VECTOR

Unlocking ECU Resources by Seed & Key

2021-07-01
Support Note SN-IMC-1-104

Author(s) Marx, Alexander
Restrictions Public Document

Table of Contents

1 Introduction

Depending on the implementation of the embedded software, some ECU resources are protected by default (e.g. calibration, measurement data acquisition) and can be used only, if they are unlocked. Using **CCP/XCP** the so called “Seed and Key” method is used to unlock the protection.

This document describes the Seed & Key functionality for the different protocols in CANape and some common problems with the Seed & Key function DLL file embedded in CANape.

2 Seed & Key with CCP protocol

In CCP the three resources CAL, DAQ and PGM could be protected. The Seed & Key information for CCP will be included in the CCP specification.

2.1 CCP protocol commands for Seed & Key

Information from the protocol commands EXCHANGE_ID, GET_SEED and UNLOCK are necessary for the Seed & Key functionality in CCP. The following information will be a short extract from the CCP 2.1 specification document.

EXCHANGE_ID:

Command

Position	Type	Description
0	byte	Command Code = EXCHANGE_ID 0x17
	byte	Command Counter = CTR
2 …	byte	CCP master device ID information (optional and implementation specific)

The CCP master and slave stations exchange IDs for automatic session configuration. This might include automatic assignment of a data acquisition setup file depending on the slave's returned ID (Plug´n´Play). The following return information is expected (contents of returned DTO:)

Positive Response

Position	Type	Description
0	byte	Packet ID: 0xFF
	byte	Command Return Code
2	byte	Command Counter = CTR
3	byte	length of slave device ID in bytes
4	byte	data type qualifier of slave device ID (optional and implementation specific)
5	byte	Resource Availability Mask
6	byte	Resource Availability Mask
7	byte	Don’t care

The information in the Resource Availability Mask and the Resource Protection Mask is bit coded. Bit

7	6	5	4	3	2	1	0
X	PGM	X	X	X	X	DAQ	CAL

GET_SEED:

Command

Position	Type	Description
0	byte	Command Code = GET_SEED 0x12
1	byte	Command Counter = CTR

2	byte	Requested slave resource or function (Resource Mask)
3 … 7	byte	Don't care

See EXCHANGE_ID for a description of the resource mask.

Only one resource or function may be requested with one GET_SEED command. If more than one resource is requested, the GET_SEED command together with the following UNLOCK command, has to be performed multiple times.

Returns ´seed´ data for a **Seed & Key** algorithm for computing the ´key´ to unlock the requested function for authorized access (see 'Unlock Protection' below).

The following return information is expected (Contents of returned DTO:) Positive Response

Position	Type	Description
0	byte	Packet ID: 0xFF
1	byte	Command Return Code
2	byte	Command Counter = CTR
3	byte	Protection status (TRUE or FALSE)
4 … 7	byte	bytes ‘seed’ data

UNLOCK:

Command

Position	Type	Description
0	byte	Command Code = UNLOCK 0x13
1	byte	Command Counter = CTR
2 …	byte	`key”

