kamstrup

How to decrypt an M-Bus hosted response

kamstrup

1 Contents

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

1 Prerequisites

Before starting, the EKS-READy DeviceInfoServer-agreed certificate must be installed with its private key on the local machine certificate store.

Also, access to the following libraries must be granted:

- System.Security.Cryptography
- System.Security.Cryptography.X509Certificates

2 The M-Bus hosted response

When asked for an M-Bus Master on the DeviceInfoServer, the response can be something like this:


```
{
    “CommercialName”: “M-Bus Master Controller”,
    “ConfigNumber”: “1”,
    “ConsumptionType”: “Heat”,
    “DeviceType”: 2,
    “EncryptedKeyInfo”: {
        “EncryptedKeyList”: [{
                “EncryptionKey”: “mQiKf351EuXenEqNf5zPuj3NCvNUIEBoHx2jXeEZzDk=”,”KeyName”: “HAK”
            }, {
                “EncryptionKey”: “ymjB0j6xwXL8YDLKX1V2fB3m9FgPDCt5OGkRAbAwMYU=”,”KeyName”: “PAK”
            }, {
                “EncryptionKey”: “6YjsF5VpVR52Ad45qmJCUzFhkfautDa7+Fnu4EUBUMg=”,”KeyName”: “PPK”
            }
        ],
        “KeyEncryptionKey”: “KLhjd0Ear9QBjH/WlAyCUyicMPmQDUtIo+N8ShRo46j1sB5Ojx/eksNi8wdMx7SrFGkRS/Q/IpAwrR-
4RFUr6Is1M4th6VPNBwN1bWiH9jVq4HQgq2wsOhDeM0snPdH5FBHDABTdHBtCk+USssvT4rxOCoLgSoCPIoWz79hKex+tdAKkFmFsIgecdtY-
qQghrdfABbDu0v//HkJgWM2icOoEkZUPJZdFrgH2VCRyjR5usQ4EGRR25VdF0akCnVAXCw02IEnzzEOCHZbxyFxg/6JuyU00JFWtGnikb+Xdn-
y8UGSIzIEtCsf5Dm7Ly1iAwb9YzdQ8alqfhWR2qSaS8ooZQ==”
    },
    “LongName”: “M-Bus Master Controller”,
    “MeterNumber”: “For READy Tests”,
    “ProductPrefix”: “MBMC2022”,
    “ProgramNumber”: “1111”,
    “SerialNumber”: “66620054”,
    “ShortName”: “MBMC2022”,
    “TypeNumber”: “6841131AN110000801”,
    “UpdateSeqNo”: 192378157,
    “VendorId”: “KAM”,
    “KeyInfoList”: [{
            “EncryptionKey”: “0200000000A1D2100000000066620054”,”KeyName”: “DEK”
        }, {
            “EncryptionKey”: “0400000000A1D2100000000066620054”,”KeyName”: “MLK”
        }
    ]
}
```


On this response, the keys HAK, PAK and PPK will come encrypted inside the EncryptedKeyInfo structure:


```
{
     “EncryptedKeyInfo”: {
        “EncryptedKeyList”: [{
                “EncryptionKey”: “mQiKf351EuXenEqNf5zPuj3NCvNUIEBoHx2jXeEZzDk”,”KeyName”: “HAK”
            }, {
                “EncryptionKey”: “ymjB0j6xwXL8YDLKX1V2fB3m9FgPDCt5OGkRAbAwMYU”,”KeyName”: “PAK”
            }, {
                “EncryptionKey”: “6YjsF5VpVR52Ad45qmJCUzFhkfautDa7+Fnu4EUBUMg=”,”KeyName”: “PPK”
            }
        ],
…
```

These keys are encrypted using the Key Encryption Key (called KEK from now on) that can be found on the same structure:

```
…
        “KeyEncryptionKey”: “KLhjd0Ear9QBjH/WlAyCUyicMPmQDUtIo+N8ShRo46j1sB5Ojx/eksNi8wdMx7SrFGkRS/Q/IpAwrR-
4RFUr6Is1M4th6VPNBwN1bWiH9jVq4HQgq2wsOhDeM0snPdH5FBHDABTdHBtCk+USssvT4rxOCoLgSoCPIoWz79hKex+tdAKkFmFsIgecdtY-
qQghrdfABbDu0v//HkJgWM2icOoEkZUPJZdFrgH2VCRyjR5usQ4EGRR25VdF0akCnVAXCw02IEnzzEOCHZbxyFxg/6JuyU00JFWtGnikb+Xdn-
y8UGSIzIEtCsf5Dm7Ly1iAwb9YzdQ8alqfhWR2qSaS8ooZQ==”
    }
}
```

This KEK is already encrypted with the EKS-READy DeviceInfoServer-agreed certificate’s public part.

When decrypting keys, the fist step is thus to retrieve this certificate that will provide the private key to decrypt the KEK, and once we have the decrypted KEK, it can be used for decrypting the keys.

