EUROPEAN COMMISSION

DIRECTORATE-GENERAL FOR INFORMATICS

eDelivery AS4 Security Profile

Update Recommendations (January 2023)

Commission européenne/Europese Commissie, 1049 Bruxelles/Brussel, BELGIQUE/BELGIË – Tel. +32 22991111

DOCUMENT HISTORY

Date	Version	Modification	Authors
29 January 2023	1.0	Final submitted version	Juraj Somorovsky, Hackmanit / Paderborn University Jost Rossel, Hackmanit Pim van der Eijk

UPDATING AS4 PROFILE SECURITY SECTIONS

The European Commission's eDelivery Building Block intends to modernize its AS4 profile to use newer, state-of-the-art security algorithms. This is intended to allow continued secure use by the user communities in the coming years. This document describes how the specifications of eDelivery could be updated. It is based on input from cryptographic experts working under contract for the EU and reviews from Commission experts.

1. UPDATE OF THE AS4 SECURITY ALGORITHM SECTION

Note: this is intended to be an update for content in eDelivery AS4, section 3.2.6: https://ec.europa.eu/digital-building-blocks/wikis/display/DIGITAL/eDelivery+AS4+-+1.15.

The section numbers are marked as X.Y[.Z[.N]] to indicate that they need to be adjusted to the eventual numbering that will be used in the updated eDelivery AS4 profile.

X.Y Security

AS4 message exchanges can be secured at multiple communication layers: the network layer, the transport layer, the message layer, and the payload layer. The first and last of these are not normally handled by B2B communication software and are therefore out of scope for this sub-section. In this AS4 profile, message exchange MUST be secured using both transport layer security and message layer security. Transport layer security MAY be offloaded to another infrastructure component.

This section provides parameter settings based on multiple published sets of best practices. It is noted that after the publication of this specification, previously unknown vulnerabilities may be discovered in the security algorithms, formats, and exchange protocols specified in this section. Such discoveries MUST lead to revisions to this specification.

X.Y.Z Transport Layer Security

When using AS4, Transport Layer Security (TLS) provides content confidentiality and authentication. Server authentication, using a server certificate, allows the client to make sure the HTTPS connection is set up with the right server. When a message is pushed, the Sending MSH authenticates the HTTPS server of the Receiving MSH.

TLS can be directly handled by the AS4 message handler or be off-loaded to some infrastructure component. In the following, we refer to the TLS processing component as TLS implementation. For every TLS implementation conformant with this profile, the following rules shall apply:

- TLS versions and cipher suites MUST follow international and national minimum standard requirements and best practices such as [RFC9325], [ECRYPT CSA], [NIST 800-52r2] and [BSI TR-02102-2]. The decision which, if any, of these publications to follow is not specified in this profile as it may depend on other international, national and/or sectorial regulation or other factors.

- It MUST be possible to configure the accepted TLS version(s) in the TLS implementation.

- It MUST be possible to configure accepted TLS cipher suites in the TLS implementation. Note that naming conventions and recommendations for suites are specific to TLS versions.

X.Y.Z.1 TLS Versions

Implementations conformant with this profile:

- MUST NOT use SSL 3.0, TLS 1.0 and 1.1.

- MUST therefore at a minimum support TLS 1.2 [RFC5246]. TLS 1.2 is considered sufficient and offers good cryptographic primitives. With proper configuration of cipher suites it is considered sufficient for many years.

- SHOULD support the use of TLS 1.3 [RFC8446]. Note that [NIST 800-52r2] requires support for TLS 1.3 as from January 1, 2024.

X.Y.Z.2 TLS Cipher Suites

Implementations conformant with this profile should support the following TLS 1.3 cipher suites:

- TLS_AES_128_GCM_SHA256

- TLS_AES_256_GCM_SHA384

- TLS_AES_128_CCM_SHA256

These cipher suites are recommended by [BSI TR-02102-2] and [NIST 800-52r2]. Note that [ECRYPT CSA] does not make any explicit restrictions regarding TLS 1.3 cipher suites. [RFC9325] recommends to follow the recommendations from [RFC8446].