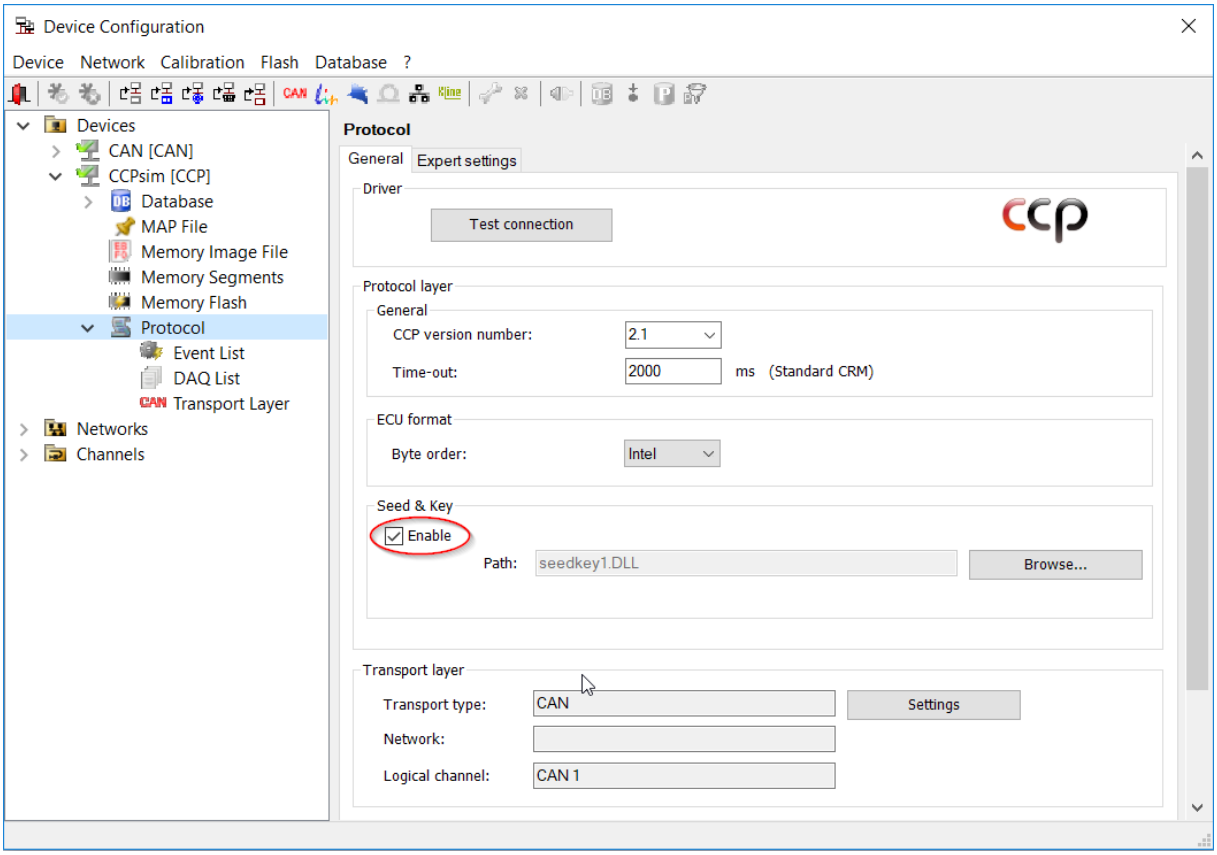
Unlocks the slave devices security protection (if applicable) using a ´key´ computed from ´seed´. See Seed & Key above. The following return information is expected (Contents of returned DTO:)

Positive Response

Position	Type	Description
0	byte	Packet ID: 0xFF
1	byte	Command Return Code
2	byte	Command Counter = CTR
3	byte	Current Privilege Status (Resource Mask)
4 … 7	byte	don't care

2.2 Seed & Key configuration in CANape

Enable or disable the Seed & Key functionality in the driver settings of the CCP device in CANape.



The Seed & Key DLL files must be in the defined directory of the CANape project. One Seed & Key DLL file could be used for one or more protected resources. This will depend on the description in the a2l file.

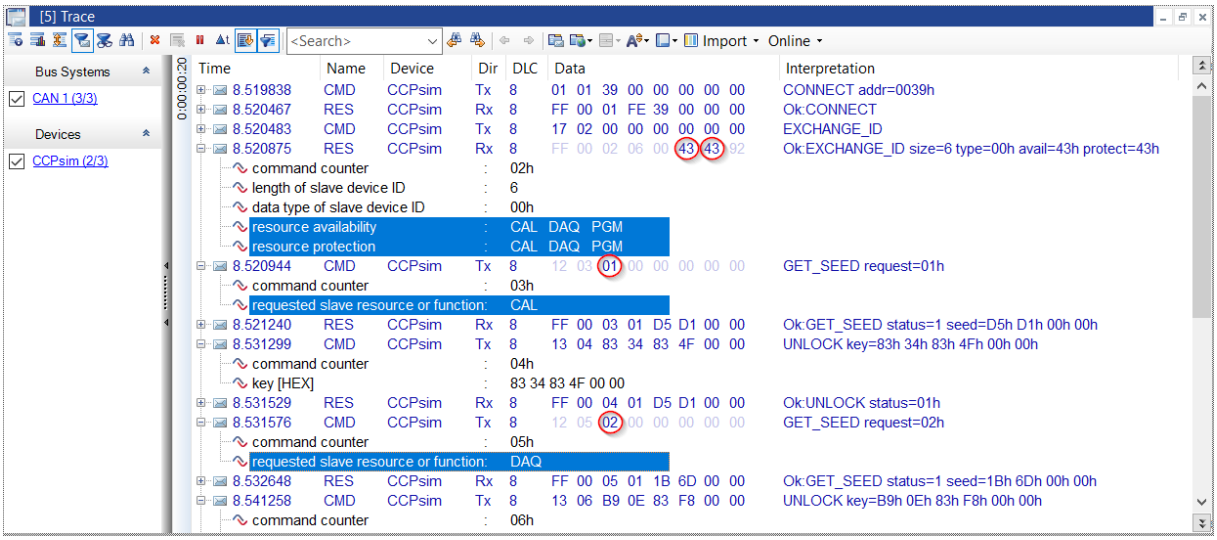
A2L file extract as example:

```
AML description and the IF_DATA section of an A2L file.


/*********************************************************/
/*   CCP AML   AML Version V2.6, 18.10.2002   */
/*********************************************************/ "ASAP1B_CCP"
taggedstruct {


…
/* Description of the authentification process */ block "SEED_KEY" struct {
char[256]; /* Name of the Seed&Key DLL for CAL Priviledge, including file-Extension
without path */
char[256]; /* Name of the Seed&Key DLL for DAQ Priviledge, including file-Extension
without path */
char[256]; /* Name of the Seed&Key DLL for PGM Priviledge, including file-Extension
without path */
};
…
/begin IF_DATA ASAP1B_CCP
…
/begin SEED_KEY
        "SEEDKEY1.DLL" "SEEDKEY1.DLL" "SEEDKEY1.DLL"
    /end SEED_KEY
 …
```

