Required modifications to CDOC for elliptic curve support

Document no.: A-101-7
Version 1.7
29.11.2017

Introduction

This document describes changes for CDOC format which enable confidentiality with the help of Elliptic Curve Cryptography (ECC).

Currently, CDOC [CDOC] is based on [XMLENC] standard. It uniquely defines

- data encryption using algorithm and mode (AES128-CBC) and
- transport key encryption algorithm (RSA PKCS#1 v1.5).

The ECC support requires to use [XMLENC1] standard which defines a key exchange primitive ECDH.

Using the newer version of the standard, it also becomes possible to use more secure encryption algorithms.

The shortcomings of the current CDOC version are described in [CDOC10]. Specification [CDOC20] describes possible improvements for these shortcomings; however, the required modifications are large and therefore affect application structure significantly.

The changes described in the current document are small and can therefore be implemented faster. From the changes described in document [CDOC10] only encryption algorithms and related modes will be taken (instead of `AES128-CBC`, `AES256-GCM` will be used).

Backward Compatibility

New and old CDOC version software implementation backward compatibility rules are the following.

- New software will support reading files created with the old version.
- New software creates only files which old version cannot read (ECC as well as RSA key based). We assume that most of the users open encrypted files with the help of the official base software, therefore big problems when reading files should not occur after the base software update.

In case the functionality of opening the encrypted files is integrated into some other software package, this software package also needs updating.

In case the functionality of encrypted file creation is integrated into some other software package, its users will only be able to encrypt files to the recipients who have old ID cards with RSA keys. To be able to encrypt files for new ID card holders, the software package needs updating.

Implementations of the previous CDOC format made incorrect use of the EncryptionProperty elements. Attribute `Name` is not allowed for this element. The resulting XML was not conformant to XML Encryption Core Schema.

Example of incorrect usage:

```
<denc:EncryptionProperty
Name="LibraryVersion">qdigidocclient|3.13.2.1498</denc:Encryption
Property>
```

Correct usage would be for example:

```
<denc:EncryptionProperty>
    <LibraryVersion>qdigidocclient|3.13.2.1498</LibraryVersion>
</denc:EncryptionProperty>
```

Applications that implement this specification should not produce old, incorrect XML. Only those applications that must be able to receive and process both old and new formats should support the old, incorrectly formatted XML documents.

This specification does not mandate the usage of specific EncryptionProperty elements.

Applications that should support both old and new formats can base the decision which codepath to use for processing and decrypting of the document on several markers. Since the decision should be done at the very beginning of the processing, before the document has been parsed, the most useful test can be based on the presence of the incorrect `EncryptionProperty` elements. Whenever such element is present old codepath must be used. Otherwise document should be processed as new.

It is expected that new applications do not hardcode processing of the `EncryptedData` element. I.e. values and methods specified in this document should be seen as a profile of [XMLENC1] that can be changed in later versions. Applications can implement only subset of the `EncryptedData` processing and whenever they encounter algorithm or construct that they do not support, should report it as not supported and not to claim that document is invalid.

Data Encryption

The file is encrypted using one random 256-bit (32-byte) symmetric key. For encryption, AES256-GCM algorithm [SP800-38D] is used. Encryption and decryption are performed the way described in section 5.2.4 of [XMLENC1].

The 12-byte initialization vector is generated randomly.

It should be taken into account that using AES256-GCM algorithm, the largest file that can be encrypted is 2^39-256 bits or almost 64 gigabytes [SP800-38D].

Data Decryption

Before returning any plaintext data, the decryption must verify that the GMAC authentication code calculated over the ciphertext matches the 16-byte checksum appended at the end of the ciphertext. In case the checksum does not match, an error shall be given instead of returning the plaintext.

Transport Key Encryption for RSA Certificate Owners

For the owners of RSA certificates, transport key is encrypted with the recipient's public key, using PKCS#1 v1.5 padding scheme, and the result is formatted as described in [CDOC]. The result is an `EncryptedKey` element where `EncryptionMethod` element denotes the encryption algorithm `http://www.w3.org/2001/04/xmlenc#rsa-1_5`.

Against the padding scheme used in PKCS#1 v1.5, the so-called Bleichenbacher attack [Bleichenbacher98] is known. Its applicability and optimization possibilities against Estonian ID card were analyzed by Bardou et al. in 2012 [BFKSST12].

In this attack the attacker uses ID card as an oracle, submitting his decryption queries and receiving response info about successful decryption (but not necessarily decrypted files).

As a result of acquiring a large number of such queries, the attacker could decrypt one file which query was not among in the original ones. It is important to note that the secret key itself does not leak as a result of this attack.

Bardou et al. estimated that such an attack against one ID card requires 28300 queries, or around 27 hours in total. Note that for a successful query, the attacker also needs the PIN1 code. The same way the attacker having access to an ID card and its PIN1 code could just decrypt the needed file, which makes the described attack against the ID card essentially worthless.

