How to decrypt a KEM file

How to decrypt a KEM 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 KEM option and entering a password to secure the file.
For more info about how to download KEM file see the superuser guide.

2 Extract Data

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

```
20210407_1119DownloadMeters.zip.kem
```

Remove extension

Rename the file and delete the '.kem' 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>
```


Kamstrup A/S • Industrivej 28, Stilling · 8660 Skanderborg · Denmark • www.kamstrup.com
FILE100003043_B_EN_07.2025

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 KEM file will be something like this:

```
<MetersInOrder orderid="" schemaVersion="2.0">
    <Meter>
        <MeterNo>78127978</MeterNo>
        <SerialNo>78127978</SerialNo>
        <EncKeys>
            <DEK>00000000000000000000000000000004</DEK>
        </EncKeys>
        <MeterName>MC602</MeterName>
        <ConsumptionType>Cooling</ConsumptionType>
        <ConfigNo>510002424003</ConfigNo>
        <ProgramNo>44458458</ProgramNo>
        <TypeNo>602A00000075DA</TypeNo>
        <VendorId>KAM</VendorId>
    </Meter>
    <Meter>
        <MeterNo>78177775</MeterNo>
        <SerialNo>78177775</SerialNo>
        <EncKeys>
            <DEK>00000000000000000000000000000004</DEK>
        </EncKeys>
        <MeterName>MC602</MeterName>
        <ConsumptionType>Heat</ConsumptionType>
        <ConfigNo>217002424003</ConfigNo>
        <ProgramNo>34451451</ProgramNo>
        <TypeNo>602C03870A1212</TypeNo>
        <VendorId>KAM</VendorId>
    </Meter>
</MetersInOrder>
```

Appendix: the KEM format file

Version 1.0 Schema

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

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">


<xs:annotation>

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

            Distribution format for Kamstrup Encryption and Meter
information.

            Copyright 2010 Kamstrup A/S. All rights reserved.

            Version 3.0 changes:

            - ConsumptionType changed from enum to string

            - DeviceType changed from enum to string

            Version 2.0 changes:

            - added: new element types (Device) and new DevicesInOrder

            - added: VendorId

            - added: Consumption type Electricity

            - removed: Key length constaint on encryption keys

      </xs:documentation>

</xs:annotation>




<!-- Definition of generic elements -->

<xs:element name="SerialNo" type="xs:string"/>

<xs:element name="VendorId" type="xs:string"/>


<!-- Definition of simple elements, meter related-->
```

```
<xs:element name="MeterNo" type="xs:string"/>

<xs:element name="MeterName" type="xs:string"/>  <!-- MeterName: MC21, MC601
etc. -->

<xs:element name="ConsumptionType" type="xs:string"/>

<xs:element name="ConfigNo" type="xs:string"/>

<xs:element name="ProgramNo" type="xs:string"/>

<xs:element name="TypeNo" type="xs:string"/>


<!-- Definition of simple elements, other devices -->

<xs:element name="DeviceType" type="xs:string"/>

<xs:element name="DeviceName" type="xs:string"/>


<!-- Definition of simple elements, flowsensors -->

<xs:element name="FlowSensorType" type="xs:string"/>

<xs:element name="FlowSensorName" type="xs:string"/>


<!-- Encryption key elements -->

<xs:element name="PK1"/>

<xs:element name="PK2"/>

<xs:element name="PK3"/>

<xs:element name="DEK"/>

<xs:element name="GPK1"/>

<xs:element name="GPK2"/>

<xs:element name="GPK3"/>

<xs:element name="GPK4"/>

<xs:element name="UFPW"/>


<!-- Definition of attributes -->
```

```
<xs:attribute name="orderid" type="xs:string"/>
<!-- Definition of schemaVersion, must be stepped when xsd i changed -->
<xs:attribute name="schemaVersion" type="xs:string"/>

<!-- Defintion of complex elements -->
<xs:element name="EncKeys">
      <xs:complexType>
            <xs:sequence>
                  <xs:element ref="PK1" minOccurs="0" maxOccurs="1"/>
                  <xs:element ref="PK2" minOccurs="0" maxOccurs="1"/>
                  <xs:element ref="PK3" minOccurs="0" maxOccurs="1"/>
                  <xs:element ref="DEK" minOccurs="0" maxOccurs="1"/>
                  <xs:element ref="GPK1" minOccurs="0" maxOccurs="1"/>
                  <xs:element ref="GPK2" minOccurs="0" maxOccurs="1"/>
                  <xs:element ref="GPK3" minOccurs="0" maxOccurs="1"/>
                  <xs:element ref="GPK4" minOccurs="0" maxOccurs="1"/>
            </xs:sequence>
      </xs:complexType>
</xs:element>

  <xs:element name="EncPasswords">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="UFPW" minOccurs="0" maxOccurs="1"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
```

```
<!-- Definition of the basic meter information model -->
<xs:element name="Meter">
	<xs:complexType>
		<xs:sequence>
			<xs:element ref="MeterNo" minOccurs="0" maxOccurs="1"/>
			<xs:element ref="SerialNo"/>
			<xs:element ref="EncKeys" minOccurs="0" maxOccurs="1"/>
			<xs:element ref="MeterName"/>
			<xs:element ref="ConsumptionType" minOccurs="0"
maxOccurs="1"/>
			<xs:element ref="ConfigNo"/>
			<xs:element ref="ProgramNo"/>
			<xs:element ref="TypeNo"/>
			<xs:element ref="VendorId"/>
		</xs:sequence>
	</xs:complexType>
</xs:element>

<!--Definition of the basic device information model (non-meters) -->
<xs:element name="Device">
	<xs:complexType>
		<xs:sequence>
			<xs:element ref="DeviceType"/>
			<xs:element ref="TypeNo"/>
			<xs:element ref="DeviceName"/>
			<xs:element ref="SerialNo"/>
			<xs:element ref="VendorId"/>
```

```
                    <xs:element ref="EncKeys" minOccurs="0" maxOccurs="1"/>
              </xs:sequence>
       </xs:complexType>
</xs:element>

  <!--Definition of the basic device information model (non-meters) -->
  <xs:element name="FlowSensor">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="FlowSensorType"/>
        <xs:element ref="TypeNo"/>
        <xs:element ref="FlowSensorName"/>
        <xs:element ref="SerialNo"/>
        <xs:element ref="VendorId"/>
        <xs:element ref="EncPasswords" minOccurs="0" maxOccurs="1"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

<!-- Definition of Meter list Information model -->
<xs:element name="MetersInOrder">
       <xs:complexType>
              <xs:sequence>
                    <xs:element ref="Meter" maxOccurs="unbounded"/>
              </xs:sequence>
              <xs:attribute ref="orderid" use="required"/>
<!-- Version number, must always be step, if xsd is changed-->
```

```
            <xs:attribute ref="schemaVersion" use="required" fixed="2.0"/>
      </xs:complexType>
</xs:element>

<!-- Definition of Device list information model -->
<xs:element name="DevicesInOrder">
      <xs:complexType>
            <xs:sequence>
                  <xs:element ref="Device" maxOccurs="unbounded"/>
            </xs:sequence>
            <xs:attribute ref="orderid" use="required"/>
<!-- Version number, must always be step, if xsd is changed-->
            <xs:attribute ref="schemaVersion" use="required" fixed="2.0"/>
      </xs:complexType>
</xs:element>

  <!-- Definition of FlowSensor list information model -->
  <xs:element name="FlowSensorsInOrder">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="FlowSensor" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute ref="orderid" use="required"/>
      <!-- Version number, must always be step, if xsd is changed-->
      <xs:attribute ref="schemaVersion" use="required" fixed="2.0"/>
    </xs:complexType>
  </xs:element>
```

```
</xs:schema>
```