In addition, TLS_CHACHA20_POLY1305_SHA256 may be used [RFC8446].

For TLS 1.2, this profile recommends the usage of Perfect Forward Secure (PFS) cipher suites. Implementations conformant with this profile should support the following TLS 1.2 cipher suites:

- TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384

- TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256

- TLS_ECDHE_ECDSA_WITH_AES_256_CCM

- TLS_ECDHE_ECDSA_WITH_AES_128_CCM

- TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384

- TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256

These cipher suites are compatible with the recommendations of [BSI TR-02102-2], [NIST 800-52r2], and [ECRYPT CSA].

Further cipher suites may be used when following specific regulations. For example, [ECRYPT CSA] recommends the usage of Camellia for record layer encryption. [BSI TR-02102-2], [NIST 800-52r2], and [ECRYPT CSA] recommends the usage of TLS_DHE_* cipher suites.

X.Y.Z.3 Supported Groups for (EC)DH Key Exchange

Implementations conformant with this profile should support the following elliptic curves:

- secp256r1
- secp384r1
- secp521r1
- x25519
- x448

When using Finite Field Diffie Hellman, at least ffdhe3072 should be used.

X.Y.Z.4 Certificate Key Lengths

Implementations conformant with this profile must use RSA, ECDSA, or EdDSA X.509 certificates. For RSA certificates, keys larger than 3000 bits are mandatory. For ECDSA, keys larger than 250 bits are mandatory.

X.Y.Z.5 TLS Client Authentication

Transport Layer client authentication authenticates the Sender (when used with the Push MEP binding) or Receiver (when used with Pull). Since this profile uses WS-Security for message authentication, the use of client authentication at the Transport Layer can be considered redundant. Whether or not client authentication is to be used depends on the deployment environment. To support deployments that do require client authentication, implementations MUST allow Transport Layer client authentication to be configured for an AS4 HTTPS endpoint. Mutual Authentication or “two way” TLS Authentication is a combination of client and server authentication.

X.Y.Z. Message Layer Security using WS-Security

To provide message layer protection for AS4 messages, this profile REQUIRES the use of the following Web Services Security version 1.1.1 OASIS specifications, profiled in ebMS3.0 [EBMS3] and AS4 [AS4]:

- Web Services Security SOAP Message Security [WSSSMS].
- Web Services Security X.509 Certificate Token Profile [WSSX509].
- Web Services Security SOAP Message with Attachments (SwA) Profile [WSSSWA].

The X.509 Certificate Token Profile supports the signing and encryption of AS4 messages. This profile REQUIRES the use of X.509 tokens for message signing and encryption, for all AS4 exchanges. The AS4 option of using Username Tokens, which is supported in the AS4 ebHandler Conformance Profile, MUST NOT be used. The AS4 message MUST be signed prior to being encrypted (see section 7.6 of [EBMS3CORE]).

X.Y.Z. Message Signing

AS4 message signing is based on the W3C XML Signature recommendation used by WS-Security. AS4 can be configured to use specific digest and signature algorithms based on identifiers defined in this recommendation. At the time of publication of the AS4 specification [AS4], the current version of W3C XML Signature was the June 2008, XML Signature, Second Edition specification [XMLDSIG]. The current version is the April 2013, Version 1.1 specification [XMLDSIG1], which defines important new algorithm identifiers. In addition, Ed25519 and Ed448 algorithms are available based on [RFC8410] and [RFC9231].

This AS4 profile uses the following AS4 parameters and values:

- The **PMode[].Security.X509.Sign** parameter MUST be set in accordance with section 5.1.4 and 5.1.5 of [AS4].

- The **PMode[].Security.X509.Signature.HashFunction** parameter MUST be set to http://www.w3.org/2001/04/xmlenc#sha256.

