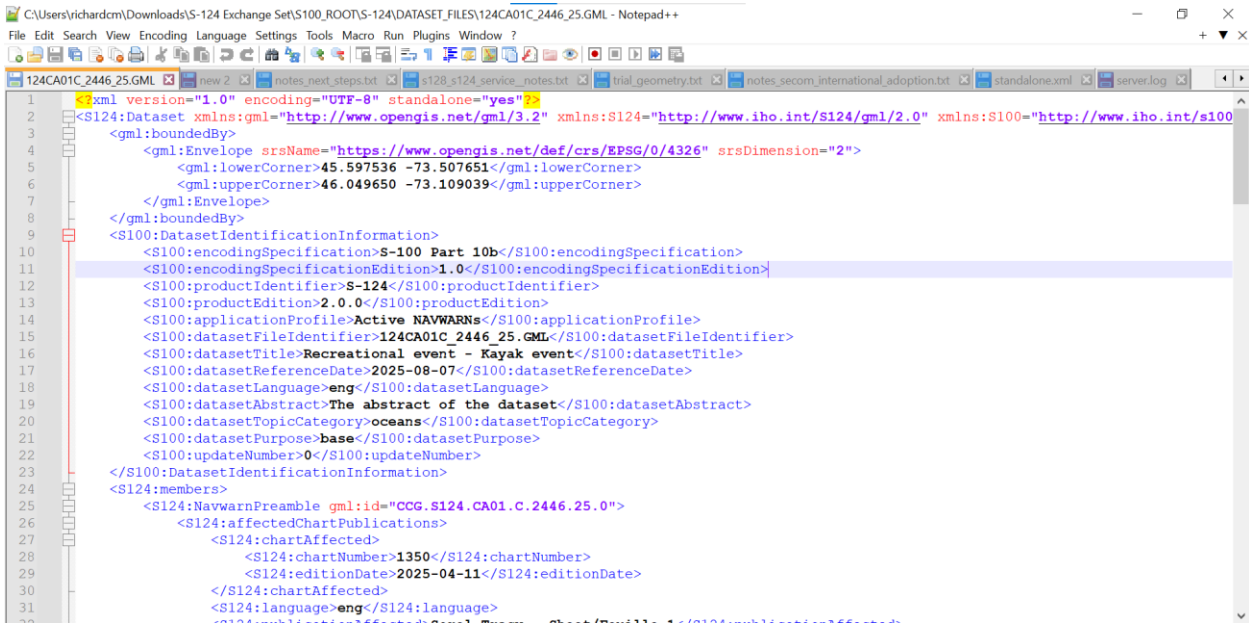


Figure 3.9



Figure 3.10

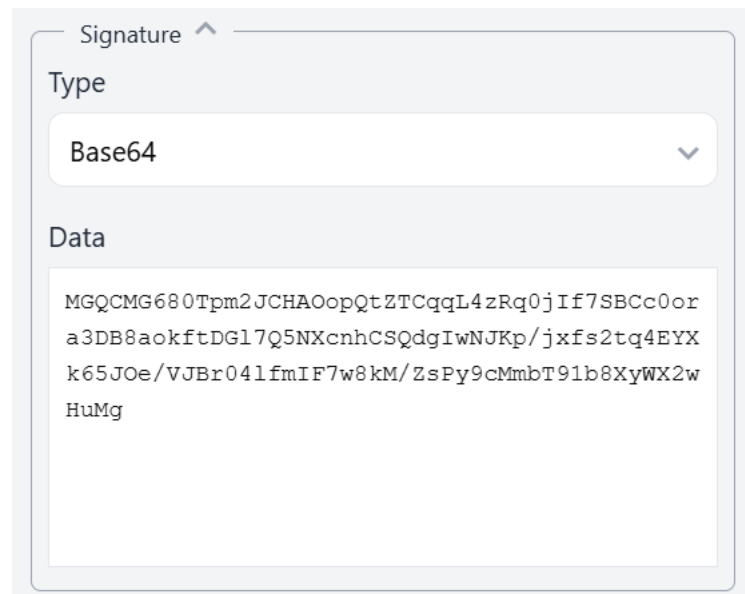


Figure 3.11

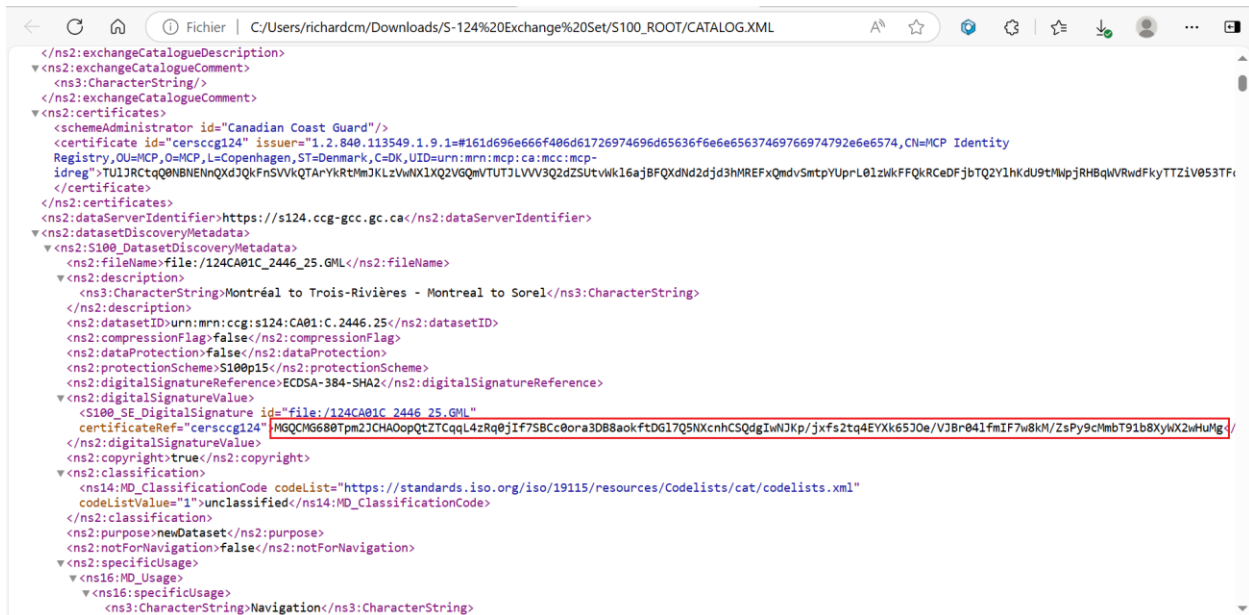


Figure 3.12

- b) In `Java`,
- Assign the signature value (figure 3.12) to the `(String)` signature variable;
 - Assign the encoded certificate value (figure 3.1) to the `(String)` `encodedPublicCertificate` variable;
 - Assign the textual content (figure 3.9) of the file associated with the signature to the `(String)` content variable;
 - execute the following commands (use algorithm 'SHA384WITHECDSA'):

```
byte[] decodedPublicCertificate = Base64.getDecoder().decode(encodedPublicCertificate);