Transport Key Encryption for ECC Certificate Owners

For the owners of ECC certificates, the AES256-GCM transport key is encrypted with KW-AES256 key wrapping algorithm using 32-byte key derived from the 48-byte shared secret. The shared secret is calculated using ECDH key exchange algorithm involving the sender's ephemeral ECC key and the public ECC key from the recipient's authentication certificate.

ECDH key exchange uses the same key which is also used for authentication. Since ECC authentication in turn relies on ECDSA signature scheme, this solution rises the question whether the use of ECDH and ECDSA with the same key is secure.

This question was studied in 2011 by Degabriele et al [DLPSS11]. They gave ECDH and ECDSA co-usage security proof in the so-called generic group model, where the attacker is allowed to use only elliptic curve point group operations.

Generic group model is weaker than the so-called concrete model, where the attacker can also access the implementation, use algebraic properties of the specific curve, etc.

At the same time, the best known attacks against elliptic curve cryptography are not able to exploit the operations beyond these of the generic group. This is, among other things, also the reason why cryptographic primitives relying on elliptic curves allow us to use much shorter keys compared to RSA.

It is also worth noting that the security of ECDSA signature scheme itself is proved only in generic group model [HMV06].

ECC Transport Key Encryption

Encryption process is the following.

1. Sender takes from the recipient's certificate his ECC public key `Rp` and the description of the used curve.
2. Sender generates an ephemeral ECC private and public key pair `(Ss,Sp)` using the same curve as used by the recipient. For each ECC recipient, a new ephemeral key is generated.
3. Sender calculates (in memory) the result of ECDH key exchange operation `Ksr`, using his own ephemeral private key `Ss` and recipient's public key `Rp`.
4. Sender derives 32-byte wrapping key from the shared secret `Ksr`. For key derivation, the algorithm `http://www.w3.org/2009/xmlenc11#ConcatKDF` is used. The key derivation algorithm and the choosing logic for its parameters is explained in document [SP800-56Ar2] section 5.8 and appendix B. In the context of this specification, the derived key is linked to this algorithm specification, sender's identifier and recipient's identifier.
5. Sender encrypts the 32-byte AES256-GCM transport key with the 32-byte wrapping key. See [XMLENC1], section 5.7.
6. Sender forms `EncryptedKey` element with the following subelements.
    - `EncryptionMethod` encryption algorithm is `http://www.w3.org/2001/04/xmlenc#kw-aes256`.
    - `CipherValue` is the encrypted transport key.
    - In the `KeyInfo` there is the `AgreementMethod` element, where
        - `Algorithm` attribute denotes the algorithm `http://www.w3.org/2009/xmlenc11#ECDH-ES` and
        - `xenc11:KeyDerivationMethod` subelement denotes the key derivation algorithm `http://www.w3.org/2009/xmlenc11#ConcatKDF`.
            - `xenc11:ConcatKDFParams` subelement determines key derivation algorithm parameters.
                - The hash function used for key derivation is `http://www.w3.org/2001/04/xmlenc#sha384`. The choice of the hash function is based on the recommendation in the document [SP800-56Ar2] section 5.8.1 table 7.
                - `AlgorithmID` attribute value is set to the byte string "ENCDOC-XML|1.1".
                - `PartyUInfo` attribute value is the sender's public key (base64-decoded `PublicKey` value of the `OriginatorKeyInfo` element).
                - `PartyVInfo` attribute value is the recipient's certificate (base64-decoded `X509` value of the `RecipientKeyInfo` element).
        - Under `OriginatorKeyInfo` there is the `dsig11:ECKeyValue` subelement, having the value of the public key `Sp` of sender's ephemeral ECC key pair.
        - Under `RecipientKeyInfo` subelement there is the subelement `ds:X509Data` containing the recipient's certificate.

See also [XMLENC1] EXAMPLE 42.

[SP800-56Ar2] does not specify how additional attributes should be concatenated and if and how the length of invididual attributes should be encoded. Several options are described, but the choice of the particular method is left to the applications. Several publicly available libraries ([MSRJSCL], [JOSE4J]) that implement ConcatKDF algorithm just concatenate the attributes without any information about the length of the attributes. For ease of implementation and interoperability this option should also be used for CDOC.

ECC Transport Key Decryption

Decryption process is the following.