- The **PMode[].Security.X509.Signature.Algorithm** parameter MUST be set to http://www.w3.org/2021/04/xmldsig-more#eddsa-ed25519.

This AS4 profile anticipates an update to the OASIS AS4 specification to reference this newer version of the XML Signature specification.

The use of XML Signature in AS4 provides Non Repudiation of Origin (NRO) at Message Exchange level.

Note that the usage of the Ed25519 curve implies that the message signer has an EdDSA certificate using the Ed25519 curve. This certificate is signed by a CA that might use a different signing algorithm (RSA or ECDSA). This standard does not prescribe any algorithms for CAs.

X.Y.Z. Message Encryption

For encryption, WS-Security leverages the W3C XML Encryption recommendation used by WS-Security. The following AS4 processing mode parameters configure this feature:

- The **PMode[].Security.X509.Encryption.Encrypt** parameter MUST be set in accordance with section 5.1.6 and 5.1.7 of [AS4].

- The parameter **PMode[].Security.X509.Encryption.Algorithm** MUST be set to http://www.w3.org/2009/xmlenc11#aes128-gcm. This is the algorithm used as value for the **Algorithm** attribute of **xenc:EncryptionMethod** on **xenc:EncryptedData.** This means that in this profile, AES MUST NOT be used in CBC mode.

As specified in section 5.1.6 of [AS4] and in https://issues.oasis-open.org/browse/EBXMLMSG-111, when XML Encryption is used, all and only payload MIME parts MUST be encrypted. The **eb:Messaging** header and any of its sub-elements MUST NOT be encrypted at message layer. Note that this header remains encrypted at transport layer.

In WS-Security, there are three mechanisms to reference a security token (see section 3.2 in [WSSX509]). The ebMS3 and AS4 specifications do not constrain this; neither do they provide a P-Mode parameter to select a specific option. For interoperability, implementations SHOULD therefore implement all three options. It is RECOMMENDED that implementations allow configuration of security token reference type, so that a compatible type can be selected for a communication partner. Note that as *BinarySecurityToken* is the most widely implemented option for security token references in AS4 implementations, implementations SHOULD implement this option.

In this version of this AS4, message encryption is based on the Elliptic Curve Diffie-Hellman Key Exchange algorithm.

- For the encryption algorithm, http://www.w3.org/2001/04/xmlenc#kw-aes128. This is the algorithm used as a value for the Algorithm attribute of **xenc:EncryptionMethod** in **xenc:EncryptedKey**. It describes the key encryption key.

- For the key agreement method, http://www.w3.org/2009/xmlenc11http://www.w3.org/2021/04/xmldsig-more#x25519. This is the algorithm used as value for the Algorithm attribute of **xenc:AgreementMethod** in **ds:KeyInfo**.

- When using x25519 public keys, the originator key info has a **ds:KeyValue** containing a **ds11:ECKeyValue** element. That element has a **ds11:NamedCurve** with *URI* set to *urn:oid:1.3.101.110*.

- For the key derivation method, the http://www.w3.org/2009/xmlenc11#ConcatKDF MUST be used. This is the algorithm used as a value for the *Algorithm* attribute of **xenc11:KeyDerivationMethod** in **xenc:AgreementMethod**.

- The values of the attributes *AlgorithmID*, *PartyUInfo* and *PartyVInfo* of the **xenc11:ConcatKDFParams** element MUST be set to empty strings.

In the base implementation, ECDH is used in so-called ephemeral-static mode (ECDH-ES) in which the sender creates an agreed encryption key based on a short-lived sender key in combination with a long-lived recipient key.

Alternatively, optionally, sender or recipient may use ebCore Certificate Update to update the static key frequently, as explained below in section X.Y below.

