European Commission

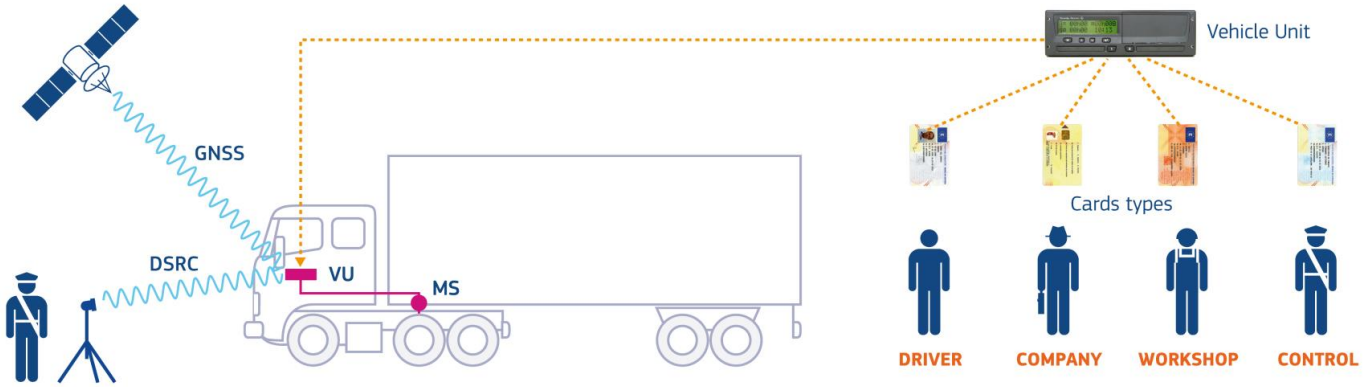
JRC TECHNICAL REPORTS

Smart Tachograph

User manual for the sample cryptographic keys and digital certificates Generation Tool

Klaas Mateboer, David Bakker (UL)
Luigi Sportiello (JRC)

Version 1.0
23 May 2017



Joint Research Centre

Contact information
Email: erca@jrc.ec.europa.eu

JRC Science Hub
https://ec.europa.eu/jrc

JRC106943

Luxembourg (Luxembourg): Publications Office of the European Union, 2017.

How to cite this report: Mateboer K., Bakker D., Sportiello L.; Smart Tachograph: User manual for the sample cryptographic keys and digital certificates Generation Tool, EUR, doi

Contents

Abstract
In order to aid manufacturers, component personalisers, certification authorities and other Digital Tachograph stakeholders with the development and testing of equipment and systems complying with the Generation-2 Smart Tachograph specifications, a tool has been developed that can be used to generate sample cryptographic keys and digital certificates. Stakeholders may use this tool to generate keys and certificates with specific properties for their testing purposes.

This document is the user manual for the tool. It explains

- how to download the tool, build the tool from the provided source files and execute it. Note that building the tool is only necessary in case a user decides to make changes to the source files.
- which functions are supported by the tool and what the exact syntax of the respective commands is.
- how to use the tool to generate symmetric and asymmetric keys and certificates that comply with the specifications in Appendix 11 of Annex 1C. All types of keys and certificates specified in this Appendix can be generated by the tool.
- how to use the tool to sign certificates that do not comply with Appendix 11. Such certificates may be useful for 'unhappy flow' testing, i.e. testing that equipment is resilient against unexpected or wrong inputs.

1 Introduction

The digital tachograph system (Generation-1 Digital Tachograph) has been introduced by Council Regulation 3821/85 as amended by Council Regulation 2135/98 [1] and Commission Regulation 1360/2002 [2]. Regulation (EU) No 165/2014 [3] and its Commission Implementing Regulation (EU) 2016/799 [4] call for the introduction of a Generation-2 Smart Tachograph.

Annex 1C (Requirements for construction, testing, installation, and inspection) to [4] includes the technical specifications of the Smart Tachograph system. Detailed technical information is contained in a series of sixteen appendices to Annex 1C. Appendix 11 (Common Security Mechanisms) describes all details of the cryptographic security mechanism used to protect the data stored and transmitted in the Smart Tachograph system.

1.1 Scope of this document

Security mechanisms have been defined to secure the exchange and storage of data by the Smart Tachograph equipment, namely Vehicle Units, Tachograph Cards, Motion Sensors and External GNSS Facilities. Such mechanisms are mainly based on cryptographic solutions. Specifically, a cryptographic infrastructure has been defined, with symmetric keys, asymmetric keys and digital certificates stored in the Smart Tachograph equipment, allowing the execution of cryptographic algorithms and protocols. Section 2.1 below gives an overview of the cryptographic infrastructure of the Smart Tachograph. In order to support Member State Certification Authorities (MSCAs), manufacturers, component personalisers and other stakeholders with the development and testing of equipment and systems complying with these new specifications, a comprehensive set of Generation-2 sample keys and certificates has been developed. Ref. [9] describes the contents of this sample set.

However, stakeholders may need or wish to use other sample keys and certificates than those provided in the sample set. Reasons for this may be, for example:

- test proper functioning of asymmetric key pairs generated by the stakeholder
- use symmetric keys derived from a stakeholder-generated master key
- use stakeholder-specific data (e.g. serial numbers, nation code, manufacturer code) in equipment certificates
- use specific validity periods for certificates
- use certificates that do not comply with some requirement(s) in the specifications. This may be useful to perform ‘unhappy flow’ testing of equipment’s resilience against erroneous input.

In order to allow stakeholders to generate their own keys and certificates, a sample keys and certificates Generation Tool has been developed. This document is the user manual of this tool.

Please note that most of the Generation-2 equipment must also contain Generation-1 cryptographic keys and digital certificates, in order to be able to communicate to Generation-1 equipment. However, for these keys and certificates stakeholders may use cryptographic test material developed or made available in the past. The key generation tool described in this document can only be used for generating Generation-2 keys and certificates.

1.2 Intended audience

This document is intended for stakeholders involved in the development of equipment and systems for the Smart Tachograph system. Readers of this document should be familiar with the contents of Annex 1C, and especially with Appendix 11 to that Annex.

1.3 Disclaimer

The sample keys and certificates Generation Tool has to be only seen as a support to the development and testing processes of digital tachograph stakeholders. In the event of any conflict between the output of the tool and the Implementing Regulation 2016/799 [4] and its Annexes and Appendices, the latter shall prevail. Each stakeholder remains fully responsible for making sure that their equipment complies with all requirements in [4].

1.4 Document structure

This document is structured as follows:

- Chapter 2 gives an overview of the Smart Tachograph system and its security mechanism. This serves as background information. Readers familiar with the Smart Tachograph system may skip this chapter.
- Chapter 3 explains how to download, build and execute the tool.
- Chapter 4 lists the functions supported by the tool and gives a detailed description of each respective command, including its purpose, syntax and input and output. It also lists all possible errors and warnings and gives some usage examples for each command.
- Chapter 5 describes how to use the various functions of the tool in conjunction in order to create each of the keys and certificates that play a role in the Smart Tachograph ecosystem.
- Chapter 6 contains some questions and answers related to problems that may be encountered during execution of the tool.

2 Background: Smart Tachograph security mechanisms

Figure 1 shows an overview of the Smart Tachograph system.

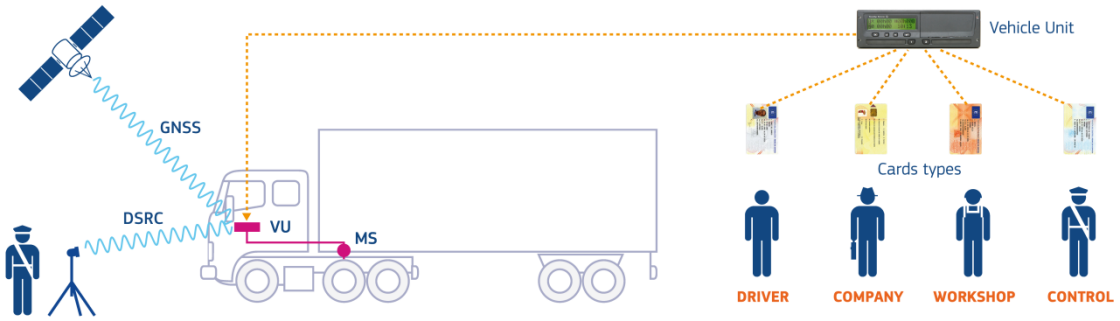


Figure 1 Overview of the Smart Tachograph system

The vehicle unit (VU) is the central component of the system. Every vehicle is equipped with a vehicle unit whose main task is to log driving activities.