byte[] publicCertificateValue = Base64.getDecoder().decode(decodedPublicCertificate);
X509Certificate certificate = (X509Certificate) CertificateFactory.getInstance("X.509").generateCertificate(new
ByteArrayInputStream(publicCertificateValue));
PublicKey publicKey = certificate.getPublicKey();

Signature ecdsaSignature = Signature.getInstance("SHA384withECDSA");
ecdsaSignature.initVerify(publicKey);
ecdsaSignature.update(content.getBytes());
boolean isValid = ecdsaSignature.verify(Base64.getDecoder().decode(signature.getBytes()));
```

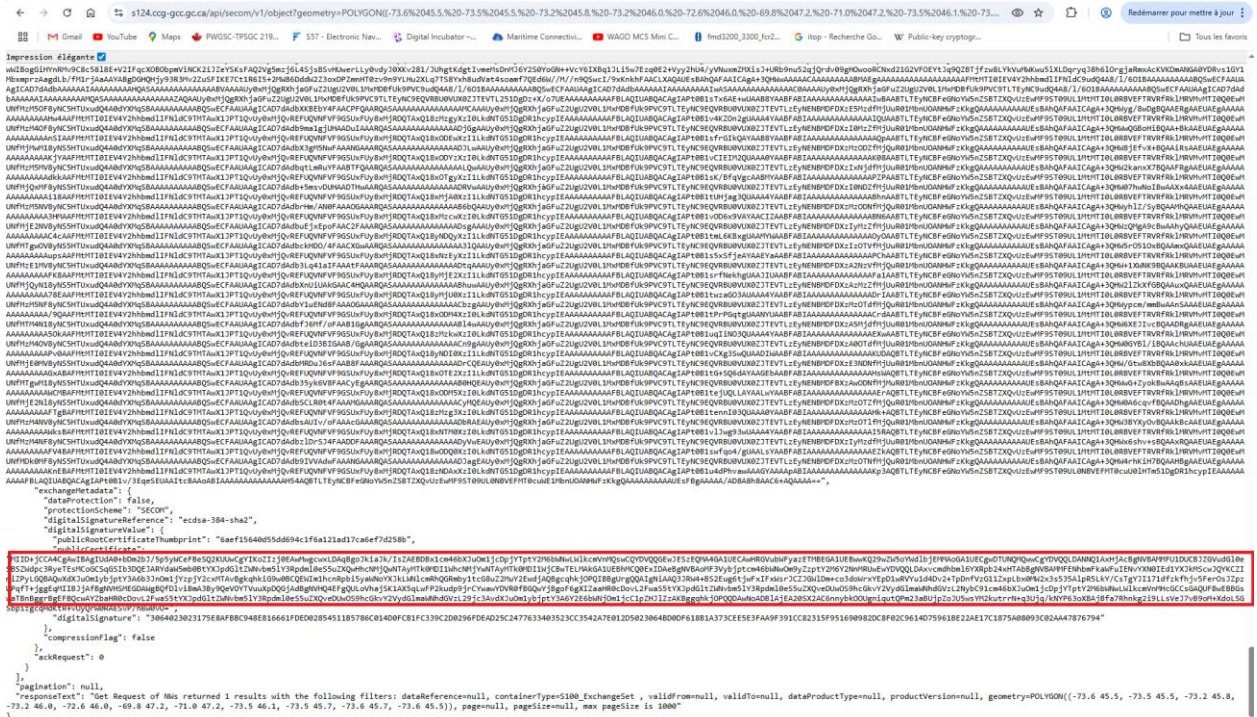



Figure 3.15

The certificate value is already decoded, so step 2 from the previous section is not required.