X.Y. Security Header Processing Example

A sending MSH performs security processing and constructs the security header as follows:

1. The message parts that are to be signed (header, empty body and MIME parts) are selected in accordance with AS4.

2. Message digests are computed for all parts following [WSSSWA].

3. A **SignedInfo** section is created and the message is signed using sender's signing key, determined from the applicable P-Mode. (As noted below in X.Y, the static P-Mode configuration may be updated prior to its expiration using ebCore Certificate Update).

4. A per-message ephemeral originator key agreement key is constructed of the required curve type.

5. The recipient's static public key information is determined from the applicable P-Mode. (As noted below in X.Y, the static public key agreement key may be frequently updated using ebCore Certificate Update).

6. A shared secret is constructed from the two keys using key ECDH-ES agreement.

7. The shared secret is used as an input into the key derivation method (ConcatKDF) to derive an AES key wrap key.

8. An AES symmetric key is generated at random.

9. The AES key is wrapped and used to encrypt the MIME payload parts following [WSSSWA].

10. An **EncryptedData** element is added representing the parts encryption.

The resulting WS-Security header might look as follows:

```
<wsse:Security xmlns:env="http://www.w3.org/2003/05/soap-envelope"
    xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd"
    xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
    env:mustUnderstand="true">

    <xenc:EncryptedKey xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
        xmlns:xenc11="http://www.w3.org/2001/04/xmlenc#"
        wsu:Id="EK-6263cc2e-e01a-4bd2-a2f3-39f9c74e82ab">
        <xenc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#kw-aes128"/>
        <ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#" Id="KI-c0afa373">
            <xenc:AgreementMethod Algorithm="http://www.w3.org/2009/xmlenc11#ECDH-ES">
                <xenc11:KeyDerivationMethod Algorithm="http://www.w3.org/2009/xmlenc11#ConcatKDF">
                    <xenc11:ConcatKDFParams AlgorithmID="" PartyUInfo="" PartyVInfo="">
                        <ds:DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256"/>
                    </xenc11:ConcatKDFParams>
                </xenc11:KeyDerivationMethod>
                <xenc:OriginatorKeyInfo>
                    <ds:KeyValue>
                        <ds11:ECKeyValue xmlns:ds11="http://www.w3.org/2021/04/xmldsig-more#x25519">
                            <!-- Public ephemeral X25519 key.
                                See http://oid-info.com/get/1.3.101.110 and RFC 8410
                            -->
                            <ds11:NamedCurve URI="urn:oid:1.3.101.110"/>
                            <ds11:PublicKey> ENCODED </ds11:PublicKey>
                        </ds11:ECKeyValue>
                    </ds:KeyValue>
                </xenc:OriginatorKeyInfo>
                <xenc:RecipientKeyInfo>
                    <ds:KeyValue>
                        <!-- Assumes the recipient key is exchanged using ebCore or
                            some other mechanism.  It has therefore has been shared as a
                            certificate and can be referenced using its SKI.
                        -->
                        <wsse:SecurityTokenReference>
                            <wsse:KeyIdentifier
                                EncodingType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0#Base64Binary"
                                ValueType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509SubjectKeyIdentifier"
                                > ENCODED </wsse:KeyIdentifier>
                        </wsse:SecurityTokenReference>
                    </ds:KeyValue>
                </xenc:RecipientKeyInfo>
            </xenc:AgreementMethod>
        </ds:KeyInfo>
        <xenc:CipherData>
            <xenc:CipherValue>ENCODED</xenc:CipherValue>
```

```
    </xenc:CipherData>
    <xenc:ReferenceList>
        <xenc:DataReference URI="#ED-ad394cf3-a2c0-442e-9943-f01cea6782cb"/>
