How to decrypt a KEM2 file

How to decrypt a KEM2 file

Table of Contents

The sample code in this document is based on C# and .NetCore.

1 Download the file

Download the file from the Encryption Key Service portal. Selecting the KEM2 option and entering a password to secure the file.
For more info about how to download KEM file see the superuser guide.

2 The KEM2 file

The KEM2 file is an evolution of the KEM file with the feature of being signed by Kamstrup with its certificate's private key. By verifying this digital signature, the authenticity of the file as originating from Kamstrup and the integrity of the file itself is guaranteed.

To know more about the validating procedure, go to the 5 Validating the signature and for a deeper understanding of the KEM2 file format APPENDIX A: The KEM2 format file.

3 Extract Data

The downloaded file should look like the example below. Ending on 'zip.kem2'

```
20210812_1131_selected_DownloadDevices.zip.kem2
```

Remove extension

Rename the file and delete the '.kem2' from it make it a zip file.

Unzip

The output folder will contain, at least, the files below:

- 363A6D7848DF4882B0991674191A9B84.kem
- meter_information_file.xsd

Schema file

The file ending with '.xsd' contains the schema of the data when it has been decrypted. It can be used to validate that the output has the expected format.

Be aware that for some devices the schema could be different (device_information_file.xsd).

Data file

The file ending with '.kem' is the one containing the data. That is an XML file that should look like the following one.

```
<?xml version="1.0" encoding="utf-8"?><EncryptedData
Type="http://www.w3.org/2001/04/xmlenc#aes128-cbc"
xmlns="http://www.w3.org/2001/04/xmlenc#">
<EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#aes128-cbc" />
       <CipherData>
              <CipherValue>
                     Nd7AyPuTPjsRgvS0l3ExFLqmsAgKDzOQYBQToY23/4g7ArcUF3RXEuYhwY2Uskdlq
                     knRyvmZSuTlrNqctiafLB5c9pgpEZ0KydL1Sm26OmhsZjMzg2qhcLPoA3DV/aA97s
                     aotIlxYk3Pq2DmpkJtA8Mw4Kh57TirNOcFB3mhTN4V3GjwevmNib2nZ22EvU0V17i
                     GHCQPNMl3VN5lK5Ev8QnNT4Mjtlr3d0P7iWkO3w7H35lYPdoDZzaZiCluCTxKUZMW
                     TGTRRcUHgvDifSmCXhmpZS7f4AA+3S9x2LUB6vdHlQYh5FxQT/tsy/mLVphzDYXEE
                     Ej+TLG7Y3g59T0WVf6h+x8J0Rh5SGWfsV58F6+276cDkfMX7A8KanU6owlW6MFg/s
                     5s1RyK397mjDw9UJh9bIJuTc5Yeq6D8I3K5c5EyF+MYDGUo+SxwXbrX/j0UJkmua/
                     w4yhzBSvccF0yGQXS4QpVEK8yq+dRHvtpac8ZCXRS2OLP+eZt8qCQJDmpJ/M02kgx
                     UAo9Ocd30EBokXLxuwjOqxeZ/ZNTG3vFtn89dMhW+QtL8k8UL1hBkvaCClApLoi38
                     RVCFd/D9LIGJBDtsa20eu9miHX5CraLaLzsoXYXE8ijSU5u2we4N3mSGBzJWam/2i
                     JkHIxrGpRYqovNDHT7BHd/EVwGlzbekbVueWm8fjcnkXGY5GtxnciuIUJ4rvfvdmq
                     tcFXOR5iAx0RK25Y0/I6mmj3+S0f24hGatmzNfVWmNqFXmybO717i+sQMfyflTli/
                     RY6Eeb/0G3qABv+NIWJP+8p2CSDCC59LmrsDWFtDxsEBn+RQnTMT+vh4VDPOFEuM1
                     SNlvvHozlwsIV0jdQxw1muc7yVzeaHZoGIJe+fculeB4VLsaGxZVOHnd3R6iwYc1v
                     z7r1KiDXlh/F0zORtaAowoqglPNfQ=
              </CipherValue>
       </CipherData>
</EncryptedData>
```