In order to do so, each vehicle unit is securely paired to a motion sensor (MS), which is connected to the gearbox of the vehicle and provides the vehicle unit with a signal that represents the vehicle's speed. During pairing, which is done once by a workshop, the VU and the motion sensor mutually authenticate each other. After this, part of the communication between a VU and a motion sensor is authenticated and encrypted.

Users of a vehicle unit are equipped with a tachograph card (TC). There are four types of cards, corresponding to the four roles of Driver, Control Officer, Workshop or Company. Upon insertion of a card in a VU, the card and the VU mutually authenticate each other. Subsequently, communication between them is authenticated and in some cases encrypted. Moreover, once the VU is authenticated to the card, it is allowed to write data to some of the data structures on the card. In addition, the exact working mode of a VU is determined by the card(s) that is/are inserted into its card slots.

The vehicle unit is also able to periodically log the vehicle's position and check the plausibility of the motion sensor's signal by means of position determination based on a Global Navigation Satellite System (GNSS). The GNSS receiver that is necessary to do so is either contained in the VU itself, or in an External GNSS Facility (EGF). In case an EGF is used, the vehicle is securely coupled once to the VU by a workshop. During coupling, the VU and EGF authenticate each other. Afterwards, communication between them is authenticated.

The data stored on either a vehicle unit or a card must be downloaded by a control officer during a check or by the responsible company. Although the communication interfaces used for downloading data from a card or from a VU are rather different, the security requirements on both interfaces are equal: the integrity, authenticity and non-repudiation of the downloaded data must be protected by means of a digital signature created by the VU or the card.

Finally, the Smart Tachograph system also contains Remote Early Detection Communication Readers (REDCRs). These are readers that are operated by a control officer and are positioned along the roadside or on officers' vehicles. They are capable of interrogating a Remote Communication Facility (RCF) in a passing vehicle over a DSRC link, without having to stop the vehicle. The VU periodically stores relevant data, in particular driving and control data, in the RCF. Such a system allows for a frequent verification of the vehicle status through remote interrogations. The data communicated over the DSRC link is encrypted and authenticated by the VU before it is sent to the RCF.

2.1 Smart Tachograph cryptographic infrastructure

In order to fulfil the security requirements for each of the interactions outlined above, the vehicle unit, tachograph cards, motion sensor and EGF all contain a number of cryptographic elements, namely public/private key pairs, digital certificates and/or symmetric keys. They comply with Appendix 11 of Annex 1C, which also gives a full specification of the protocols adopted to secure the interaction of the different system components.

Public/private key pairs are based on Elliptic Curve Cryptography (ECC), symmetric keys are based on the AES algorithm, whereas as hash algorithm SHA-2 has been adopted.

A number of pre-defined key lengths and hash sizes have been specified and combined together to form cipher suites, which assure a consistent level of security for all interactions of the Smart Tachograph components. The cipher suites are summarized in Table 1. All system components mentioned above must support all cipher suites.

Cipher suite	ECC key size (bits)	AES key length (bits)	Hashing algorithm
CS#1	256	128	SHA-256
CS#2	384	192	SHA-384
CS#3	512/521	256	SHA-512

Table 1 Cipher suites defined for the Smart Tachograph system

For Elliptic Curve Cryptography there is the need to choose domain parameters. Appendix 11 of Annex 1C allows two sets of standardized domain parameters, the NIST and Brainpool domain parameters. Both for the NIST and the Brainpool standard a set of domain parameters for each of the key sizes specified in Table 1 has been selected. The complete set is shown in Table 2.

Name	Key size (bits)
NIST P-256	256
BrainpoolP256r1	256
NIST P-384	384
BrainpoolP384r1	384
BrainpoolP512r1	512
NIST P-521	521

Table 2 Allowed standardized domain parameters for ECC

A cryptographic infrastructure has been designed for the generation and deployment of the cryptographic elements in the Smart Tachograph system. It is made of three layers, a European level, a member state level and a system component level. Such an infrastructure acts both as Public Key Infrastructure (PKI) and as symmetric keys distribution infrastructure.

2.1.1 Asymmetric keys and Public Key Infrastructure

The ECC key pairs are part of a Public Key Infrastructure (PKI), as shown in Figure 2. This Public Key Infrastructure (PKI) consists of three levels. From top to bottom, these are:

- the European level, managed by the European Root Certificate Authority (ERCA).
- the Member State level, managed by the Member State Certificate Authority (MSCA) of every member state involved in the digital tachograph system.
- The equipment level, managed by the manufacturers or personalizers of vehicle units, tachograph cards and EGFs.

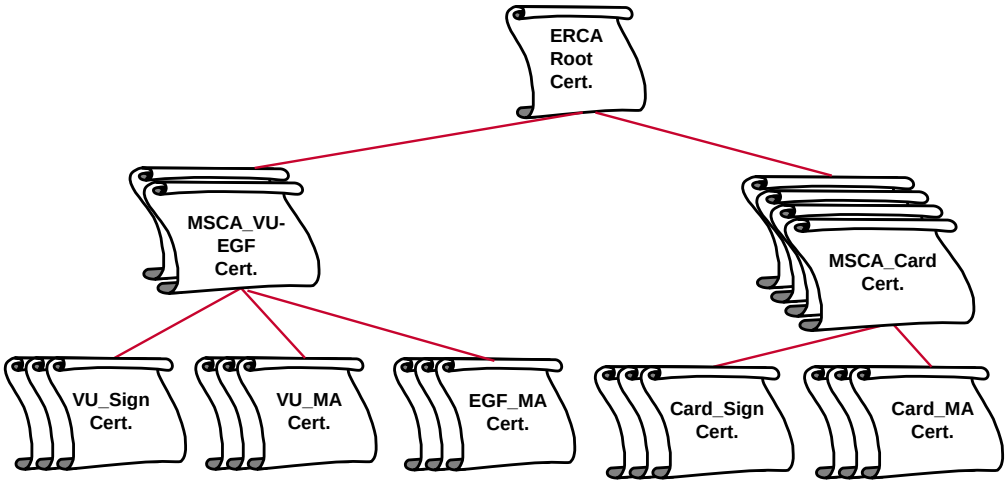


Figure 2 Smart Tachograph PKI

On the European level, the ERCA creates a single ECC key pair that serves as the root key pair of the entire PKI. The ERCA also creates a self-signed root certificate containing the root public key. ERCA certificates have a validity period of 34 years and 3 months. The ERCA root key pair and certificate are renewed over time (see next section). The ERCA uses the corresponding private key to sign MSCA certificates. The public MSCA keys to be signed are sent to the ERCA by the MSCAs.

On the Member State level, each MSCA that needs to issue certificates for VUs or EGFs creates a key pair, which is indicated in Appendix 11 as MSCA_VU-EGF. Then it requests the ERCA to sign a certificate for the respective public key. Similarly, each MSCA that needs to issue certificates for tachograph cards creates an MSCA_Card key pair and asks the ERCA to sign the corresponding certificate. MSCA_VU-EGF certificates are valid for 17 years and 3 months. MSCA_Card certificates have a validity period of 7 years and 1 month. The MSCA uses the corresponding private keys to sign equipment certificates.