3 Getting the certificate

To get the certificate, given one thumbprint (at the certThumprint variable on the sample below), the storedCertificate must be searched until one matches it.

```
X509Certificate2 certificate = null;
using (var store = new X509Store(StoreName.My, StoreLocation.LocalMachine))
{
store.Open(OpenFlags.ReadOnly);
foreach (var storedCertificate in store.Certificates) {
        if (storedCertificate.Thumbprint == certThumbprint.ToUpper()) {
                        certificate = storedCertificate;
                        break;
                }
        }
store.Close();
}
```

4 Decrypting the KEK

Once the certificate is stored at the variable certificate, it can be used for decrypting the KEK using RSA:

```
var privateKey = (RSACryptoServiceProvider)certificate.PrivateKey;
var keyArray = privateKey.Decrypt(Convert.FromBase64String(KEKToDecrypt), RSAEncryptionPadding.OaepSHA1);
var decryptedKEK = Convert.ToBase64String(keyArray);
```

5 Using the decrypted KEK for decrypting keys

With the KEK decrypted, any key can be decrypted using the AES Key Wrap Algorithm RFC 3394:

```
string decryptedKey;
var ciphertext = Convert.FromBase64String(encryptedKey);
var kek = Convert.FromBase64String(decryptedKek);
var C = Block.BytesToBlocks(ciphertext);

// 1) Initialize variables
var A = C[0];
var R = new Block[C.Length - 1];
for (int i = 1; i < C.Length; i++)
        R[i - 1] = C[i];

var n = R.Length;

// 2) Calculate intermediate values
for (long j = 5; j >= 0; j--)
{
        for (long i = n - 1; i >= 0; i--)
        {
                byte[] byteBlock;
                var t = n * j + i + 1;  // add 1 because i is zero-based
                A ^= t;
                byte[] cipherBytes = A.Concat(R[i]);

                using (var alg = Aes.Create("AesManaged"))
                {
                        alg.Padding = PaddingMode.None;
                        alg.Mode = CipherMode.ECB;
                        alg.Key = kek;

                        using (MemoryStream msDecrypt = new MemoryStream())
                        {
                                using (var decryptor = alg.CreateDecryptor())
                           {
                                        using (CryptoStream csDecrypt = new CryptoStream(msDecrypt,
decryptor, CryptoStreamMode.Write))
                                    {
                                                csDecrypt.Write(cipherBytes, 0, alg.BlockSize / 8);
                                                byteBlock = msDecrypt.ToArray();
                                        }
                           }
                       }
                        }
                    var B = Block.BytesToBlocks(byteBlock);
                    A = B[0];
                    R[i] = B[1];
                }
            }

            // 3) Output the results
            decryptedKey =  Convert.ToBase64String(Block.BlocksToBytes(R));
```

6 Auxiliary class block

For help with decryption of the algorithm before writing:

```
public class Block
{
        private byte[] _b = new byte[8];
        public byte[] Bytes => _b;

        public Block(Block b) : this(b.Bytes) {}

        public Block(byte[] bytes) : this(bytes, 0) {}

        public Block(byte[] bytes, int index) {
                if (index + 8 > bytes.Length)
                        throw new ArgumentException("BufferLengthError", nameof(bytes));
                if (index < 0)
                        throw new ArgumentOutOfRangeException(nameof(index));

                Array.Copy(bytes, index, _b, 0, 8);
        }

        public byte[] Concat(Block right)
        {
                var output = new byte[16];
                _b.CopyTo(output, 0);
                right.Bytes.CopyTo(output, 8);
                return output;
        }

        public static Block[] BytesToBlocks(byte[] bytes)
        {
                if (bytes.Length % 8 != 0)
                        throw new ArgumentException("DivisibleBy8Error", nameof(bytes));

                var blocks = new Block[bytes.Length / 8];
                for (int i = 0; i < bytes.Length; i += 8)
                        blocks[i / 8] = new Block(bytes, i);

                return blocks;
        }

        public static byte[] BlocksToBytes(Block[] blocks)
        {
                var bytes = new byte[blocks.Length * 8];

                for (int i = 0; i < blocks.Length; i++)
                        blocks[i].Bytes.CopyTo(bytes, i * 8);

                return bytes;
        }

        public static Block operator ^(Block left, long right)
        {
                return Xor(left, right);
        }

        public static Block Xor(Block left, long right)
        {
                var result = new Block(left);
                ReverseBytes(result.Bytes);
                long temp = BitConverter.ToInt64(result.Bytes, 0);
                result = new Block(BitConverter.GetBytes(temp ^ right));
                ReverseBytes(result.Bytes);
                return result;
        }

        internal static void ReverseBytes(byte[] bytes)
        {
                for (int i = 0; i < bytes.Length / 2; i++)
                {
                        byte temp = bytes[i];
                        bytes[i] = bytes[(bytes.Length - 1) - i];
                        bytes[(bytes.Length - 1) - i] = temp;
                }
        }
 }
```

Kamstrup • 55123410_A1_EN_01.2023

Kamstrup A/S
Industrivej 28, Stilling
DK-8660 Skanderborg
T: +45 89 93 10 00
info@kamstrup.com
kamstrup.com