2) (Optional) Using the certificate value from the request response, extract the public key from the certificate. The two following options can be used:

- a) Put the decoded certificate value in a file named `cg_certificate.pem` and surround it by `-----BEGIN CERTIFICATE-----` and `-----END CERTIFICATE-----` (see figure 3.16)

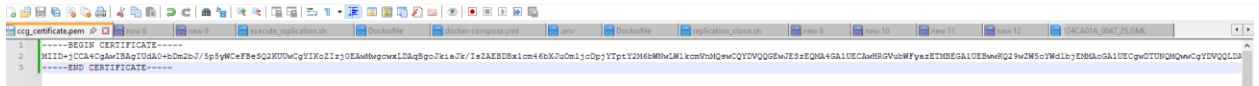


Figure 3.16

Execute the following openssl command in WSL (Linux environment in Windows) or in a Linux terminal:

```
openssl x509 -pubkey -noout -in cg_certificate.pem > ec-secp384-r1-pub-key-extracted.pem (see figure 3.17).
```

That command will create the file `ec-secp384-r1-pub-key-extracted.pem` containing the public key (see figure 3.18).

```
richardcm@QLABDEVW06: ~
richardcm@QLABDEVW06: $ openssl x509 -pubkey -noout -in ccg_certificate.pem > ec-secp384-r1-pub-key-extracted.pem
```

Figure 3.17

```
-----BEGIN PUBLIC KEY-----
MHYwEAYHKoZIzj0CAQYFK4EEACIDYgAERY0VuFgUthLoOrY8BoSBvRyQm58lpQ
3vuR27aFqI9B3KQoIVWL1XeA79vX6G53LxtdWoaS28dDFtad70d+Q7a5e55G9wz
E4GCB9e9XES8V47v83lgaCWao1j6nD/
-----END PUBLIC KEY-----
```

Normal text file length: 215 lines: 6 Ln: 1 Col: 1 Pos: 1 Unix (LF) UTF-8 INS

Figure 3.18

- b) Use a programming language (like Java with the `java.security` library in the following example) to load the decoded certificate and extract the public key (see figure 3.19).

```
X509EncodedKeySpec publicKeySpec = new X509EncodedKeySpec(decodedCertificateValue.getBytes());
PublicKey publicKey = keyFactory.generatePublic(publicKeySpec);
```

Figure 3.19

- 3) Validate the signature with the public key and the signature value. To validate the signature, you can use an online tool (option a) or a programming language like Java (option b).

- a) If you have skipped step 2, copy the public key extracted from the CCG public certificate (the content of the whole following text).

-----BEGIN PUBLIC KEY-----

```
MHYwEAYHKoZlZjOCAQYFK4EEACIDYgAENyUVuPgUthLoOrY8BcSBcVrKyQmSRlpQ
5vnKN3aFq8WBKQ9cEVWltXeA79vk6Q531cxtDWcaS28dDFtsd7Od+QJaUeS5GPwr
E4GCSNe9XX85H4Y7+RXqzrCWac1j6n0/
```

-----END PUBLIC KEY-----

Go to the following web site <https://emn178.github.io/online-tools/ecdsa/verify/> and select ECDSA->Verify Signature

- i) Select Base64 for the Input Encoding (see figure 3.20);
- ii) Copy the content of the data field (figure 3.21) associated with the signature to validate and add it to the Input text area (see figure 3.20);
- iii) Select SECG secp384r1 / X9.63 ansip384r1 / NIST P-384 for the Curve option (see figure 3.20);
- iv) Select SHA384 for the Signature Algorithm (see figure 3.20);
- v) Select Pem Text for the Public Key Type (see figure 3.22);
- vi) Add the Public Key content (surrounded by -----BEGIN PUBLIC KEY----- and -----END PUBLIC KEY-----) in the Public Key Data text area (see figure 3.22);
- vii) Select HEX for the Signature Type (see figure 3.23). The signature format in the request responses are encoded in HEX instead of Base64;
- viii) Copy the content of the exchangeMetadata->digitalSignatureValue->digitalSignature field associated with the data that is being validated (In this example, see figure 3.24) and add it to the Signature Data text area (see figure 3.23);
- ix) Click on verify button (see figure 3.20).