On the equipment level, the manufacturer or personaliser of each VU, card or EGF[1] creates at least an equipment key pair for mutual authentication. The component will use this key pair to authenticate itself to other equipment it will interoperate with during its lifetime. In addition, for a VU, a workshop card or a driver card, the manufacturer or personaliser also creates a key pair for signing. These components will use their signing private key to sign data that is downloaded from them. Other component types are not required to support data downloads and hence do not need a signing key pair. All generated public keys are sent to the competent MSCA for the generation of the respective certificate. The validity period of equipment certificates is as follows:

- VU_Sign 15 years and 3 months
- VU_MA 15 years and 3 months
- Driver Card_Sign: 5 years and 1 month
- Workshop Card_Sign: 1 year and 1 month
- Driver Card_MA: 5 years
- Company Card_MA: 5 years
- Control Card_MA: 2 years
- Workshop Card_MA: 1 year
- EGF_MA 15 years

[1] Note that motion sensors do not contain asymmetric keys or certificates.

All certificates issued within the Smart Tachograph system are so-called card-verifiable certificates. Their format is equal for all types of certificates and is show in the table below:

Field	Tag	Length (bytes)	ASN.1 data type (see Appendix 1 of Regulation (EU) 2016/799 [4])
ECC Certificate	'7F 21'	var	
ECC Certificate Body	'7F 4E'	var	
Certificate Profile Identifier	'5F 29'	'01'	INTEGER(0..255)
Certificate Authority Reference	'42'	'08'	KeyIdentifier
Certificate Holder Authorisation	'5F 4C'	'07'	CertificateHolder Authorisation
Public Key	'7F 49'	var	
Domain Parameters	'06'	var	OBJECT IDENTIFIER
Public Point	'86'	var	OCTET STRING
Certificate Holder Reference	'5F 20'	'08'	KeyIdentifier
Certificate Effective Date	'5F 25'	'04'	TimeReal
Certificate Expiration Date	'5F 24'	'04'	TimeReal
ECC Certificate Signature	'5F 37'	var	OCTET STRING

Table 3 Smart Tachograph certificate format

2.1.2 Symmetric keys

Concerning symmetric keys in the Smart Tachograph system, there are two kinds of keys managed in the cryptographic infrastructure:

- Keys for protecting the communication between a VU and a Motion Sensor:
  - Motion Sensor Master Keys. These are constituted from a Workshop Card Master Key part and a VU Master Key part.
  - Identification Keys, which are derived from the Motion Sensor Master Keys and a constant vector.
- Keys for protecting the communication over a DSRC link between a VU and a Remote Early Detection Communication Reader:
  - DSRC Master Keys
  - VU-specific DSRC keys for encryption and authentication, derived from a DSRC Master Key

The motion sensor keys are used to secure the link between VU and Motion Sensor. The DSRC keys are used to secure the data uploaded on the Remote Communication Facility.
The Motion Sensor Master Key, along with its constituting parts, and the DSRC Master Key are generated by the ERCA. Every time the ERCA replaces the ERCA root key pair, it also replaces the Master Keys (see next section). The Master Keys are provided by the ERCA to the MSCAs. A MSCA can use the Motion Sensor Master Keys to generate specific cryptographic material to be installed in a Motion Sensor. A MSCA can also provide the Workshop Card Master Key part to be installed in Workshop cards, or can provide the VU Master Key part to be installed in VUs according to Appendix 11 of Annex 1C. A MSCA can use the DSRC Master Keys to generate the VU-specific DSRC keys for a VU. A MSCA can also provide the DSRC Master Keys to be installed in Control and Workshop card according to Appendix 11 of Annex 1C. Each generation of these Master Keys will be in use by Smart Tachograph component for 34 years.

2.2 ERCA keys replacement and link certificates

Appendix 11 of Annex 1C describes a mechanism that ERCA can use to replace the root key pair and respective certificate, along with the associated master keys. The appendix also specifies that such a replacement shall take place every 17 years.

An important advantage of this key replacement is that it gives ERCA a chance to review whether the security level of the root key pair and associated master keys is still sufficient. If this is not the case, ERCA can decide to switch to longer key lengths.
Whenever ERCA creates a new root key pair, the following actions take place:

- In order to ensure interaction between equipment issued under different generations of the root key, ERCA will issue a so-called link certificate. A link certificate contains the new ERCA public key and is signed with the previous ERCA private key. By verifying the link certificate, equipment issued under the previous ERCA key pair is able to trust the new ERCA public key and so it is allowed to interact with equipment issued under the new ERCA public key. See Appendix 11 of Annex 1C for more details.
- ERCA also generates a new Motion Sensor Master Key, and relative parts, and a new DSRC Master Key. For the concerned equipment all master keys, and related materials, that are valid at the time of its issuance have to be installed in its memory. This ensures the interaction between equipment linked to different master key generation. See Appendix 11 of Annex 1C for more details.

3 Downloading, building and executing the tool

The sample keys and certificates Generation Tool is a Java command line application. Please note that the following conditions have to be satisfied for its execution:

- The tool requires Java 8 [10] with Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy Files installed [11]**Error! Reference source not found.**; see below for instructions on how to do this.
- The tool depends on the Bouncy Castle JCE provider [12]. The required bcprov-jdk15on-1.56.jar needs to be in the same folder as the executable jar.

3.1 Downloading

Downloading and unzipping the tool results in a folder containing a readme.txt file, a 'src' folder and a 'installed' folder:

- The readme.txt file contains the information in this chapter and the next ones in a condensed format.
- The 'installed' folder contains a compiled version of the tool (file 'tachograph-keytool-1.0.0.jar') that can be directly used.
- The 'src' folder contains all source files of the tool. This allows stakeholders to adapt the tool to their wishes, should this be necessary.

3.2 Building

This section briefly describes the prerequisites for building the tool, if necessary.

The 'sample keys and certificates generation tool' project uses Maven to manage the build process. In order to build:

- Maven has to be installed [13].
- JDK 8 has to be installed [10].
- Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy Files have to be installed [11]. These can be downloaded from http://www.oracle.com/technetwork/java/javase/downloads/jce8-download-2133166.html. Unzip the downloaded zip and copy the files 'local_policy.jar' and 'US_export_policy.jar' to the $JAVA_HOME/jre/lib/security folder. Note: these jars will be already there so they should be overwritten.

Run the following command in the unzipped folder to build the tool:

```
mvn clean install
```

The output of the build is an executable .jar file, similar to the .jar file provided in the 'installed' folder. After building, the 'target' folder contains all results of the build process. This includes the executable .jar file. Also the Bouncy Castle bcprov-jdk15on-1.56.jar is present in the 'target' folder at the end of the building process.

Note that building the tool has been tested on the Windows and Linux platforms.

3.3 Executing

The sample keys and certificates Generation Tool is a Java command line application. To use the tool on Windows machines, open a command window in the folder in which the executable jar file is located, e.g. by pressing SHIFT + right mouse button and selecting 'open command window here'. Use the following command to run the tool (the version number may be different):

```
java -jar tachograph-keytool-1.0.0.jar
```

A batch file st.bat (st for ‘smart tachograph’) is added in the ‘installed’ folder to simplify command line usage. Use the following commands to show details on the supported tool functions, as described in chapter 4:

- st generate ec
- st generate aes
- st create ca
- st create equipment
- st create request
- st sign
- st link
- st verify
- st derive dsrc
- st derive msmk
- st derive msik
- st encrypt ms
- st encrypt pk

Note that the tool accepts any command or parameter truncation as long as one of the options starts with the specified truncation. So, assuming the st.bat file is used, the following command lines are equivalent (refer to section 4.2.1):

- st generate ec erca1 brainpoolp256r1
- st gen ec erca1 brainpoolp256r1
- st g e erca1 brainpoolp256r1

4 Supported functions

4.1 Overview

4.1.1 General syntax description

The tool supports the functions expressed by the following commands:

- generate ec — to generate a ECC key pair.
- generate aes — to generate an AES key.
- create ca — to create a (self-signed) ERCA or MSCA certificate for a previously generated key pair.
- create equipment — to create a (self-signed) equipment certificate for a previously generated key pair
- create request — to create a (self-signed) VU certificate for a previously generated key pair, where a certificate request number is used instead of a VU serial number.
- sign — to sign a previously created certificate, using a previously created CA private key and corresponding certificate.
- link — to create and sign the Link certificate between two previously generated ERCA root certificates.
- verify — to verify a certificate.
- derive dsrc — to derive a VU-specific DSRC key from a DSRC master key
- derive msmk — to derive a motion sensor master key from a motion sensor master key–VU part and a motion sensor master key–WC part.
- derive msik — to derive a motion sensor identification key from a motion sensor master key.
- encrypt ms — to encrypt a motion sensor serial number with an identification key.
- encrypt pk — to encrypt a pairing key with a motion sensor master key.