Description elements

MetersInOrder- Root element where the children are a list of **Meter** elements. This element is optional.

DevicesInOrder- Root element where the children are a list of **Device** elements. This element is optional.

FlowSensorsInOrder- Root element where the children are a list of **FlowSensor** elements. This element is optional.

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

Meter - Element with all the information for a single meter. Contains inside an **EncKeys** element for storing the keys.

Device - Element with all the information for a single device. Contains inside an **EncKeys** element for storing the keys.

FlowSensor - Element with all the information for a single flow sensor. Contains inside an **EncPasswords** element for storing the passwords.

SerialNo - The serial number of a device. Will be a number between 6-8 digits.

VendorId - The VendorId must be 3 uppercase digits.

MeterNo - Contains a customer defined serial number. This element is optional and only appears on **Meters**.

MeterName - The short name of the meter. This element only appears on **Meters**.

ConsumptionType - 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 and only appears on **Meters**.

ConfigNo - The config number of the meter. This element only appears on **Meters**.

ProgramNo - The program number of the meter. This element only appears on **Meters**.

TypeNor - The type number of the meter. This element only appears on **Meters**.

DeviceType - The name of the DeviceType indicates the device type. The Kamstrup Device Types are listed in the comments at the top of the schema. This element only appears on **Devices**.

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.

EncKeys - A list containing the device keys.

EncPasswords - A list containing the flow sensor passwords.