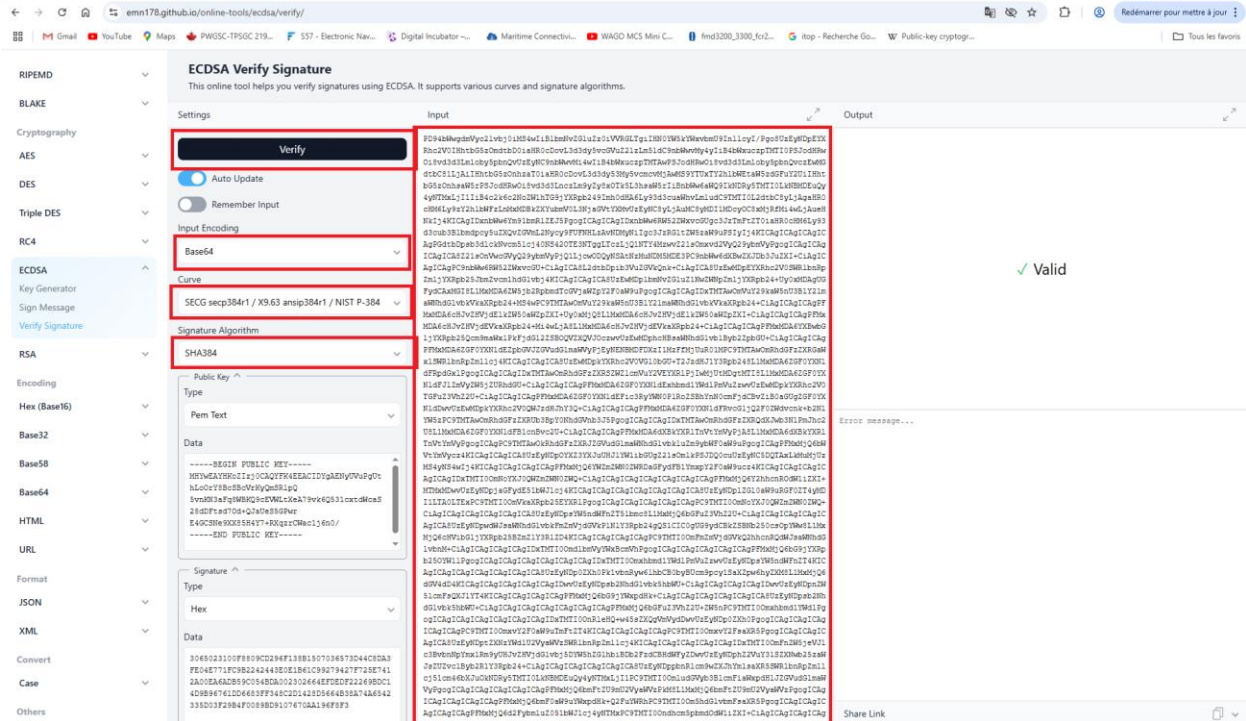


Figure 3.20

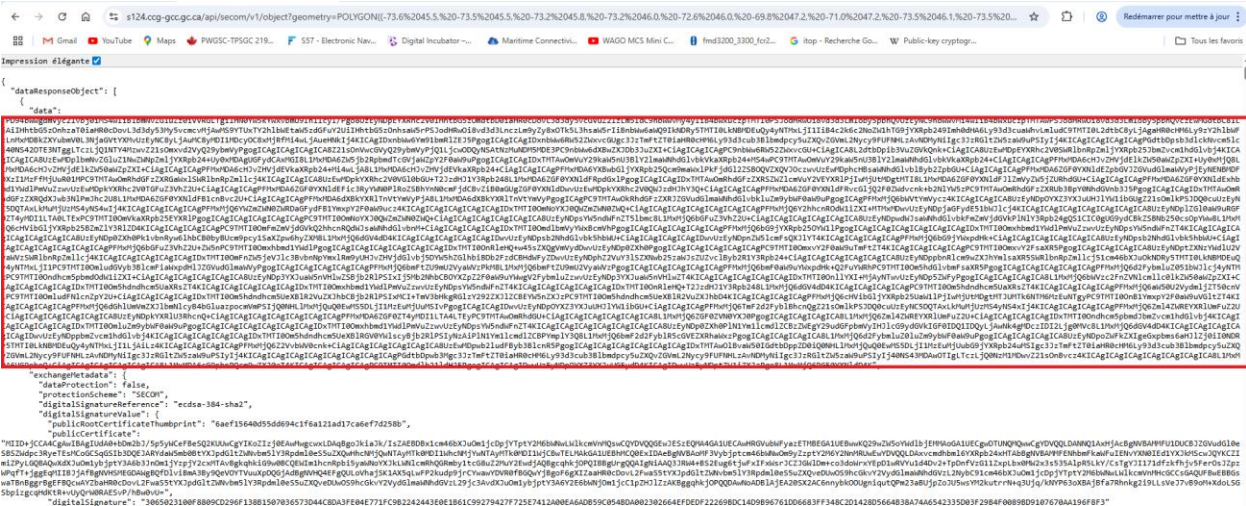


Figure 3.21

Public Key ^

Type

Pem Text

Data

```

-----BEGIN PUBLIC KEY-----
MHYwEAYHKoZIzj0CAQYFK4EEACIDYgAENyUVuPgUt
hLoOrY8BcSBcVrKyQmSRlpQ
5vnKN3aFq8WBKQ9cEVWltXeA79vk6Q531cxtDWcaS
28dDFtsd7Od+QJaUeS5GPwr
E4GCSNe9XX85H4Y7+RXqzrCWaclj6n0/
-----END PUBLIC KEY-----
    
```

Figure 3.22

Signature ^

Type

Hex

Data

```

3065023100F8809CD296F138B1507036573D44C8DA3
FE04E771FC9B2242443E0E1B61C99279427F725E741
2A00EA6ADB59C054BDA002302664EFDEDF22269BDC1
4D9B96761DD6683FF348C2D1428D5664B38A74A6542
335D03F29B4F0089BD9107670AA196F8F3
    
```

Figure 3.23

3.3 VALIDATION OF THE CATALOG.SIGN SIGNATURE (IN THE S-100 EXCHANGE SET)

To validate the signature contained in the catalog.sign file associated with the catalog.xml file contained in the S-100 exchange set (see figure 3.25), you have to follow these steps (Step 1 and 2 are for intermediate/advanced users, we suggest to directly go to step 4 if you are not a developer or an experienced user):

- 1) (Optional) Extract the certificate from the catalog.xml file first. To do it, you have to copy the certificate value contained in the "`<certificate>`" tag (see figure 3.26).