Section 4.2 describes these commands in more detail.

Commands can have required and/or optional parameters. In the descriptions in section 4.2, required parameters are put between triangular brackets: <parameter>. Optional parameters are furthermore enclosed between square brackets: [<parameter>]. The brackets are not to be actually used in the commands.

Note that names of **input** files must be given including their extension. For **output** files, the tool will add the correct extension automatically; this should not be included in the parameter. For both types of files, either the full path or a path relative to the location of the executable jar can be specified.

4.1.2 Input parameter validations

The tool will perform general checks on all inputs, which may result in one or more errors or warnings. In case of a warning, the requested output is generated, but the user is alerted to some unusual or non-compliant property of that output. In case of an error, no output could be generated.

General errors:

- If a list of possible values is specified for a given parameter, but the user inputs a value that is not in the list.
- If illegal characters are used for an output file name. (Note that if an extension is specified, no error or warning will be given, but a file having a ‘double extension’ will be created.)

- For an input file name: If there is no file with that name in the current folder, or if the extension is omitted.
- If an incorrect format (e.g. numeric / alphanumeric, wrong length, …) is used for one of the input parameters. Numeric and alphanumeric input formats are specified in section 4.1.3. Parameter lengths (if applicable) are specified for each function in section 4.2.
- If a required parameter is missing or too many parameters are provided.

Apart from these general checks, the tool may perform specific validations based on the input parameter in question. These are described per command in section 4.2.

4.1.3 Numeric and alphanumeric input formats

Numerical input values can be provided both in hexadecimal and decimal format. If the input value is shorter than the expected length, it will be left-padded with '00' bytes. In case the input is too long, only the right-most digits will be kept, and a warning will be given in case significant bits are lost.

- 0xff or 0xFF or 255 are equal for a 1-byte numeric field.
- 0x0001 or 0x01 or 1 are equal for a 2-byte numeric field. 0x020001 will result in the same output, but a warning will be given.

Alphanumerical input values must be provided in alphanumerical format. If the input value is shorter than the expected length, it will be right-padded with spaces. In case the input is too long, only the left-most digits will be kept, and a warning will be given in case digits are lost. Double quotes ("") may optionally be used.

- "EC" or "EC " or EC are all equally valid for a 3-byte alphanumeric field.
- " EC" (starting with two spaces) will result in " E", and a warning will be given.

4.2 Detailed function descriptions

4.2.1 Generate ec

This command is used to generate an ECC key pair. The private key and the public key are both stored in a PKCS#8 file. For the format of this file, refer to Appendix 2.

Syntax:
generate ec <name> <curve>

name: the name of the PKCS#8 file to be created.
curve: the standard name of the elliptic curve parameters to be used, one of:

- secp256r1
- secp384r1
- secp521r1
- brainpoolp256r1
- brainpoolp384r1
- brainpoolp512r1

Example: generate ec MSCA_VU-EGF(1) brainpoolp256r1
Output: A file MSCA_VU-EGF(1).pkcs8 containing the key pair, structured as described in Appendix 2.

Errors:

- See section 4.1.2.

Warnings:

- None

4.2.2 Generate aes

This command is used to generate a AES symmetric key, and store it in a binary file.

Syntax:
generate aes <name><size>

name: the name of the file to be created
size: the size in bits of the secret key to be generated (128, 192, or 256)

Example: generate aes keys/DSRCMK-2 192
Output: A file DSRCMK-2.bin containing the 192-bits AES key in plain text, stored in a folder called 'keys', which is located in the folder where the executable jar is.

Errors:

- See section 4.1.2.

Warnings:

- None

4.2.3 Create ca

This command is used to create a self-signed ERCA root or MSCA certificate, using an existing ECC key pair stored in a PKCS#8 file. The certificate is stored in a file with a .cert extension. The certificate format complies to Appendix 11; see also Table 3 above.

Note: In case this command is used to create an ERCA root certificate, the certificate is ready for use immediately. In case it is used to create a MSCA certificate, the resulting certificate must still be signed with the proper ERCA certificate using the 'sign' command, see section 4.2.6.

Syntax:
create ca <type> <name> <keyname> <nationnumeric> <nationalpha> <serialnumber> <additionalinfo> <caidentifier> [<effectivedate> [<expirationdate>]]

type:	one of the following options: ○ erca ○ msca_vu_egf ○ msca_card
name:	the name of the certificate file to be created
keyname:	the name of the PKCS#8 file containing the key to be certified, including the .pkcs8 file extension.
nationnumeric:	a numerical identifier of the nation; see Appendix 1 to Annex 1C [4]. If the selected type is erca, any other value than 'FD' will result in a warning.
nationalpha:	a country code of up to three characters; see Appendix 1 to Annex 1C [4]. In case less than three characters are input, the tool adds spaces to the right. If the selected type is erca, any value other than "EC" (or "EC ") will result in a warning.
serialnumber:	1-byte key serial number; see Appendix 1 to Annex 1C [4]
additionalinfo:	2-byte CA-specific additional coding; see Appendix 1 to Annex 1C [4].
caidentifier:	the CA identifier; see Appendix 1 to Annex 1C [4]. Any value other than '01' will result in a warning
effectivedate:	the certificate effective date and time in ISO 8601 format (e.g. 2018-01-01T12:00:00). If no effective date is specified by the user, the current system time will be used.
expirationdate:	the certificate expiration date and time in ISO 8601 format. If no expiration date is specified by the user, it will be based on the effectivedate and the validity period specified in Appendix 11 of Annex 1C [4]. Note that it is not possible to specify the expiration date without specifying the effective date as well.

Notes:

- For creating ERCA link certificates, use the link command specified in section 4.2.7.
- For some data elements in the certificate, the tool will automatically determine the value:
  - The value of the certificate profile identifier will be set to '00'.
  - The value of the Certificate Authority Reference (CAR) will be chosen equal to those for the Certificate Holder Reference; see below. (For MSCA certificates, the CAR will subsequently be replaced in a 'sign' command, see section 4.2.6.)
  - The value for the Certificate Holder Authorisation (CHA) will be chosen based on specified 'type' parameter.
  - The value for the domain parameters OID will be taken from the specified .pkcs8 file.
  - The value of the public point will be taken from the specified .pkcs8 file.
- The signature over the certificate will generated using the private key in the specified .pkcs8 file.

  Please refer to chapter 5.3 to learn how to create certificates using other (non-compliant) values for these data elements.

Example 1: create ca erca /src/test/keys/erca(1) erca(1).pkcs8 0xFD EC 23 0 01

Output: A file erca(1).cert containing the requested certificate. Note that the value used for the serial number will be '0x17' (23 decimal). The file is stored in a folder /src/test/keys/ relative to the root.

Example 2: create ca msca_card mscacard_1 mscacard_1.pkcs8 0x0d "D" 0x0f 0xFFFF 01 2018-01-01T12:00:00
Output: A file mscacard_1.cert containing the requested certificate. The file is stored in the current folder.

Errors:

- See section 4.1.2.
- If a non-existent date (e.g. February 30th) is specified in the effectivedate or expirationdate parameters.
- If the indicated .pkcs8 file is not a proper PKCS#8 data structure complying with Appendix 2.

Warnings:

- In case (based on the specified effective date and/or expiration date) the resulting certificate is not yet valid or is expired already.
- In case both the effective date and the expiration date are specified by the user, and the resulting certificate validity period is not compliant with Appendix 11.
- In case for one or more input parameters a value is used that is not compliant to the specifications; see above.

4.2.4 Create equipment

This command is used to create a self-signed equipment certificate, using an existing ECC key pair stored in a PKCS#8 file. The certificate is stored in a file with a .cert extension. The certificate format complies to Appendix 11; see also Table 3 above.