Convert it to data

The encrypted data is the value of the content of the tag 'CipherValue ' in the data file. Load the file as xml, search by tag and convert it to byte array.

```
var encryptedXmlDocuent = new XmlDocument();
encryptedXmlDocuent.Load(kemFilePath);
var encryptedElement = encryptedXmlDocuent.GetElementsByTagName("CipherValue")[0] as
XmlElement;
var encryptedData = Convert.FromBase64String(encryptedElement.InnerText);
```

4 Decrypt Data

Secure Password

Need to convert the password that was entered when downloaded the file into a SecureString.

```
SecureString pwd = plainPwd.ConvertToSecureString();

public static SecureString ConvertToSecureString(this string unsecurePassword)
{
    if (unsecurePassword == null)
    {
        throw new ArgumentNullException("unsecurePassword");
    }

    var securePassword = new SecureString();
    foreach (var c in unsecurePassword)
        securePassword.AppendChar(c);
    securePassword.MakeReadOnly();
    return securePassword;
}
```

Decrypt

Given the key and the encrypted data it needs to set the decryptor and transform the data.

That is marked as unsafe. So, the project itself needs to mark as allowed to use unsafe code.

```
public static unsafe byte[] GetDecryptedData(SecureString key, byte[] textToDecrypt)
{
    const int keyLength = 0x10;
    byte[] decryptedText;
    var keyAs16Bytes = new byte[keyLength];

    //validations

    var unmanagedBytes = Marshal.SecureStringToGlobalAllocAnsi(key);
    try
    {
        byte* byteArray = (byte*)unmanagedBytes;
        var pEnd = byteArray;
        while (*pEnd++ != 0){}
        var length = (int)((pEnd - byteArray) - 1);
        for (var i = 0; i < length; ++i)
        {
            keyAs16Bytes[i] = *(byteArray + i);
        }

        using (var alg = new RijndaelManaged())
        {
            //User cipher block chaning
            alg.Mode = CipherMode.CBC;
            alg.Padding = PaddingMode.PKCS7;

            // 128 bit key
            alg.KeySize = 0x80;
            alg.BlockSize = 0x80;

            // Secret key
            alg.Key = keyAs16Bytes;
            // Initialation Vector
            alg.IV = keyAs16Bytes;

            decryptedText = alg.CreateDecryptor().TransformFinalBlock(textToDecrypt,
0, textToDecrypt.Length);
        }
    }
    catch (CryptographicException e)
    {
        throw new Exception("CryptographicException while decrypting document.",
          e);
    }
    finally
    {
        Marshal.ZeroFreeGlobalAllocAnsi(unmanagedBytes);
        Array.Clear(keyAs16Bytes, 0, keyLength);
    }

    return decryptedText;
}
```

Validations

Those are some of the recommended checks that can be done before the decryption and avoid further exception.

```
    if (textToDecrypt == null || textToDecrypt.Length == 0)
    {
        throw new Exception("No encrypted data");
    }

    if (key == null || key.Length == 0)
    {
        throw new Exception("Key cannot be null or empty");
    }

    if (key.Length > keyLength)
    {
        throw new Exception("Key size is wrong. The key can not be larger than " +
keyLength);
    }
```

Output

To make it easier to read and/or as source for xml document, it needs to be converted to a string.

```
decrtyptedText = Encoding.UTF8.GetString(decryptedData);
```