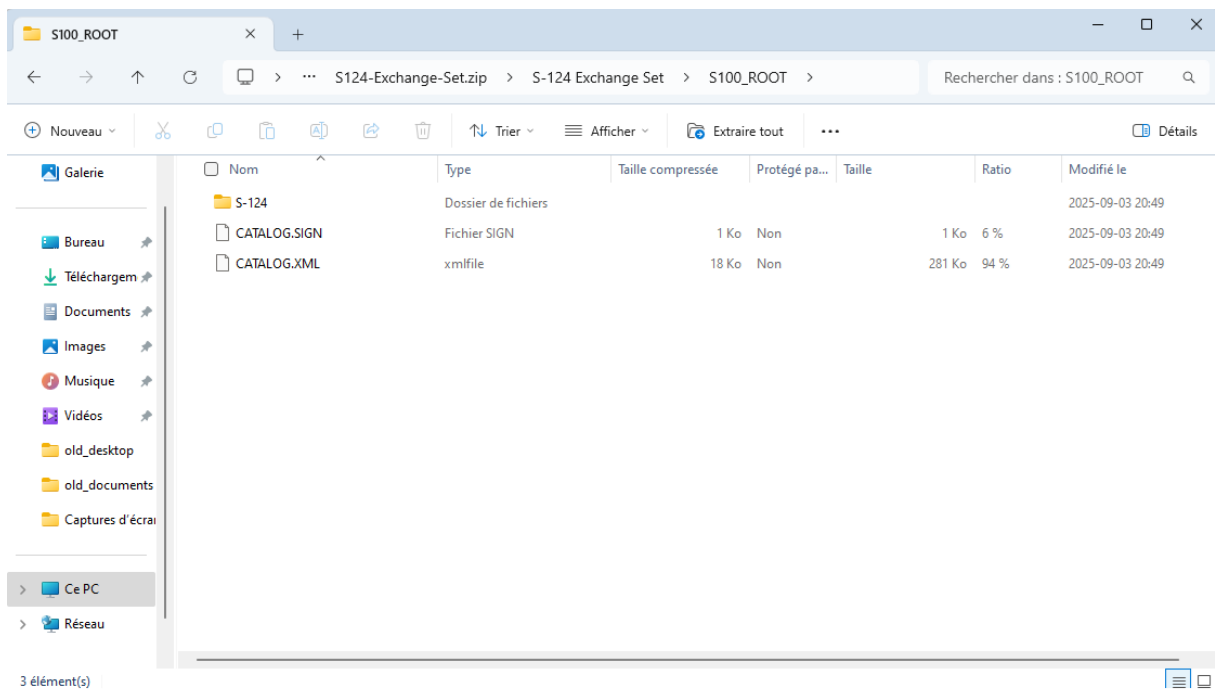


Figure 3.25

```

<ns6:MD_CharacterSetCode codeList="http://www.iana.org/assignments/character-sets" codeListValue="UTF-8">UTF-8</ns6:MD_CharacterSetCode>
</ns6:characterEncoding>
</ns6:PT_Locale>
</ns2:defaultLocale>
<ns2:exchangeCatalogueDescription>
<ns3:CharacterString>S124 Exchange Set of active canadian NAWARNs</ns3:CharacterString>
</ns2:exchangeCatalogueDescription>
<ns2:exchangeCatalogueComment>
<ns3:CharacterString/>
</ns2:exchangeCatalogueComment>
<ns2:certificates>
<schemaAdministrator id="Canadian Coast Guard"/>
<certificate id="censcgc124" issuer="1.2.840.113549.1.9.1-#161d696e6664f06d61726974696d65636f6e6e6537469766974792e6e574,CN=MCP Identity
Registry,OU=MCP,OU=MCP,L=Copenhagen,ST=Denmark,C=DK,UID=urn:mcp:censcgc:mcp
idreg">TUL1R3CQ8NBNEInQxdJQkF5vVvKqTArYkRtMmJkLzVwXlXQ2VGQmVTUTLVVV3QdZSUTvVkl6ajBFQXdNdZjd3hMREFxQmdvSmtPvUprL0LzkwFFQkRCeDFJbTQ2Y1hKdU9tMmpjRHBqVrWdFkyTTZ1V053TF
</certificate>
</ns2:certificates>
<ns2:dataServerIdentifier>https://s124.ccg-gcc.gc.ca</ns2:dataServerIdentifier>
<ns2:datasetDiscoveryMetadata>
<ns2:S100_DatasetDiscoveryMetadata>
<ns2:fileName>file:/124CA01C_2446_25.GML</ns2:fileName>
<ns2:description>
<ns3:CharacterString>Montréal to Trois-Rivières - Montreal to Sorel</ns3:CharacterString>
</ns2:description>
<ns2:datasetID>urn:mcp:ccg:s124:CA01:C.2446.25</ns2:datasetID>
<ns2:compressionFlag>false</ns2:compressionFlag>
<ns2:dataProtection>false</ns2:dataProtection>
<ns2:protectionScheme>S100p15</ns2:protectionScheme>
<ns2:digitalSignatureReference>ECDSA-384-SHA2</ns2:digitalSignatureReference>
<ns2:digitalSignatureValue>
<S100_SE_DigitalSignature id="file:/124CA01C_2446_25.GML"
certificateRef="censcgc124">MGQCMG680Tpm23CHA0opQTZCqL4zRq0jIf758C60ra3DB8aokftDGL7Q5MxchCSQdIwJkP/jxfs2tq4EYXk65J0e/VJBr041fmIF7w8kH/ZsPy9cMmbT91b8XyWx2wHuMg<
</ns2:digitalSignatureValue>
<ns2:copyright>true</ns2:copyright>
<ns2:classification>
<ns14:MD_ClassificationCode codeList="https://standards.iso.org/iso/19115/resources/CodeLists/cat/codeLists.xml"
codeListValue="1">unclassified</ns14:MD_ClassificationCode>
</ns2:classification>