Note: The resulting certificate must still be signed with the proper MSCA certificate using the 'sign' command, see section 4.2.6.

Syntax:
Create equipment <type> <name> <keyname> <serialnumber> <month><year><manufacturercode> [<effectivedate> [<expirationdate>]]

type:	one of the following options: ○ driver_card_ma ○ driver_card_sign ○ workshop_card_ma ○ workshop_card_sign ○ control_card_ma ○ company_card_ma ○ vu_ma ○ vu_sign ○ egf_ma
name:	the name of the certificate file to be created.
keyname:	the name of the PKCS#8 file containing the key to be certified.
serialnumber:	the 4-byte equipment serial number; see Appendix 1 to Annex 1C [4].

month:	the month of manufacturing of the VU. Format: 1 byte BCD. A warning will be given if the number entered is higher than 12 (decimal).
year:	the year of manufacturing of the VU. Format: 1 byte BCD. If more than 2 digits are entered, the tool will only use the last two; e.g. 2017 becomes 17.
manufacturercode:	the 1-byte manufacturer code; see Appendix 1 to Annex 1C [4].
effectivedate:	the certificate effective date and time in ISO 8601 format (e.g. 2018-01-01T12:00:00). If no effective date is specified by the user, the current system time will be used.
expirationdate:	the certificate expiration date and time in ISO 8601 format. If no expiration date is specified by the user, it will be based on the effectivedate and the validity period specified in Appendix 11 of Annex 1C [4]. Note that it is not possible to specify the expiration date without specifying the effective date as well.

Notes:

- For creating VU certificates based on a request serial number, please see section 4.2.5.
- For some data elements in the certificate, the tool will automatically determine the value:
  - The value of the certificate profile identifier will be set to '00'.
  - The value of the Certificate Authority Reference (CAR) will be chosen equal to the Certificate Holder Reference; see below. (The CAR will subsequently be replaced in a 'sign' command, see section 4.2.6.)
  - The value for the Certificate Holder Authorisation (CHA) will be based on the specified 'type' parameter.
  - The value for the Domain Parameters OID will be taken from the specified .pkcs8 file.
  - The value of the public point will be taken from the specified .pkcs8 file.
  - For the Card Holder Reference (CHR) field:
    - The value of the monthYear data element will be will be based on the specified 'month' and 'year' parameters.
    - The value of the type data element will be based on the specified 'type' parameter.
- The signature over the certificate will generated using the private key in the specified .pkcs8 file.

  Please refer to chapter 5.3 to learn how to create certificates using other (non-compliant) values for these data elements.

Example: create equipment driver_card_ma drivercard_1 driver_card_1.pkcs8 0x02000131 4 17 0x41 2008-01-01T12:00:00 2019-01-01T12:00:00
Output: A file driver_card_1.cert containing the specified certificate, with an effective date in the past and an incorrect validity period.

Errors and Warnings: See for 'create ca' command in section 4.2.3.

4.2.5 Create request

This command is identical in syntax to the 'create equipment' command described in the previous section. However, it will result in a VU certificate that does not contain a VU serial number in the CHR, but a request serial number instead. For more information, please see Appendix 11 to Annex 1C [4], requirement CSM_154 and Appendix 2 to Annex 1C, data type `CertificateRequestID`.

Syntax:
Create request <type> <name> <keyname> <serialnumber> <month><year><manufacturercode> [<effectivedate> [<expirationdate>]]

type:	one of the following options: ○ vu_ma ○ vu_sign
name:	the name of the certificate file to be created.
keyname:	the name of the PKCS#8 file containing the key to be certified.
serialnumber:	the 4-byte request serial number; see Appendix 1 to Annex 1C [4].
month:	the month of manufacturing of the VU. Format: 1 byte BCD. A warning will be given if the number entered is higher than 12 (decimal).
year:	the year of manufacturing of the VU. Format: 1 byte BCD. If more than 2 digits are entered, the tool will only use the last two; e.g. 2017 becomes 17.manufacturercode: the 1-byte manufacturer code; see Appendix 1 to Annex 1C [4].
effectivedate:	the certificate effective date and time in ISO 8601 format (e.g. 2018-01-01T12:00:00). If no effective date is specified by the user, the current system time will be used.
expirationdate:	the certificate expiration date and time in ISO 8601 format. If no expiration date is specified by the user, it will be based on the effectivedate and the validity period specified in Appendix 11 of Annex 1C [4]. Note that it is not possible to specify the expiration date without specifying the effective date as well.

Notes: See for 'create equipment' command in section **Error! Reference source not found.**.

Example: create request vu_sign vu_02 vu.pkcs8 0x00000001 4 2017 0x41 2018-03-31T00:00:00
Output: A file vu_02.cert containing the specified certificate, which is based on a certificate serial number rather than a VU serial number.

Errors and Warnings: See for 'create ca' command in section 4.2.3.

4.2.6 Sign

This command is used to sign an existing (self-signed) certificate with an existing private key of a Certificate Authority (CA). The private key must be stored in a .pkcs8 file, and a .cert file containing the certificate corresponding to this private key must also be present.

The command does the following:

- Replace the signature of the self-signed certificate by a signature created with the indicated CA private key.
- Replace the Certificate Authority Reference field of the self-signed certificate such that it properly references the CA certificate.

Syntax:
sign <selfsignedcertificate> <caprivatekey> <cacertificate> <name>

selfsignedcertificate:	the name of the file containing the self-signed certificate to be signed.
caprivatekey:	the name of the .pkcs8 file containing the CA private key.
cacertificate:	the name of the .cert file containing the CA certificate.
name:	the name of the certificate file to be created.

Example: sign drivercard_1.cert mscacard_1.pkcs8 mscacard_1.cert drivercard_1-final

Output: A file drivercard_1-final.cert containing a certificate based on the existing drivercard_1 certificate, but properly signed with the mscacard_1 private key and referring in the CAR field to the mscacard_1 certificate.

Errors:

- See section 4.1.2.
- If the indicated .pkcs8 file is not a proper PKCS#8 data structure complying with Appendix 2 and containing a valid private key.
- If one of the indicated .cert files does not comply with the certificate format in Appendix 11 of Annex 1C.

Warnings:

- If the effective date of a signed certificate is before that of the signing certificate.
- If the expiration date of a signed certificate is after that of the signing certificate.
- If the signing certificate authorisation does not match with the signed certificate. For example, if an equipment certificate is signed by an ERCA root certificate.
- If the specified caprivatekey and the public key in the specified cacertificate do not form a valid key pair.
- If the signature over the self-signed certificate is not correct.

4.2.7 Link

This command is used to create and sign an ERCA link certificate between two existing ERCA root certificates. Note that according to Appendix 11 of Annex 1C the effective date of the second ERCA certificate should be exactly 17 years after the effective date of the first ERCA certificate.

Syntax:

link <privatekey> <currentcertificate> <nextcertificate> <name>

privatekey:	the name of the .pkcs8 file containing the current ERCA private key
currentcertificate:	the name of the .cert file containing the current ERCA certificate
nextcertificate:	the name of the .cert file containing the next ERCA certificate
name:	the file name of the link certificate to be created.

Notes:

- The tool will automatically determine the value of all data elements in the certificate, possibly based on the specified input certificates:
  - The value of the Certificate Profile Identifier is set to ‘00’.
  - The Certificate Authority Reference field is copied from the CHR field of the .cert file indicated in the currentcertificate parameter.
  - The Certificate Holder Authorisation field is identical to the CHA field of both input certificates.
  - Domain Parameters and Public Point are copied from the .cert file indicated in the nextcertificate parameter.
  - The Certificate Effective Date is chosen equal to the CEfD of the next certificate.
  - The Certificate Expiry Date is chosen equal to the CExD of the current certificate.

Example: link erca(1).pkcs8 erca(1).cert erca(2).cert erca(1)-erca(2)

Output: A file erca(1)-erca(2).cert containing the link certificate between the indicated ERCA root certificates.