The output result of decrypting a KEM2 file will be something like this:

```
<Devices>
	<Device>
		<DeviceId>
			<SerialNumber>23310208</SerialNumber>
			<ManufacturerId>KAW</ManufacturerId>
		</DeviceId>
		<ConsumptionTypeName>WaterReclaimed</ConsumptionTypeName>
		<DeviceTypeName>Meter</DeviceTypeName>
		<CustomerDeviceNumber>KWM2231-CDMN</CustomerDeviceNumber>
		<OrderNumber>KWM2231-23310208</OrderNumber>
		<ProductPrefix>KWM2231</ProductPrefix>
		<DeviceCommercialName>flowIQ® 2200</DeviceCommercialName>
		<LongName>flowIQ® 2200</LongName>
		<ShortName>KWM2231</ShortName>
		<ConfigNumber>1</ConfigNumber>
		<ProgramNumber>1111</ProgramNumber>
		<TypeNumber>02K810070N212</TypeNumber>
		<Keys>
			<Key>
				<Name>DEK</Name>
				<Value>0100000000A1D2100000000023310208</Value>
			</Key>
			<Key>
				<Name>LorawanAppKey</Name>
				<Value>0900000000A1D2100000000023310208</Value>
			</Key>
		</Keys>
```

```
            <ExtendedDeviceData>
                    <DeviceProperty>
                            <Name>LorawanDevEUI</Name>
                            <Value>001LorawanDevEUI23310208</Value>
                    </DeviceProperty>
            </ExtendedDeviceData>
        </Device>
        <Signature> … </Signature>
</Devices>
```

5 Validating the signature

Installing the certificates

To verify the signature, you need to install the Public key certificate on the Personal folder of the Local Machine Certificate Manager downloading it from:

https://eks.kamstrup.com/certificates/Eks.Signing.PublicKey.cer

To have a valid certification path you will need also to install on Trusted Root Certification Authorities:

https://eks.kamstrup.com/certificates/Eks.Signing.HCWSSRootCA.cer

https://eks.kamstrup.com/certificates/Eks.Signing.CA.crt

And this one on the Intermediate Certification Authorities:

https://eks.kamstrup.com/certificates/Eks.Signing.C1.crt

To see how to use the Public Key certificate for validating the signature, please look at the section “Validate the signature” of this document.

Taking the certificate

First of all, you will need to go through the machine certificates store to look for the Public Key certificate. To do that, just need to know the thumbprint of the certificate to recognize it ("14d7305cadf3c982a390c94cc6e0054091b63531" in this case) and then loop the X509Store to get it and save into a X509Certificate class. Both classes are from the namespace System.Security.Cryptography.X509Certificates.

```
var certThumbprint = "14d7305cadf3c982a390c94cc6e0054091b63531";

X509Certificate2 cert = null;
using (var store = new X509Store(StoreLocation.LocalMachine))
        {
            store.Open(OpenFlags.ReadOnly);

            foreach (var certificate in store.Certificates)
            {
                if (certificate.Thumbprint == certThumbprint.ToUpper())
                {
                    cert = certificate;
                    break;
                }
            }
            store.Close();
        }
```

Using the certificate for validating the signature

We will need to take the class SignedXml from the namespace SignatureSystem.Security.Cryptography.Xml, load the "Signature" tag on that class and call the CheckSignature method with the PublicKey from the X509Certificate class.

```
var decryptedXmlDocument = new XmlDocument();
decryptedXmlDocument.LoadXml(decryptedText);
var signedXml = new SignedXml(decryptedXmlDocument);
signedXml.LoadXml((XmlElement)decryptedXmlDocument.GetElementsByTagName("Sign
ature")[0]);
var verifySignature = signedXml.CheckSignature((RSA)cert.PublicKey.Key);
```