```

Figure 3.26

2) (Optional) Decode the certificate value because the certificate value in the catalog.xml is encoded in Base64. The 2 following options can be used:

- a) with the following online tool <https://www.base64decode.org/> :

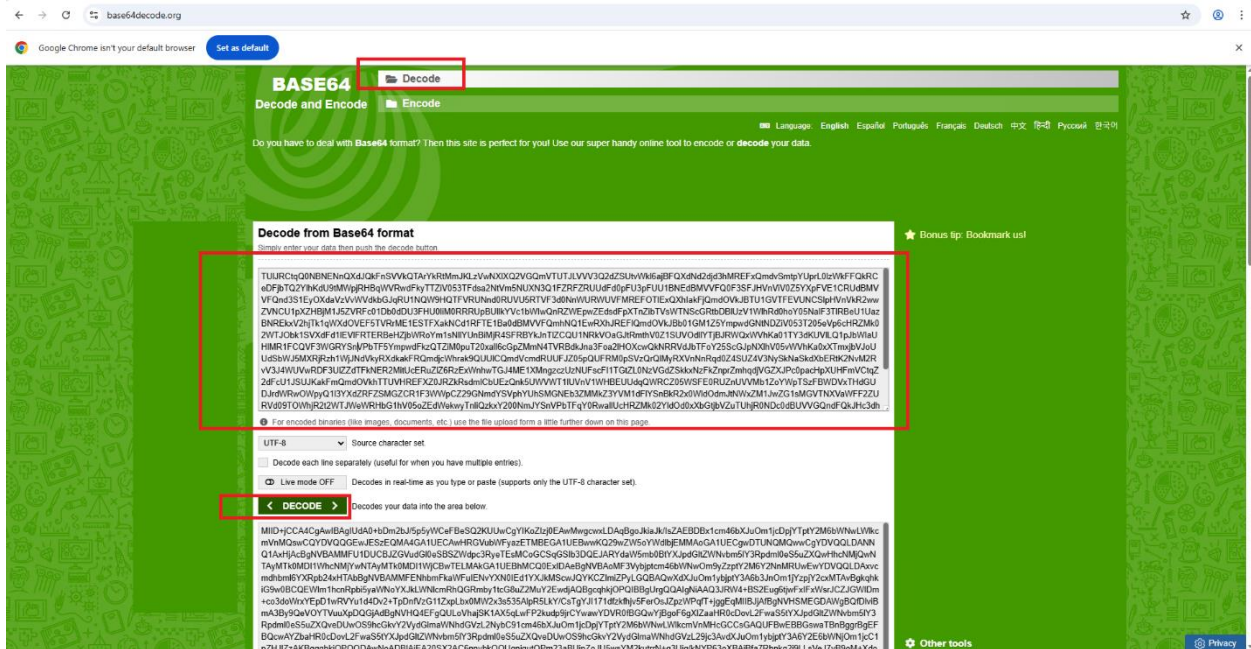


Figure 3.27

- b) Use a base 64 decoding tool from a programming language (Base64.getDecoder() in Java for example):

```
String decodedCertificateValue = new String(Base64.getDecoder().decode(certificateValue));
```

Figure 3.28

- 3) (Optional) With the decoded certificate value, extract the public key from the certificate. The two following options can be used:

- a) Put the decoded certificate value from step 2 in a file named `cgc_certificate.pem` and surround it by `-----BEGIN CERTIFICATE-----` and `-----END CERTIFICATE-----` (see figure 3.29);

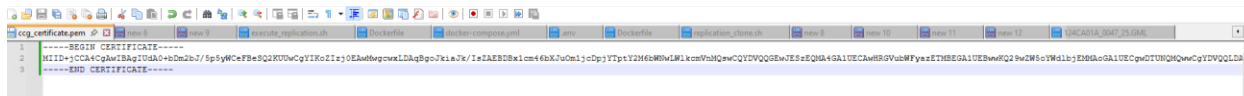


Figure 3.29

Execute the following openssl command in WSL (Linux environment in Windows) or in a Linux terminal:

```
openssl x509 -pubkey -noout -in cgc_certificate.pem > ec-secp384-r1-pub-key-extracted.pem
```

(see figure 3.30)

That command will create the file `ec-secp384-r1-pub-key-extracted.pem` containing the public key (see figure 3.31).