    </xenc:ReferenceList>
</xenc:EncryptedKey>

 <xenc:EncryptedData xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
    Id="ED-ad394cf3-a2c0-442e-9943-f01cea6782cb" MimeType="application/gzip"
    Type="http://docs.oasis-open.org/wss/oasis-wss-SwAProfile-1.1#Attachment-Content-Only">
    <xenc:EncryptionMethod Algorithm="http://www.w3.org/2009/xmlenc11#aes128-gcm"/>
    <ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
        <wsse:SecurityTokenReference
            xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd"
            xmlns:wsse11="http://docs.oasis-open.org/wss/oasis-wss-wssecurity-secext-1.1.xsd"
            wsse11:TokenType="http://docs.oasis-open.org/wss/oasis-wss-soap-message-security-1.1#EncryptedKey">
            <wsse:Reference URI="#EK-6263cc2e-e01a-4bd2-a2f3-39f9c74e82ab"/>
        </wsse:SecurityTokenReference>
    </ds:KeyInfo>
    <xenc:CipherData>
        <xenc:CipherReference URI="cid:1400668830234@seller.eu">
            <xenc:Transforms>
                <ds:Transform xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
                    Algorithm="http://docs.oasis-open.org/wss/oasis-wss-SwAProfile-1.1#Attachment-Ciphertext-Transform"
                />
            </xenc:Transforms>
        </xenc:CipherReference>
    </xenc:CipherData>
 </xenc:EncryptedData>

 <wsse:BinarySecurityToken
    EncodingType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0#Base64Binary"
    ValueType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509v3"
    wsu:Id="X509-48b6d459-777b-4226-81bd-df327f37b30c"
    > ENCODED </wsse:BinarySecurityToken>
 <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
    Id="SIG-adcdc058-ddac-4437-8902-ab37cf037ca4">
    <ds:SignedInfo>
        <ds:CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#">
            <ec:InclusiveNamespaces xmlns:ec="http://www.w3.org/2001/10/xml-exc-c14n#"
                PrefixList="env"/>
        </ds:CanonicalizationMethod>
        <ds:SignatureMethod Algorithm="http://www.w3.org/2021/04/xmldsig-more#eddsa-ed25519"/>
        <ds:Reference URI="#_840b593a-a40f-40d8-a8fd-89591478e5df">
            <!-- The (empty) SOAP body -->
            <ds:Transforms>
                <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
```

```
            </ds:Transforms>
            <ds:DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256"/>
            <ds:DigestValue>jyTXyVrh+cX3iJzgmxqiHdnnJQxcX6kTGHPES1YUYEs=</ds:DigestValue>
        </ds:Reference>
        <ds:Reference URI="#_210bca51-e9b3-4ee1-81e7-226949ab6ff6">
            <!-- the AS4 eb:Messaging header -->
            <ds:Transforms>
                <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
            </ds:Transforms>
            <ds:DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256"/>
            <ds:DigestValue>5RMz5/mSIFTI1+amk+XLHsLR2yE7h5KFgAsLrHrya98=</ds:DigestValue>
        </ds:Reference>
        <ds:Reference URI="cid:1400668830234@seller.eu">
            <!-- A message payload in a MIME attachment -->
            <ds:Transforms>
                <ds:Transform
                    Algorithm="http://docs.oasis-open.org/wss/oasis-wss-SwAProfile-1.1#Attachment-Content-Signature-Transform"
                />
            </ds:Transforms>
            <ds:DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256"/>
            <ds:DigestValue>wVgT8wKEsJlO0O5OjjQB/vw9mGsxi1n/0dc9qeRqFM4=</ds:DigestValue>
        </ds:Reference>
    </ds:SignedInfo>
    <ds:SignatureValue>CyVaSr9BLh7m4KC7xNszOsmJNM6aNJPKwQwNNqY5cvu3GgSIYBQWecg==</ds:SignatureValue>
    <ds:KeyInfo Id="KI-29066baf-2595-444f-9d27-58667dc40da3">
        <wsse:SecurityTokenReference wsu:Id="STR-a54b721a-0d19-4112-b1cf-06752cd826fa">
            <wsse:Reference URI="#X509-48b6d459-777b-4226-81bd-df327f37b30c"
                ValueType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509v3"
            />
        </wsse:SecurityTokenReference>
    </ds:KeyInfo>
  </ds:Signature>
</wsse:Security>
```

The receiving AS4 MSH processes the secured message containing this security header as follows.