APPENDIX A: The KEM2 format file

Version 2.0 Schema

```
<?xml version="1.0" encoding="utf-8"?>

<xs:schema targetNamespace="https://eks.kamstrup.com/KEM2" id="KEM2"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:ds=http://www.w3.org/2000/09/xmldsig# attributeFormDefault="unqualified"
elementFormDefault="qualified" xmlns:kam="https://eks.kamstrup.com/KEM2"
version="1.0">

  <xs:import namespace="http://www.w3.org/2000/09/xmldsig#"
schemaLocation="xmldsig-core-schema.xsd"/>

  <xs:annotation>

    <xs:documentation xml:lang="en">

      Kamstrup Key Exchange Format (KEM2)

      Version 2.0 :

      DeviceTypeName valid values : Meter, Repeater, Concentrator, Gateway,
       Valve, FlowSensor

      ConsumptionTypeName valid values : "" Empty string, Electricity,
       VolumeCold, VolumeHeat, Cooling, HeatCooling, Heat, Pressure, Energy,
       TemperatureHumidity, Unknown

    </xs:documentation>

  </xs:annotation>

  <xs:element name="Devices" type="kam:DevicesType">

    <xs:unique name="DeviceUniqueness">

      <xs:selector xpath="kam:Device/kam:DeviceId"/>

      <xs:field xpath="kam:SerialNumber"/>

      <xs:field xpath="kam:ManufactureId"/>

    </xs:unique>

  </xs:element>
```

```
  <xs:complexType name="DevicesType" >

    <xs:sequence>

      <xs:element type="kam:DeviceType" name="Device" maxOccurs="unbounded"
minOccurs="1">

      </xs:element>

      <xs:element ref="ds:Signature"/>

    </xs:sequence>

    <xs:attribute name="SchemaVersion" type="xs:NMTOKEN" use="required"
fixed="2.0"/>

  </xs:complexType>

  <xs:complexType name="DeviceType">

    <xs:sequence>

      <xs:element type="kam:DeviceIdType" name="DeviceId" maxOccurs="1"
minOccurs="1">

      </xs:element>

      <xs:element name="ConsumptionTypeName" type="xs:normalizedString"
maxOccurs="1" minOccurs="0"/>

      <xs:element name="DeviceTypeName" type="xs:normalizedString"
maxOccurs="1" minOccurs="1"/>

      <xs:element name="CustomerDeviceNumber" type="xs:normalizedString"
maxOccurs="1" minOccurs="0"/>

       <xs:element name="OrderNumber" type="xs:normalizedString"
maxOccurs="1" minOccurs="0"/>

<xs:element name="ProductPrefix"  maxOccurs="1" minOccurs="0">

        <xs:simpleType>

          <xs:restriction base="xs:normalizedString">

            <xs:maxLength  value="25"/>

          </xs:restriction>

        </xs:simpleType>

      </xs:element>
```

```
            <xs:element name="DeviceCommercialName"  maxOccurs="1"
minOccurs="0">
        <xs:simpleType>
          <xs:restriction base="xs:normalizedString">
            <xs:maxLength  value="300"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
      <xs:element name="LongName" maxOccurs="1" minOccurs="1">
        <xs:simpleType>
          <xs:restriction base="xs:normalizedString">
            <xs:maxLength  value="300"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
      <xs:element name="ShortName" maxOccurs="1" minOccurs="1">
        <xs:simpleType>
          <xs:restriction base="xs:normalizedString">
            <xs:maxLength  value="100"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
      <xs:element name="ConfigNumber" type="xs:NMTOKEN"  maxOccurs="1"
minOccurs="0"/>
      <xs:element name="ProgramNumber" type="xs:NMTOKEN" maxOccurs="1"
minOccurs="0"/>
```

```
                  <xs:element name="TypeNumber" type="xs:NMTOKEN"
maxOccurs="1" minOccurs="0"/>

      <xs:element type="kam:KeysType" name="Keys" maxOccurs="1"
minOccurs="1">

        <xs:unique name="uniqueKeyName">

          <xs:selector xpath="kam:Key"/>

          <xs:field xpath="kam:Name"/>

          <xs:field xpath="@Version"/>

        </xs:unique>

      </xs:element>

      <xs:element type="kam:ExtendedDeviceDataType" name="ExtendedDeviceData"
maxOccurs="1" minOccurs="0">

        <xs:unique name="uniqueExtendedDeviceData">

          <xs:selector xpath="kam:DeviceProperty"/>

          <xs:field xpath="kam:Name"/>

        </xs:unique>

      </xs:element>

    </xs:sequence>

  </xs:complexType>

  <xs:complexType name="DeviceIdType" >

    <xs:sequence>

      <xs:element name="SerialNumber" maxOccurs="1" minOccurs="1">

        <xs:simpleType>

          <xs:restriction base="xs:NMTOKEN">

            <xs:pattern value="[0-9]{6,8}"/>

          </xs:restriction>

        </xs:simpleType>

      </xs:element>
```

