DV-key format

This paragraph describes the technical format of key file provided to Dienstverleners (Service Providers).

Formats of cryptographic keys provided to Service Providers

Parties that are provided with either Encrypted Identities or Encrypted Pseudonyms are also provided cryptographic keys to decrypt them to respectively Identities or Pseudonyms. These cryptographic keys are provided as ECPrivateKey as specified in RFC5915, in files formatted according to the PEM specifications (RFC1421).

Encrypted key format

Each key(file) containing one DV-key will be provided in an encrypted form, to protect the key from disclosure during the provisioning. The encryption is in CMS as per RFC5652. This format is typically used in PEM or S/MIME format.

The encryption has the following charcteristics:

- Hybrid encryption, using a new random symmetric key per encrypted DV-key-file
- AES-256 in CBC-mode for symmetric encryption
- the AES key is encrypted to the DV with RSA encryption, using the RSA public key from the PKIoverheid certificate in the request.
- This results in 'keyTransport' in CMS for protection of the content-encryption key.
- the CMS reference the Issuer and SerialNumber of the DV-certificate

The structure of the CMS is:

field	value / comment
version	0 (fixed, as per RFC5652)
recipientInfos	1 RecipientInfo for the receiver (DV)
...recipientInfo	van het type KeyTransportRecipientInfo
.....version	0 (fixed, as per RFC5652)
.....rid	IssuerSerial (sequence)
.....Issuer	copy of DV certificate (via Interface spec BSNk: provideDVKeys)
.....Serial	copy of DV certifcate (via Interface spec BSNk: provideDVKeys)
..... keyEncryptionAlgorithm	1.2.840.113549.1.1.7 = rsaOAEP (as per RFC3447?)
.....hashAlgorithm	2.16.840.1.101.3.4.2.1 = SHA-256 as hash function for OAEP
.....maskGenAlgorithm	1.2.840.113549.1.1.8 = PKCS#1 MGF as mask function for OAEP
.....pSourceAlgorithm	2.16.840.1.101.3.4.2.1 = SHA-256 as digest for the mask function for OAEP
.....key	the AES content-encryption key encrypted with RSA-OAEP to the above referenced certificate
...encryptedContentInfo	
.....contenttype	1.2.840.113549.1.7.1 = data (PKCS #7)
..... contentEncryptionAlgorithm	2.16.840.1.101.3.4.1.42 = aes256-CBC (NIST Algorithm, as per RFC3565)
.....iv	16-byte octet string, the IV for the AES encryption
.....encryptedContent	the PEM-encoded content (DV PP-key as described below), encrypted with AES-256-CBC using the content-encryption key included above.

This can be created using openssl:

```
openssl cms -encrypt -aes256 -in key-file.pem -binary -recip dv-cert.pem -keyopt rsa_padding_mode:oaep -keyopt
rsa_oaep_md:sha256 -keyopt rsa_mgf1_md:sha256 -out encrypted-key-file.p7 -outform DER
```

This can be decrypted using openssl:

```
openssl cms -decrypt -in encrypted-key-file.p7 -inkey dv-key.pem -inform DER -out key-file.pem
```

Key file format

The actual cryptographic keys obtained after decryption, have the format described below.

As per RFC5915 (section 4), all DV-key files contain a EC private key object, using the PEM-encoding of the DER-encoded ECPrivateKey structure. The actual key is preceded by the line

```
-----BEGIN EC PRIVATE KEY-----
```

and followed with the line

```
-----END EC PRIVATE KEY-----
```

Headers

These headers use a 'key: value' notation, as specified in RFC1421. Applicable headers MUST be present for a given keytype, unless stated otherwise,

Header name	Meaning	Applicable to keytypes
`SchemeVersion`	Scheme version (see also 2d. Polymorphic Pseudonymization Notation.)	All
`SchemeKeySetVersion`	SchemeKeySetVersion (see also 2d. Polymorphic Pseudonymization Notation.)	All
`Recipient`	OIN of the recipient. (Service Provider) (see also 2d. Polymorphic Pseudonymization Notation.)	All except EP Migration
`RecipientKeySetVersion`	RecipientKeySetVersion for a Service Provider corresponds with the issue date of the PKIoverheid certificate used to request their keys at the party generating the keys within the scheme. The RecipientKeySetVersion will therefore be an 8-digit decimal representation of a date in the YYYYMMDD format. (see also 2d. Polymorphic Pseudonymization Notation.)	All except EP Migration

<table>
<tr><td><code>Type</code></td><td>MUST have one of the following values (case-sensitive):<table><tr><th>Value</th><th>Meaning</th></tr><tr><td><code>EI Decryption</code></td><td>Provided through Interface spec BSNk: provideDVKeys

A decryption key to decrypt an identity (BSN) from an Encrypted Identity. Relying Party must be present on Autorisatielijst BSN in order to receive this key.</td></tr><tr><td><code>EP Decryption</code></td><td>Provided through Interface spec BSNk: provideDVKeys