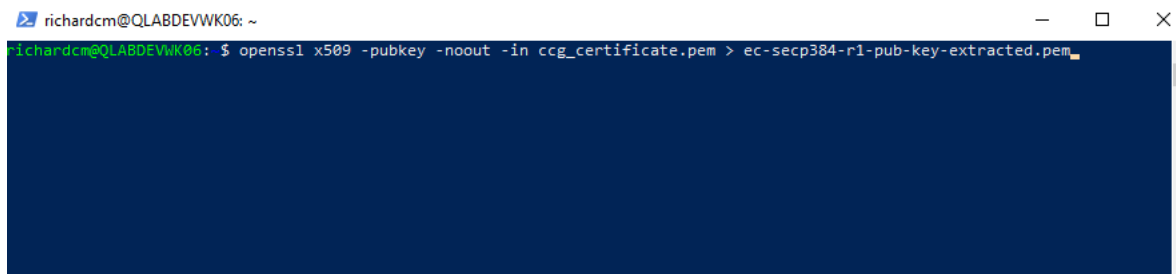


Figure 3.30

```

1 -----BEGIN PUBLIC KEY-----
2 MHYwEAYHKoZIzjOCAQYFK4EEACIDYgAENyUVuPgUthLoOrY8BcSBcVrKyQmSRlpQ
3 5vnKN3aFq8WBKQ9cEVWltXeA79vk6Q531cxtDWcaS28dDFtsd7Od+QJaUeS5GPwr
4 E4GCSNe9XX85H4Y7+RXqzrCWac1j6n0/
5 -----END PUBLIC KEY-----
6

```

Figure 3.31

- b) Use a programming language (like Java with the java.security library in the following example) to load the decoded certificate and extract the public key.

```

X509EncodedKeySpec publicKeySpec = new X509EncodedKeySpec(decodedCertificateValue.getBytes());
PublicKey publicKey = keyFactory.generatePublic(publicKeySpec);

```

Figure 3.32

- 4) Validate the signature with the public key and the signature value. To validate the signature, you can use an online tool (option a) or a programming language like Java (option b).

- a) If you have skipped steps 1 and 2 and 3, copy the following public key extracted from the CCG public certificate:

-----BEGIN PUBLIC KEY-----

MHYwEAYHKoZIzjOCAQYFK4EEACIDYgAENyUVuPgUthLoOrY8BcSBcVrKyQmSRlpQ
5vnKN3aFq8WBKQ9cEVWltXeA79vk6Q531cxtDWcaS28dDFtsd7Od+QJaUeS5GPwr
E4GCSNe9XX85H4Y7+RXqzrCWac1j6n0/

-----END PUBLIC KEY-----

Go to the following web site <https://emn178.github.io/online-tools/ecdsa/verify/> and select ECDSA->Verify Signature.

- i) Select Base64 for the Input Encoding (see figure 3.33);
- ii) Copy the content of the catalog.xml file (figure 3.34) associated with the signature to validate and add it to the Input text area (see figure 3.33);
- iii) Select SECG secp384r1 / X9.63 ansip384r1 / NIST P-384 for the Curve option (see figure 3.33);
- iv) Select SHA384 for the Signature Algorithm (see figure 3.33);
- v) Select Pem Text for the Public Key Type (see figure 3.33);
- vi) Add the Public Key content (surrounded by -----BEGIN PUBLIC KEY----- and -----END PUBLIC KEY-----) in the Public Key content text area (see figure 3.33);
- vii) Select Base64 for the Signature Type (see figure 3.33).
- viii) Copy the content of the catalog.sign file (In this example, see figure 3.35) and add it to the Signature Data text area (see figure 3.33);
- ix) Click on verify button.