```
                    <xs:element name="ManufacturerId" maxOccurs="1"
minOccurs="1">
        <xs:simpleType>
          <xs:restriction base="xs:NMTOKEN">
            <xs:pattern value="[A-Z]{3}"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="KeysType">
    <xs:sequence>
      <xs:element type="kam:KeyType" name="Key" maxOccurs="unbounded"
minOccurs="0">
      </xs:element>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="KeyType">
    <xs:sequence>
      <xs:element name="Name">
        <xs:simpleType>
          <xs:restriction base="xs:normalizedString">
            <xs:minLength  value="3"/>
            <xs:maxLength  value="50"/>
          </xs:restriction>
        </xs:simpleType>
```

```
                    </xs:element>
      <xs:element name="Value">
        <xs:simpleType>
          <xs:restriction base="xs:hexBinary">
            <xs:minLength  value="8"/>
            <xs:maxLength value="512"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
    </xs:sequence>
    <xs:attribute name="Version" type="xs:unsignedShort">
    </xs:attribute>
  </xs:complexType>
  <xs:complexType name="ExtendedDeviceDataType">
    <xs:sequence>
      <xs:element type="kam:DevicePropertyType" name="DeviceProperty"
maxOccurs="unbounded" minOccurs="0">
      </xs:element>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="DevicePropertyType">
    <xs:sequence>
      <xs:element name="Name">
        <xs:simpleType>
          <xs:restriction base="xs:string">
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
      <xs:element name="Value">
```

```
            <xs:simpleType>
          <xs:restriction base="xs:string">
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```

Description elements

Devices - Root element where the children are a list of Device elements.

Device - Element with all the information for a single device. At least one device must be within each XML document to be valid.

DeviceId - Element containing the SerialNumber and the ManufactureId. The combination of SerialNumber and ManufactureId should be unique within each XML document to be valid.

> **SerialNumber** - The serial number of a device. Will be a number between 6-8 digits.
>
> **ManufactureId** - The ManufactureId will be 3 uppercase digits.

ConsumptionTypeName - Containing a known Consumption Type or alternatively “Unknown” or ””. The known Consumption Types can be found in the comments at the top of the schema. This element is optional.

DeviceTypeName - The name of the DeviceType indicates the device type. The Kamstrup Device Types are listed in the comments at the top of the schema.

CustomerDeviceNumber - Contains a customer defined serial number. This element is optional.

OrderNumber - The order number where the device was purchased. This element is optional.

ProductPrefix - Product identifier prefix. This element is optional.

DeviceCommercialName - Commercial name of the device. This element is optional.

LongName - The long descriptive name of the device.

ShortName - The short name of the device.

ConfigNumber - The config number of the device. This element is optional.

ProgramNumber - The program number of the device. This element is optional.

TypeNumber - The type number of the device. This element is optional.

Keys - A list containing the device keys.

Every **key** will have the following fields:

> **Version** – The key version of the keys that support versioning. This element is optional.
>
> **Name** - The name of the key.
>
> **Value** - The encryption key value represented as a HEX Binary type. Must be 8-512 bytes.

ExtendedDeviceData - Contains a list of additional device properties. Additional device identifiers may be provided here. This element is optional.

Every **DeviceProperty** will have the following fields:

> **Name** - Name of the property.
>
> **Value** - value of that property (string).

Signature - Digital signature of XML.

APPENDIX B: The deprecated versions

This section lists deprecated KEM2 schemas.

Version 1.0

The version 1.0 schema is identical to the 2.0 schema except that the key version element was a mandatory numerical-only element.

The key version of keys exported through this schema was set to 1.

The key version handling in version 1.0 was incorrect. Version 1.0 is deprecated and is no longer supported in system.