Errors:

- See section 4.1.2.
- If the indicated .pkcs8 file is not a proper PKCS#8 data structure complying with Appendix 2 and containing a valid private key.
- If one of the indicated .cert files does not comply with the certificate format in Appendix 11 of Annex 1C.

Warning:

- If the effective date of the next ERCA certificate is less than 17 years after the effective date of the current ERCA certificate.

4.2.8 Verify

This command is used to verify the signature over a previously generated certificate. Certificate verification is possible only if the certificate format conforms to Appendix 11, such that it can be parsed.

Syntax:

verify <certificate> <cacertificate>

certificate:	the name of the file containing the certificate to be verified
cacertificate:	the name of the file containing the CA certificate needed to verify the certificate.

Example: verify drivercard1.cert mscacard1.cert

Output:

- If verification was successful: a command line message “verified: ” plus the name of the verified certificate.
- If verification was not successful: a command line message “Certificate signature is invalid”

Errors:

- See section 4.1.2.

Warnings:

- If the CAR field in the certificate to be verified does not equal the CHR field in the CA certificate. (This will normally result in an unsuccessful verification.)
- If the validity period of the certificate to be verified precedes or exceeds the CA certificate validity period.
- If the CA certificate authorisation does not match with the certificate to be verified. For example, if an equipment certificate is verified using an ERCA root certificate. (This will normally result in an unsuccessful verification.)

4.2.9 Derive dsrc

This command is used to derive vehicle unit-specific DSRC encryption and authentication keys from a DSRC Master key, as specified in section 9.2.2.1 of Appendix 11 to Annex 1C [4].

Syntax: There are two possible syntaxes for this command, one using separate values for the serial number, month, year and manufacturer code, the other using a single value for the extended serial

number. If the first syntax is used, the tool concatenates these inputs together with the value for the VU equipment type (‘06’) into the VU extended serial number. If the second syntax is used, the tool uses the extended serial number as specified.

Syntax 1:

derive dsrc <name> <dsrcmk> <serialnumber> <month> <year> <manufacturercode>

name:	the base name for the output files
dsrcmk:	the name of the file containing the DSRC master key
serialnumber:	the 4-byte equipment serial number; see Appendix 1 to Annex 1C [4].
month:	the month of manufacturing of the VU. Format: 1 byte BCD. A warning will be given if the number entered is higher than 12 (decimal).
year:	the year of manufacturing of the VU. Format: 1 byte BCD. If more than 2 digits are entered, the tool will only use the last two; e.g. 2017 becomes 17.
manufacturercode:	the 1-byte manufacturer code; see Appendix 1 to Annex 1C [4].

Errors:

- See section 4.1.2.

Warnings:

- See note above for ‘month’.

Syntax 2:

derive dsrc <name> <dsrcmk> <extendedserialnumber>

name:	the base name for the output files
dsrcmk:	the name of the file containing the DSRC master key
extendedserialnumber:	the 16-byte equipment extended serial number; see Appendix 1 to Annex 1C [4]. Format is fixed-size 16 digits (hexadecimal).

Errors:

- See section 4.1.2.

Example 1: derive dsrc dsrcVU_01 dsrcmk_1.bin 0x00000001 1 18 0x23
Output:

- The resulting encryption key, stored in a file named <name>-enc.bin
- The resulting authentication key, stored in a file named <name>-mac.bin

Example 2: derive dsrc dsrcVU_01 dsrcmk_1.bin 0000000101120623
Output:

- The resulting encryption key, stored in a file named <name>-enc.bin
- The resulting authentication key, stored in a file named <name>-mac.bin

Note that the resulting keys will have the same value in both examples.

4.2.10 Derive msmk

This command is used to derive a motion sensor master key by XOR-ing the Motion Sensor Master Key – VU part ($K_{M\text{-}VU}$) and the Motion Sensor Master Key – Workshop part ($K_{M\text{-}WC}$), as described in section 9.2.1.1 of Appendix 11 to Annex 1C [4].

Syntax:
derive msmk <name> <msmk_vu> <msmk_wc>

name:	the base name for the output file
msmk_vu:	the name of the file containing the Motion Sensor Master Key – VU part
msmk_wc:	the name of the file containing the Motion Sensor Master Key – Workshop part.

Example: derive msmk msmk(1) msmk_vu(1).bin msmk_wc(1).bin
Output: A file named msmk(1).bin, containing the resulting motion sensor master key.

Errors:

- See section 4.1.2.
- If the lengths of the AES keys in the files indicated by 'msmk_vu' and 'msmk_wc' are not equal.

Warnings:

- If the files indicated by 'msmk_vu' and 'msmk_wc' do not contain a key of 128, 192 or 256 bits.

4.2.11 Derive msik

This command is used to derive a motion sensor identification key by XOR-ing the Motion Sensor Master Key with a constant vector CV, as described in section 9.2.1.1 of Appendix 11 to Annex 1C [4].

Syntax:
derive msik <name> <msmk>

name:	the base name for the output file
msmk:	the name of the file containing the Motion Sensor Master Key

Example: derive msik msik(1) msmk(1).bin
Output: A file named msik(1).bin, containing the resulting motion sensor identification key.

Errors:

- See section 4.1.2.
- If the file indicated by 'msmk' does not contain a key of 128, 192 or 256 bits.

Warnings: None

4.2.12 Encrypt ms

This command is used to encrypt a motion sensor extended serial number with an identification key. The tool will derive the identification key from the specified motion sensor master key. The encrypted motion sensor extended serial number will be stored in a file named <name>-esn-enc.bin.

Syntax: There are two possible syntaxes for this command, one using separate values for the serial number, month, year and manufacturer code, the other using a single value for the extended serial

number. If the first syntax is used, the tool concatenates these inputs together with the value for the motion sensor equipment type ('07') into the motion sensor extended serial number. If the second syntax is used, the tool uses the extended serial number as specified.

Syntax 1:
encrypt ms <name> <msmk> <serialnumber> <month> <year> <manufacturercode>

name:	the base name for the output file
msmk:	the name of the file containing the motion sensor master key
serialnumber:	the 4-byte equipment serial number; see Appendix 1 to Annex 1C [4].
month:	the month of manufacturing of the motion sensor. Format: 1 byte BCD. A warning will be given if the number entered is higher than 12 (decimal).
year:	the year of manufacturing of the motion sensor. Format: 1 byte BCD. If more than 2 digits are entered, the tool will only use the last two; e.g. 2017 becomes 17.
manufacturercode:	the 1-byte manufacturer code; see Appendix 1 to Annex 1C [4].

Errors:
- See section 4.1.2.
- If the file indicated by 'msmk' does not contain a key of 128, 192 or 256 bits.

Warnings:
- See note above for 'month'.

Syntax 2:
encrypt ms <name> <msmk> <extendedserialnumber>

name:	the base name for the output file
msmk:	the name of the file containing the motion sensor master key
extendedserialnumber:	the 16-byte equipment extended serial number; see Appendix 1 to Annex 1C [4]. Format is fixed-size 16 digits (hexadecimal).

Errors:
- See section 4.1.2.
- If the file indicated by 'msmk' does not contain a key of 128, 192 or 256 bits.

Example 1: encrypt ms ms_01 msmk_01.bin 0x0000000A 5 2018 0x45
Output: A file named ms_01-esn-ecn.bin, containing the encrypted motion sensor extended serial number.

Example 2: encrypt ms ms_01 msmk_01.bin 0000000A05180745
Output: A file named ms_01-esn-ecn.bin, containing the encrypted motion sensor extended serial number.

Note that the result will be the same in both examples.

4.2.13 Encrypt pk

This command is used to encrypt a pairing key with a motion sensor master key. The encrypted pairing key will be stored in a file named <name>-pk-enc.bin

Syntax:

encrypt pk <name> <msmk> <pk>

name:	the base name for the output file
msmk:	the name of the file containing the motion sensor master key
pk:	the name of the file containing the pairing key

Example: encrypt ms01 msmk_01.bin pk_of_ms01.bin

Output: A file named ‘ms01-pk-enc.bin’, containing the encrypted pairing key.

Errors:

- See section 4.1.2.