1. Recipient finds the `EncryptedKey` element where the subelement `X509` of the subelement `RecipientKeyInfo` identifies the recipient's authentication certificate, for which he has the corresponding private key.
2. Recipient verifies that it understands and supports all the algorithms and parameters used on the sender's side.
   - Data encryption algorithm is `http://www.w3.org/2009/xmlenc11#aes256-gcm`.
   - `EncryptionMethod` element denotes the key encryption algorithm `http://www.w3.org/2001/04/xmlenc#kw-aes256`.
   - `AgreementMethod` element algorithm is `http://www.w3.org/2009/xmlenc11#ECDH-ES`.
   - `xenc11:KeyDerivationMethod` subelement denotes the key derivation algorithm `http://www.w3.org/2009/xmlenc11#ConcatKDF`.
   - `xenc11:ConcatKDFParams` hash function is `http://www.w3.org/2001/04/xmlenc#sha384`.
3. Recipient verifies attributes of the `ConcatKDFParams` element.
   - `AlgorithmID` attribute value must match the byte string "ENCDOC-XML|1.1".
   - `PartyUInfo` attribute value must match the sender's public key in `OriginatorKeyInfo` element.
   - `PartyVInfo` attribute value must match the recipient's certificate specified in the `RecipientKeyInfo` element.
4. From `OriginatorKeyInfo` the recipient takes the subelement `dsig11:ECKeyValue` containing the sender's ephemeral public ECC key `Sp`.
5. Recipient performs in his ID card the ECDH key exchange, using private authentication key `Rs` stored on the card, and the sender's public key `Sp`. Key exchange result is the shared secret `Ksr`.
   - Note that, before performing the cryptographic operation of generating `Ksr` from `Rs` and `Sp`, the recipient should verify that `Sp` actually encodes a valid point on the used elliptic curve. If this check is not performed, the system becomes vulnerable to invalid curve attacks ([BMM2000], [JSS2015]). The currently used ID card implements such a verification, but it is important to demand this functionality also from the future generations of ID cards.

6. Recipient derives the transport key wrapping key from the shared secret `Ksr`. Sender must have been using the `http://www.w3.org/2009/xmlenc11#ConcatKDF` algorithm.
7. Recipient uses the derived key to decrypt the wrapped transport key. Sender must have been using the `http://www.w3.org/2001/04/xmlenc#kw-aes256` algorithm.
8. With the obtained transport key, the data is decrypted using the algorithm `http://www.w3.org/2009/xmlenc11#aes256-gcm`.

References

- **[XMLENC]** "XML Encryption Syntax and Processing", https://www.w3.org/TR/xmlenc-core/
- **[XMLENC1]** "XML Encryption Syntax and Processing Version 1.1", https://www.w3.org/TR/xmlenc-core1/
- **[CDOC]** "Encrypted DigiDoc Format Specification, Document Version 1.1" https://www.id.ee/public/SK-CDOC-1.0-20120625_EN.pdf
- **[CDOC10]** "CDOC 1.0 Notes and caveats", https://github.com/martinpaljak/idcrypt/wiki/CDOC-1.0
- **[CDOC20]** "CDOCv2 DRAFT v0.1", https://github.com/martinpaljak/idcrypt/wiki/CDOC-2.0
- **[SP800-56Ar2]** "NIST Special Publication 800-56A Revision 2. Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography", http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-56Ar2.pdf
- **[SP800-38D]** "NIST Special Publication 800-38D. Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC", http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-38d.pdf
- **[BMM2000]** Ingrid Biehl, Bernd Meyer, and Volker Müller. "Differential fault attacks on elliptic curve cryptosystems." In Advances in Cryptology—CRYPTO 2000, pp. 131-146. Springer Berlin/Heidelberg, 2000.
- **[Bleichenbacher98]** Daniel Bleichenbacher: Chosen Ciphertext Attacks against Protocols Based on the RSA Encryption Standard PKCS #1. In: Krawczyk, H. (ed.) CRYPTO 1998. LNCS, vol. 1462, pp. 1–12. Springer, Heidelberg (1998).
- **[BFKSST12]** Romain Bardou, Riccardo Focardi, Yusuke Kawamoto, Lorenzo Simionato, Graham Steel, Joe-Kai Tsay: Efficient Padding Oracle Attacks on Cryptographic Hardware. In CRYPTO 2012. LNCS, vol. 7417, pp. 608-625. Springer, Heidelberg (2012).
- **[DLPSS11]** Jean Paul Degabriele, Anja Lehmann, Kenneth G. Paterson, Nigel P. Smart, and Mario Strefler. "On the joint security of encryption and signature in EMV." Cryptology ePrint Archive: Report 2011/615, https://eprint.iacr.org/2011/615
- **[HMV06]** Darrel Hankerson, Alfred J. Menezes, and Scott Vanstone. Guide to elliptic curve cryptography. Springer Science & Business Media, 2006.
- **[JSS2015]** Tibor Jager, Jörg Schwenk, and Juraj Somorovsky. "Practical invalid curve attacks on TLS-ECDH." In European Symposium on Research in Computer Security, pp. 407-425. Springer International Publishing, 2015.
- **[MSRJSCL]** "MSR JavaScript Cryptography Library", https://www.microsoft.com/en-us/

research/project/msr-javascript-cryptography-library/

- **[JOSE4J]** "jose.4.j", https://bitbucket.org/b_c/jose4j/wiki/Home