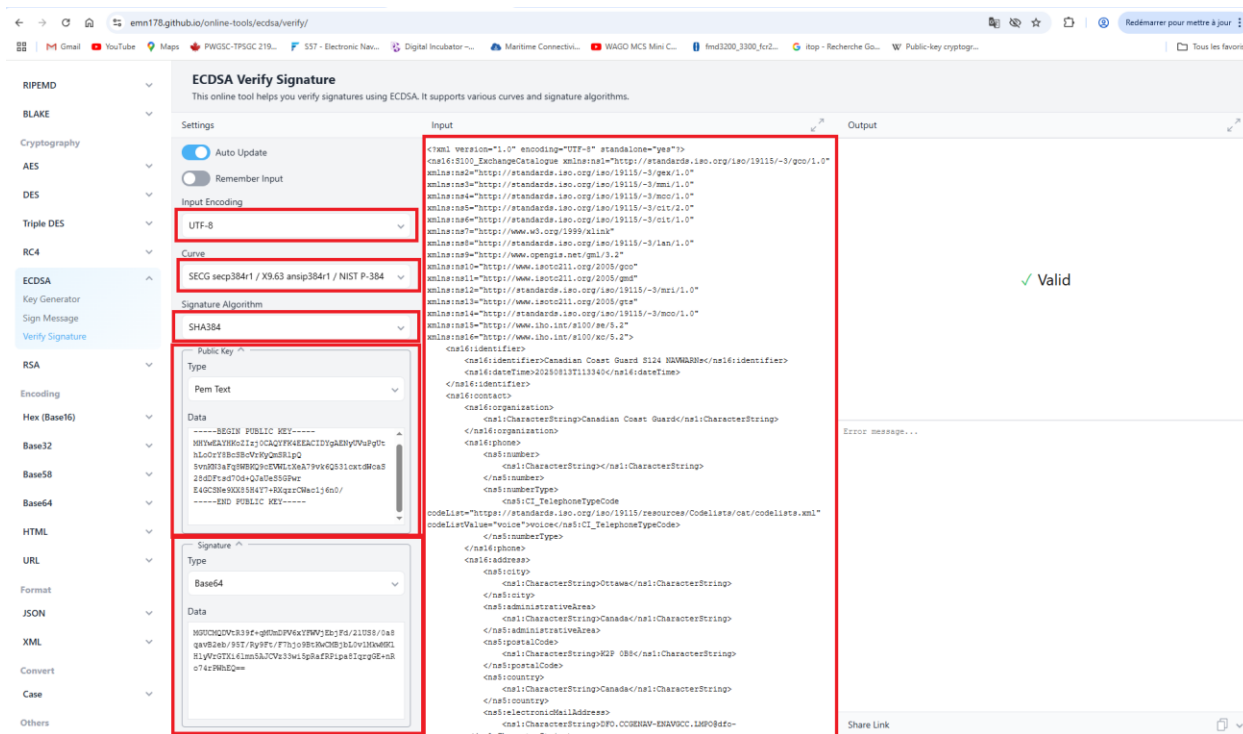


Figure 3.33

```

1 <xml version="1.0" encoding="UTF-8" standalone="yes">
2 <ns1:ExchangeCatalogue xmlns:ns1="http://standards.iso.org/iso/19115/-3/gco/1.0" xmlns:ns2="http://standards.iso.org/iso/19115/-3/gmi/1.0" xmlns:ns3="http://standar
3 <ns1:identifier>
4   <ns1:identifier>Canadian Coast Guard S124 NAVWARB</ns1:identifier>
5   <ns1:dateTime>20250813T13340</ns1:dateTime>
6 </ns1:identifier>
7 <ns1:contact>
8   <ns1:organization>
9     <ns1:characterString>Canadian Coast Guard</ns1:characterString>
10  </ns1:organization>
11  <ns1:phone>
12    <ns1:number>
13      <ns1:characterString></ns1:characterString>
14    </ns1:number>
15    <ns1:numberType>
16      <ns1:CI_TelephoneNumber codeList="http://standards.iso.org/iso/19115/resources/CodeLists/cat/codeLists.xml" codeListValue="voice">voice</ns1:CI_TelephoneNumber>
17    </ns1:numberType>
18  </ns1:phone>
19  <ns1:address>
20    <ns1:city>
21      <ns1:characterString>Ottawa</ns1:characterString>
22    </ns1:city>
23    <ns1:administrativeArea>
24      <ns1:characterString>Canada</ns1:characterString>
25    </ns1:administrativeArea>
26    <ns1:postalCode>
27      <ns1:characterString>R2P 0B8</ns1:characterString>
28    </ns1:postalCode>
29    <ns1:country>
30      <ns1:characterString>Canada</ns1:characterString>
31    </ns1:country>
32    <ns1:electronicMailAddress>
33      <ns1:characterString>DFO_COEEMAV-ENAVOCC.LMP0@dfo-mpo.gc.ca</ns1:characterString>
34    </ns1:electronicMailAddress>
35  </ns1:address>
36 </ns1:contact>
37 <ns1:productSpecification>
38   <ns1:name>Navigational Warnings</ns1:name>
39   <ns1:date>20250328</ns1:date>
40   <ns1:productIdentifier>S-124</ns1:productIdentifier>
41   <ns1:number>218</ns1:number>
42   <ns1:complianceCategory4</ns1:complianceCategory4>
43 </ns1:productSpecification>
44 <ns1:defaultLocale>
45   <ns1:PT_Locale>
46     <ns1:language>
47       <ns1:languageCode codeListValue="enq" codeSpace="ISO 639-2/T">English</ns1:languageCode>
48     </ns1:language>
49     <ns1:country>
50       <ns1:countryCode codeListValue="CAN" codeSpace="ISO 3166-2">Canada</ns1:countryCode>
51     </ns1:country>
52     <ns1:characterEncoding>
53       <ns1:HD_CharacterSetCode codeList="http://www.iana.org/assignments/character-sets" codeListValue="UTF-8">UTF-8</ns1:HD_CharacterSetCode>
54     </ns1:characterEncoding>
55   </ns1:PT_Locale>
56 </ns1:defaultLocale>
57 <ns1:exchangeCatalogueDescription>
58   <ns1:characterString>S124 Exchange Set of active canadian NAVWARB</ns1:characterString>
59 </ns1:exchangeCatalogueDescription>
60 <ns1:exchangeCatalogueComment>

```

Figure 3.34

```

1 MGUCMFzFXuzUOaBgcuaAjDXULqnYU7Fu4UUN7H4/AjUqfE1RO6zXof3/jFQQTcVhggEJPAixAI7n9N9BjgX5hbH58yOcoHkSJYpWvpG8uxiW1xvUkqn1RWxq2Mb8aU/SS5Q5UjhBA==

```

Figure 3.35

- b) In Java,
 - i) Assign the signature value (figure 3.35) to the (String) signature variable;

ii) Assign the encoded public certificate from the catalog.xml file (figure 3.26) to the (String) encodedPublicCertificate variable;

iii) Assign the content of the catalog.xml file (see figure 3.34) to the (String) content variable;

iv) execute the following commands (use algorithm 'SHA384WITHECDSA'):

```
byte[] decodedPublicCertificate = Base64.getDecoder().decode(encodedPublicCertificate);
byte[] publicCertificateValue = Base64.getDecoder().decode(decodedPublicCertificate);
X509Certificate certificate = (X509Certificate) CertificateFactory.getInstance("X.509").generateCertificate(new
ByteArrayInputStream(publicCertificateValue));
PublicKey publicKey = certificate.getPublicKey();

Signature ecdsaSignature = Signature.getInstance("SHA384withECDSA");
ecdsaSignature.initVerify(publicKey);
ecdsaSignature.update(content.getBytes());
boolean isValid = ecdsaSignature.verify(Base64.getDecoder().decode(signature.getBytes()));
```