Warnings:

- If the file indicated by ‘pk’ does not contain a key of 128, 192 or 256 bits.
- If a pairing key is encrypted with a motion sensor key of lower strength; e.g. if the pairing key is 256 bits and the motion sensor master key is 128 bits.

5 Using the tool

This chapter explains how to use the functions described in chapter 4 to create the desired keys and certificates:

- Section 5.1 describes how to use these functions to create valid asymmetric key pairs and certificates.
- Section 5.2 describes how to use these functions to create valid symmetric keys and other cryptographic material.
- Section 5.3 describes how to create invalid certificates, which may be used for unhappy flow testing.

5.1 Creating valid asymmetric key pairs and certificates

5.1.1 Creating an ERCA root key pair and certificate

Creating a valid ERCA root key pair and associated certificate involves the following steps:

1. Generate an ECC key pair using the 'generate ec' command, choosing the desired ECC domain parameters and key strength.
2. Create a self-signed certificate for the ECC key pair generated in the previous step, using the 'create ca' command with 'erca' as the type parameter and proper values for the other parameters.

5.1.2 Creating an ERCA Link certificate

Creating a valid ERCA link certificate involves the following steps:

1. Generate a first ERCA root key pair and certificate as described in the previous section.
2. Generate a second ERCA root key pair and certificate.
3. Create a link certificate between these two root certificates by using the 'link' command.

5.1.3 Creating a MSCA key pair and certificate

Creating a valid MSCA key pair and associated certificate involves the following steps:

1. Generate an ECC key pair using the 'generate ec' command, choosing the desired ECC domain parameters and key strength.
2. Create a self-signed certificate for the ECC key pair generated in the previous step, using the 'create ca' command with 'msca_card' or 'msca_vu-egf' as the type parameter and proper values for the other parameters.
3. Sign the output of step 2 with an ERCA root private key, using the 'sign' command.

5.1.4 Creating an equipment certificate

Creating a valid equipment key pair and associated certificate involves the following steps:

1. Generate an ECC key pair using the 'generate ec' command, choosing the desired ECC domain parameters and key strength.
2. Create the corresponding self-signed certificate for the ECC key pair generated in the previous step, using the 'create equipment' command with the desired type parameter and proper values for the other parameters.
3. Sign the output of step 2 with the proper MSCA private key (either a MSCA_Card or an MSCA_VU-EGF key), using the 'sign' command.

An exception is the creation of a VU certificate that does not contain a VU serial number, but a certificate request serial number. For such a certificate, the 'create request' command should be used in step 2 instead of the 'create equipment' command.

5.2 Creating valid symmetric keys and cryptographic material

5.2.1 Creating a DSRC Master Key, a Motion Sensor Master Key part or Pairing Key

In the Smart Tachograph system, a couple of symmetric keys are randomly generated. These are:

- the DSRC Master Key,
- the Motion Sensor Master Key – VU part ($K_{M\text{-}VU}$)
- the Motion Sensor Master Key – Workshop part ($K_{M\text{-}WC}$)
- the Pairing Key

To generate one of these keys, the 'generate aes' command should be used.

5.2.2 Creating a Motion Sensor Master Key

Creating a Motion Sensor Master Key involves the following steps:

1. Generate a Motion Sensor Master Key – VU part, as described in the previous section.
2. Generate a Motion Sensor Master Key – Workshop part, as described in the previous section.
3. Derive the Motion Sensor Master Key from these two keys by using the 'derive msmk' command.

5.2.3 Creating a Motion Sensor Identification Key

Creating a Motion Sensor Identification Key involves the following steps:

1. Create a Motion Sensor Master Key as described in the previous section.
2. Derive the Identification Key from the Motion Sensor Master Key by using the 'derive msik' command.

5.2.4 Creating VU-specific DSRC keys

Creating a set of VU-specific DSRC keys for a VU involves the following steps:

1. Create a DSRC Master Key as described in section 5.2.1.
2. Derive the VU-specific DSRC keys from this master key by using the 'derive dsrc' command.

5.2.5 Creating an encrypted motion sensor serial number

Creating an encrypted motion sensor serial number involves the following steps:

1. Create a motion sensor master key as described in section 5.2.2.
2. Encrypt the serial number with the identification key by using the 'encrypt ms' command.

Note that although the motion sensor master key is input to this command, the corresponding identification key will be used for encryption.

5.2.6 Creating an encrypted pairing key

Creating an encrypted pairing key involves the following steps:

1. Create a pairing key as described in section 5.2.1.
2. Create a motion sensor master key as described in section 5.2.3.
3. Encrypt the pairing key with the motion sensor master key by using the 'encrypt pk' command.

5.3 Creating invalid certificates

The tool will mainly produce valid keys and certificates. However, as described, the tool can produce output even if this output does not comply with the specifications in some respect. A warning will be given in such a case, but it is up to user to use the resulting output for test purposes or not.

Regarding the creation of certificates: as described, the various 'create' commands will automatically determine the value of some data elements. This reduces the number of input parameters and improves data elements' mutual consistency and compliance with the specifications. However, if a

user wants to create certificates that are not consistent and/or do not comply with the specifications, this is possible by following these steps:

1. Generate an ECC key pair using the ‘generate ec’ command.
2. Create the corresponding self-signed certificate for the ECC key pair generated in the previous step, using the appropriate ‘create’ command.
3. Open the certificate in any hex editor and replace bytes as needed.
4. Sign the output of step 3 with the appropriate CA private key, using the ‘sign’ command.

Note, however, that the tool will parse all input files according to the expected specifications, in particular Appendix 2 for .pkcs8 files and Table 3 for .cert files. If parsing is not possible, an error will be generated and no output will be produced. Examples include missing data elements, wrong lengths, etc.

6 Troubleshooting

Question: Command returns ‘Error: illegal key size’ if I try to use an AES longer than 128 bits or an ECC key longer than 256 bits for a cryptographic operations (e.g. sign, encrypt).

Answer: Please install Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy Files, as described in section 3.2.

References

Ref.	Title, author, version and date
[1]	Council Regulation (EC) No 2135/98 of 24th September 1998; Official Journal of the European Communities L274
[2]	Commission Regulation (EC) No 1360/2002 of 13th June 2002; Official Journal of the European Communities L207
[3]	Regulation (EU) No 165/2014 of the European Parliament and of the Council of 4 February 2014; Official Journal of the European Union L60
[4]	Commission Implementing Regulation (EU) 2016/799 of 18 March 2016; Official Journal of the European Union L 139
[5]	RFC 5958 Asymmetric Key Packages, S. Turner, August 2010
[6]	RFC 5912 (New ASN.1 Modules for the Public Key Infrastructure Using X.509 (PKIX)), P. Hoffman et al., June 2010
[7]	RFC 5915 (Elliptic Curve Private Key Structure), S. Turner et al., June 2010
[8]	Technical Guideline TR-03111 (Elliptic Curve Cryptography) v2.0, BSI, 2012-06-28
[9]	Smart Tachograph - Cryptographic keys and digital certificates sample set, version 1.2, April 2017
[10]	Java, https://www.oracle.com/java/index.html, April 2017
[11]	Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy Files, http://www.oracle.com/technetwork/java/javase/downloads/jce8-download-2133166.html, April 2017.
[12]	Bouncy Castle, https://www.bouncycastle.org/, April 2017
[13]	Maven, https://maven.apache.org/, April 2017

List of abbreviations and definitions

AES	Advanced Encryption Standard
CAR	Certificate Authority Reference
CEfD	Certificate Effective Date
CExD	Certificate Expiry Date
CHR	Certificate Holder Reference
CHA	Certificate Holder Authorisation
DSRC	Dedicated Short Range Communication
ECC	Elliptic Curve Cryptography
EGF	External GNSS Facility
ERCA	European Root Certificate Authority
GNSS	Global Navigation Satellite System
MA	Mutual Authentication
MS	Motion Sensor
MSCA	Member State Certificate Authority
PKI	Public Key Infrastructure
RCF	Remote Communication Facility
REDCR	Remote Early Detection Communication Reader
SHA-2	Secure Hash Algorithm 2
TC	Tachograph Card
VU	Vehicle Unit

List of figures

Appendix 1 Cryptographic elements per component

This Appendix describes all of the asymmetric keys, certificates and symmetric keys that are contained in each instance of the four main components of the Smart Tachograph system at the moment such a component is issued.