1. It identifies the **EncryptedData** element (Id="ED-ad394cf3-a2c0-442e-9943-f01cea6782cb"). In order to decrypt the encrypted data, it needs to process the **EncryptedKey** element that is referenced in the **SecurityTokenReference** element (URI="#EK-6263cc2e-e01a-4bd2-a2f3-39f9c74e82ab").

2. It processes the **AgreementMethod** element in the **EncryptedKey**. Using the **OriginatorKeyInfo** public key value and the private key identified by **RecipientKeyInfo**, it performs the ephemeral-static X25519 key agreement. The result of this operation is used as an input into the **ConcatKDF** key derivation algorithm.

3. The result of **ConcatKDF** can be used to unwrap the key using AES-KW which is located in the **CipherData** element.

4. The receiving corner can now use AES-GCM to decrypt data referenced in **EncryptedData**.

5. It identifies the XML Signature, validates all the references, and the signature value by using the public key from the sender certificate.

2. ENHANCEMENTS

X.Y. Support for OASIS ebCore Certificate Update

Note:

- The eDelivery AS4 profile does not yet support this feature. The proposal is to add it as an Optional Profile Enhancement.

- To support perfect forward secrecy in an ECDH-ES setting, ebCore Certificate update can be used to also update the recipient public key.

- As PFS requires frequent updates (i.e., not once every three years, but at most once every few days), automation is essential as otherwise it becomes a major bottleneck.

- The use of ebCore Certificate Update is conditional to the service being configured.

X.Y.Z. Introduction

To support for ebCore Certificate Update [ebcore-au-v1.0] according to this AS4 profile, the AS4 MSH:

- MUST be able to exchange ebCore Certificate Update AS4 messages. As AS4 is payload-agnostic, this imposes no special requirements on products. The only requirement on implementers deploying AS4 products is that these messages

MUST use the Service and Action values specified in sections X.Y.Z and X.Y.Z., respectively.

- MUST provide functionality to create an ebCore Certificate Update document; use it to submit an update to an AS4 configuration; convert the success/failure of such an update to a positive/negative ebCore response document; provide an interface to the AS4 MSH for submission and delivery of ebCore documents exchanged with communication partners.

- MUST sign ebCore Certificate Update messages as any AS4 message conformant to the profile.

The ebCore Certificate Update protocol can be used in conjunction with this profile for two purposes:

- As a mechanism to update a long-lived certificate containing a public key that is used, among others, for signing. In this case, the certificate is required to be issued by a Certification Authority and the update frequency is low (typically once every few years or months).

- As a mechanism to update a certificate containing a public key of a party that is only used for key agreement to support encryption. In this case, the certificate may be issued by the party rather than by a Certification Authority. As it is exchanged in an ebCore Certificate Update message, the integrity and authenticity of the public key are protected.

In addition to supporting updating the certificate used for AS4 message signing, ebCore Certificate Update MAY be used to update the static key of the recipient used in the ephemeral-static key exchange. In ideal cryptographic protocols, ephemeral keys are only used once for establishing symmetric keys. This is not possible in the base AS4 profile for recipients as there may not be a secure channel to transmit such keys.

Still, it is RECOMMENDED to change ephemeral keys as frequently as possible, giving potential attackers less chance to break previous messages. Therefore, it is RECOMMENDED to use ebCore Certificate Update to update keys such that keys are replaced within 7 days. The 7-day limit is the maximum lifetime TLS 1.3 [RFC8446] uses for session tickets which effectively break forward secrecy of TLS connections.

Automatic processing of ebCore Certificate Update messages (i.e., processing of update requests not requiring intervention by a human operator or non-immediate service management process) allows low-overhead, frequent updates of the static key contained in the certificate for the recipient for key exchange. The static key in practice approximates an ephemeral key.