A decryption key to decrypt a pseudonym from an Encrypted Pseudonym.</td></tr><tr><td><code>EP Closing</code></td><td>Provided through Interface spec BSNk: provideDVKeys

Additional key required to decrypt Encrypted Pseudonyms, that provides enahnced protection of pseudonyms. More info on EP Closing Key usage: 2b. Working with pseudonyms</td></tr><tr><td><code>EP migration source</code></td><td>Provided through Interface spec BSNk: provideDVMigrationKeys

Key to support an organizational migration (change of OIN). This key is be used to calculate an intermediary pseudonym for export, that can be used at the target organization OIN.</td></tr><tr><td><code>EP migration target</code></td><td>Provided through Interface spec BSNk: provideDVMigrationKeys

Key to support an organizational migration (change of OIN). This key is be used to calculate the target organization (OIN) pseudonym from an intermediary pseudonym.</td></tr><tr><td><code>DRKi</code></td><td>Direct Receiving Key. Used to process Direct Encrypted Pseudonyms.</td></tr></table></td><td></td></tr>
<tr><td><code>SourceMigrant</code></td><td>Source OIN in migration</td><td>EP Migration</td></tr>
<tr><td><code>SourceMigrantKeySetVersion</code></td><td>RecipientKeySetVersion of the closing key used at source</td><td>EP Migration</td></tr>
<tr><td><code>TargetMigrant</code></td><td>Target OIN in migration</td><td>EP Migration</td></tr>
<tr><td><code>TargetMigrantKeySetVersion</code></td><td>RecipientKeySetVersion of the closing key used at target</td><td>EP Migration</td></tr>
<tr><td><code>MigrationID</code></td><td>ID of the migration (for identification purpose, can be used to match source and target keys)

Type: String, MUST only contain characters (A-F)(0-9) and '-', minimum length =1</td><td>EP Migration</td></tr>
<tr><td><code>Diversifier</code></td><td>Optional Value of diversifier. Comma-separated list of key-value pairs. Keys and values separated by '='
Example:

<code>Diversifier: O=Logius,R=Tester,U=Bsnk</code>
See Diversifiers for details</td><td>DRKi,

EP Migration</td></tr>
<tr><td><code>SchemeKeyVersion</code></td><td>(Deprecated) Alias for <code>SchemeKeySetVersion</code>. Ignore when <code>SchemeKeySetVersion</code> is present. SchemeKeyVersion will be removed in a future revision of these specifications.</td><td>All</td></tr>
</table>

EC private key format

The key is a ECPrivateKey, defined by

```
ECPrivateKey ::= SEQUENCE {
    version        INTEGER { ecPrivkeyVer1(1) } (ecPrivkeyVer1),
    privateKey     OCTET STRING,
    parameters [0] ECParameters {{ NamedCurve }} OPTIONAL,
    publicKey  [1] BIT STRING OPTIONAL
}
```

thus consisting of 4 elements:

- a version (integer of value 1),
- an OCTET STRING corresponding to the actual private key (40 bytes)
- a tagged OBJECT IDENTIFIER of value 1.3.36.3.3.2.8.1.1.9 indicating the BrainpoolP320r1 curve.
- a tagged BIT STRING representing the corresponding public key (an EC point, uncompressed).

In line with RFC5915, both the curve name and the public key will be provided, even though these are indicated as optional.

We remark that the following OpenSSL (version 1.0.2) command will provide a PEM file corresponding to the above specification without the headers:

```
openssl ecparam -name brainpoolP320r1 -genkey -noout -out brainpoolP320r1key.pem
```

and key files can be read by the command

```
openssl ec -in /tmp/ec.pem -text
```

The confidentiality and authenticity of the key files in transport to the intended parties is required but not further specified. On receipt of a PEM file the intended party MUST validate that the PEM is correctly formed which also includes that the provided private key and public key match. That is, if x represents this private key and G the BrainpoolP320r1 generator, then the intended party needs to validate that x*G is equal to the public key in the key file. The recipient furthermore MUST ensure each key is adequately secured against disclosure or abuse.

Example EP decryption key file

```
-----BEGIN EC PRIVATE KEY-----
SchemeVersion: 1
SchemeKeySetVersion: 1
Type: EP Decryption
Recipient: 00000003123456780000
RecipientKeySetVersion: 20161201
SchemeKeyVersion: 1

MIGQAgEBBCikKZiJyHs5btM2JLTS3V7E7Kb3TarWX7yW8Scg1WbtdF+2b30UReEn
oAsGCSskAwMCCAEBCaFUA1IABLPWbbemrGwC5Cz02dq6XBRW8LQieNGRrgMDeLQv
or9OmLnEnPEJfiYELjxyYq1hFjrarWW2/t98sujlBImGMKifQvnT7mgp6jVDrPtF
Kt3J
-----END EC PRIVATE KEY-----
```

Note: OpenSSL PEM_read function (https://www.openssl.org/docs/man1.1.0/crypto/PEM_read.html) supports this format.