Appendix 1.1 Cryptographic elements installed in a Vehicle Unit

Asymmetric keys and certificates

Description	Purpose	Type	Source
VU private key and public key certificate for Mutual Authentication	Used by the VU to perform VU authentication towards tachograph cards and external GNSS facilities	ECC	Private key generated by VU or VU manufacturer. Certificate created and signed by MSCA
VU private key and public key certificate for signing	Used by the VU to sign downloaded data files	ECC	Private key generated by VU or VU manufacturer. Certificate created and signed by MSCA
ERCA root public key(s) and certificate(s)[2]	Used by the VU for the verification of MSCA certificates issued under the corresponding ERCA root certificate.	ECC	Generated by ERCA; inserted in VU by manufacturer at the end of the manufacturing phase or obtained from card or EGF during lifetime
Certificate of MSCA responsible for signing the VU_MA and VU_Sign certificates	Used by a card, EGF or dedicated equipment to obtain and verify the MSCA_VU-EGF public key they will subsequently use to verify the VU_MA or VU_Sign certificate		Created and signed by ERCA based on MSCA input; inserted by manufacturer at the end of the manufacturing phase

Table 4: Overview of VU asymmetric keys and certificates

Symmetric keys

Description	Purpose	Type	Source
Motion Sensor Master Key – VU part	Allowing a VU to derive the Motion Sensor Master Key if a workshop card is inserted into the VU.	AES	Generated by ERCA; inserted by VU manufacturer at the end of the manufacturing phase.
VU-specific DSRC keys for authenticity and confidentiality	Two separate keys used to ensure the authenticity confidentiality of data sent over a DSRC link between a RCF and a REDCR	AES	Derived by MSCA based on DSRC Master Key and VU serial number; inserted by VU manufacturer at the end of the manufacturing phase

Table 5: Overview of VU symmetric keys

[2] Note: Because of the regular replacement of the ERCA root key a VU may contain more than one ERCA certificates and Link certificates.

Appendix 1.2 Cryptographic elements installed in a Motion Sensor

Asymmetric keys and certificates

A motion sensor does not contain any asymmetric keys or certificates.

Symmetric keys

Description	Purpose	Type	Source
Motion sensor pairing key	Used by a VU for encrypting the motion sensor session key when sending it to the motion sensor during pairing.	AES	Generated by the motion sensor manufacturer; stored in motion sensor at the end of the manufacturing phase

Table 6: Overview of Motion Sensor symmetric keys

Apart from this key, a motion sensor contains the value of the pairing key encrypted under the motion sensor master key. It also contains the value of its serial number encrypted under the identification key[3].

Appendix 1.3 Cryptographic elements installed in a Tachograph Card

Asymmetric keys and certificates

Description	Purpose	Type	Source
Card private key and public key certificate for Mutual Authentication and session key agreement	Used by the card to perform card authentication towards VUs and perform session key agreement	ECC	Generated by card or card manufacturer/personaliser at the end of the manufacturing phase
Card private key and public key certificate for signing	Used by the card to sign downloaded data files.	ECC	Generated by card or card manufacturer/personaliser at the end of the manufacturing phase. Driver cards and workshop cards only
Certificate of MSCA responsible for signing the Card_MA and/or Card_Sign certificates	Used by a VU or dedicated equipment to obtain and verify the MSCA_Card public key they will subsequently use to verify the Card_MA or Card_Sign certificate	ECC	Created and signed by ERCA based on MSCA input; inserted by manufacturer at the end of the manufacturing phase
ERCA root public key(s) and certificate(s)	Used by the card for the verification of MSCA certificates issued under the corresponding ERCA root certificate.	ECC	Generated by ERCA; inserted in card by manufacturer at the end of the manufacturing phase or obtained from VU during lifetime

Table 7: Overview of TC asymmetric keys and certificates

[3] Note: Because the motion sensor master key and all associated keys are regularly replaced, up to three different encryptions of the pairing key and the serial number (based on consecutive generations of the motion sensor master key) may be present in a motion sensor.

Symmetric keys

Description	Purpose	Type	Source
Motion sensor master key – workshop card part[4]	Allowing a VU to derive the Motion Sensor Master Key if a workshop card is inserted into the VU	AES	Generated by ERCA; inserted in card by card manufacturer Workshop cards only
DSRC Master Key[5]	Master key to derive keys to protect confidentiality and authenticity of data sent from a VU to a control authority over a DSRC channel	AES	Generated by ERCA; inserted in card by card manufacturer Control and workshop cards only

Table 8: Overview of TC symmetric keys

Appendix 1.4 Cryptographic elements installed in an EGF

Asymmetric keys and certificates

Description	Purpose	Type	Source
EGF private key and public key certificate for Mutual Authentication	Used by the EGF to perform EGF authentication towards VUs	ECC	Private key generated by EGF or EGF manufacturer at the end of the manufacturing phase Certificate created and signed by MSCA
Certificate of MSCA responsible for signing the EGF_MA certificate	Used by a VU to obtain and verify the MSCA_VU-EGF public key it will subsequently use to verify the EGF_MA certificate	ECC	Created and signed by ERCA based on MSCA input; inserted by manufacturer at the end of the manufacturing phase
ERCA root public key(s) and certificate(s)	Used by the EGF for the verification of MSCA certificates issued under the corresponding ERCA root certificate.	ECC	Generated by ERCA; inserted in EGF by manufacturer at the end of the manufacturing phase or obtained from VU during lifetime

Table 9: Overview of EGF asymmetric keys and certificates

Symmetric keys

At issuance, an EGF does not contain any symmetric keys.

[4] Note: Because of the regular replacement of the ERCA root key and all associated keys, a workshop card may in fact contain up to three of these keys.

[5] Note: Because of the regular replacement of the ERCA root key and all associated keys, a control or workshop card may in fact contain up to three of these keys

Appendix 2 Format of .pkcs8 files

Format of .pkcs8 files created by the Sample Key and Certificate Generation Tool. All values hexadecimal.

30	**L**	SEQUENCE SIZE (1) OF OneAsymmetricKey; see RFC 5958 [5]. Both of the optional elements attributes and publicKey in this data type are omitted									
		02	**01**	**00**	Version; the value is set to '00' to indicate that the format of OneAssymmetricKey is equal to that of PrivateKeyInfo as specified in RFC 5280						
		30	**L**	PrivateKeyAlgorithmIdentifier							
				06	**07**	**2A 86 48 CE 3D 02 01**	PUBLIC-KEY: Algorithm identifier for elliptic curve is given in RFC 5912 **[6]**				
				06	**L**	**V**	PrivateKeyAlgorithms: see data type ECParameters in RFC 5912 **[6]**. The CHOICE made here is to use a namedCurve; the value is the DER-encoded OID of the relevant curve.				
		04	**L**	OCTET STRING containing private key; see RFC 5958 [5]							
				30	**L**	ECPrivateKey; see RFC 5915 [7]. Both of the optional elements parameters and publicKey in this data type are present.					
						02	**01**	**01**	version; the value represents ecPrivkeyVer1.		
						04	**L**	**V**	OCTET STRING containing the value of the private key		
						A0	**L**	parameters			
								06	**07**	**2A 86 48 CE 3D 02 01**	ECParameters; see above
						A1	**L**	publicKey			
								03	**L**	**V**	BITSTRING containing the value of the public key. Note that the first byte '00' indicates zero empty bits, as per the definition of the ASN.1 BITSTRING data type. The second byte '04' indicates the uncompressed encoding, as per TR 03111 **[8]**

JRC Mission

As the science and knowledge service of the European Commission, the Joint Research Centre's mission is to support EU policies with independent evidence throughout the whole policy cycle.

EU Science Hub
ec.europa.eu/jrc

@EU_ScienceHub

EU Science Hub - Joint Research Centre

Joint Research Centre

EU Science Hub

Publications Office