While ebCore Certificate Update packages keys using certificates, the certificates containing ECDH public keys do not need to be signed by a certification authority. As they are issued using signed ebCore Agreement Update messages, their authenticity is established.

X.Y. Support for alternative curves and algorithms

In order to provide a fall-back for the (highly unlikely) situation in which vulnerabilities are found in the algorithms for signing (based on Ed25519) or encryption (based on

X25519), or to facilitate interoperability with other AS4 profiles such as [BDEW AS4], support in AS4 products supporting this profile MAY support other curves and algorithms.

A variant supporting curves supporting BrainpoolP256r1 may be provided that differs from this security section as follows:

- The signature algorithm is set to http://www.w3.org/2001/04/xmldsig-more#ecdsa-sha256.

- When including public keys based on BrainpoolP256r1 curves, the value of the *URI* attribute on **NamedCurve** is to be set to *urn:oid:1.3.36.3.3.2.8.1.1.7*.

It MUST be possible to select the use of either the profiling in the security algorithm section defined above in X.Y.Z. or the alternative curves and algorithms using the AS4 P-Mode mechanism and related agreement reference header.

3. BIBLIOGRAPHY

[BDEW AS4] BDEW Profile. AS4-Nutzungsprofil zum Datenaustausch für regulierte Prozesse in der Energiewirtschaft. https://www.edi-energy.de/index.php?id=38&tx_bdew_bdew%5Buid%5D=1608&tx_bdew_bdew%5Baction%5D=download&tx_bdew_bdew%5Bcontroller%5D=Dokument&cHash=5fbee16dcbd284d5f9899875d50353de.

[BSI BSI TR-02102-1] Cryptographic Mechanisms: Recommendations and Key Lengths. https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/Publications/TechGuidelines/TG02102/BSI-TR-02102-1.html

[BSI TR-02102-2] Cryptographic Mechanisms: Recommendations and Key Lengths: Use of Transport Layer Security (TLS)" Version: 2022-1". https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/Publications/TechGuidelines/TG02102/BSI-TR-02102-2.html

[ECRYPT CSA] H2020-ICT-2014 – Project 645421. Algorithms, Key Size and Protocols Report (2018). https://www.ecrypt.eu.org/csa/documents/D5.4-FinalAlgKeySizeProt.pdf.

[ebcore-au-v1.0] ebCore Agreement Update Specification Version 1.0. OASIS Committee Specification 01. http://docs.oasis-open.org/ebcore/ebcore-au/v1.0/cs01/ebcore-au-v1.0-cs01.html. Latest version: http://docs.oasis-open.org/ebcore/ebcore-au/v1.0/ebcore-au-v1.0.html.

[eDeliveryAS4] https://ec.europa.eu/digital-building-blocks/wikis/display/DIGITAL/eDelivery+AS4.

[NIST 800-52r2] Guidelines for the Selection, Configuration, and Use of Transport Layer Security (TLS) Implementations. NIST Special Publication 800-52 Revision 2. https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-52r2.pdf.

[RFC5246] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.2 https://www.rfc-editor.org/rfc/rfc5246.

[RFC8410] S. Josefsson and J. Schaad. Algorithm Identifiers for Ed25519, Ed448, X25519, and X448 for Use in the Internet X.509 Public Key Infrastructure. https://www.rfc-editor.org/rfc/rfc8410.

[RFC8446] E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.3. https://www.rfc-editor.org/rfc/rfc8446.

[RFC9231] D. Eastlake 3rd. Additional XML Security Uniform Resource Identifiers (URIs). https://www.rfc-editor.org/rfc/rfc9231.html.

[RFC9325] Y. Sheffer and P. Saint-Andre and T. Fossati. Recommendations for Secure Use of Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS). https://www.rfc-editor.org/rfc/rfc9325.