2.3 CCP Trace example for Seed & Key

In the following example the connection to an ECU will be established. This example based on the CCPsim demo project of CANape. After the connect command to an ECU CANape will only unlock the CAL (calibration) and DAQ (measurement data acquisition) resource. PGM (programming) resource will be unlocked directly before a flash command will be send.

Connect Trace:



Resource availability and resorce protection in the EXCHANGE_ID response should match. Only available resources could also be protected and unlocked in a following step.

EXCHANGE_ID:

The information in the Resource Protection Mask will define which resources are protected.

2h 34m 52.926249s	CMD	Tx	8	17 36 00 00 00 00 00 00	EXCHANGE_ID
2h 34m 52.926398s	RES	Rx	8	FF 00 36 06 00 43 43 92	Ok: EXCHANGE_ID

Resource Protection Mask = 43h = 01000011 Resource Availability Mask = 43h = 01000011

CAL (calibration), DAQ (measurement data acquisition) and PGM (programming) are available and resource protected in this example.

GET_SEED:

The GET_SEED command will send a request to the device for one protected resource and will get back a seed. This seed will be needed for the calculation of the Key in a separate Seed & Key DLL file to un-protect the protected resource.

2h 34m 52.926423s	CMD	Tx	8	12 37 01 00 00 00 00 00	GET_SEED
2h 34m 52.926573s	RES	Rx	8	FF 00 37 01 34 EC 00 00	Ok: GET_SEED

Resource Mask = 01h = 00000001

In this example the GET_SEED command will send a request for a seed for the protected resource CAL (calibration). This seed will be needed for the calculation of the Key in a Seed & Key DLL file.

UNLOCK:

The return value from the Seed & Key DLL file will be send to the device by the UNLOCK command to enable the requested resource.

2h 34m 52.928348s	CMD	Tx	8	13 38 C6 2A 03 90 00 00	UNLOCK
2h 34m 52.928544s	RES	Rx	8	FF 00 38 01 34 EC 00 00	Ok: UNLOCK

The positive response from the ECU will confirm unlock status of the protected resource. In this example the current privilege status is 01h. That means that CAL is unlocked in the ECU.

01h = 00000001 = CAL is now unlocked. DAQ and PGM are locked at this moment. 03h = 00000011 = CAL and DAQ are now unlocked.

2.4 How to create a CCP Seed & Key DLL file

The following description about the creation of a Seed & Key DLL file for CCP could be found in the ASAM MCD 2MC / ASAP2 Interface Specification.

8.7 Win32 API for the ASAP1a CCP Seed & Key algorithm DLL Author: Michael Rossmann, SIEMENS

In order to have a common implementation of the Seed & Key algorithms used for getting access to a locked, CCP accessed ECU, the following API is proposed:

Function name:	ASAP1A_CCP_ComputeKeyFromSeed
Parameter 1:	Pointer to the Seed data, retrieved from ECU's GET_SEED cmd, 4 BYTEs of data.
Parameter 2:	Pointer to 6 BYTEs of data, returning the calculated Key.

An example of an implementation of such an API written in C++ is like follows:

```
/*
// Header file for ASAP1a CCP V2.1 Seed&Key Algorithm
*/
#ifndef
_SEEDKEY_H_
#define
_SEEDKEY_H_
#ifndef
DllImport
#define DllImport_declspec(
dllimport ) #endif
#ifndef DllExport
#define DllExport_declspec(
dllexport ) #endif
#ifdef SEEDKEYAPI_IMPL // only defined by
implementor of SeedKeyApi #define SEEDKEYAPI
DllExport_cdecl
#else
#define SEEDKEYAPI DllImport
cdecl #endif
#ifdef
cplusplus
extern "C"
{ #endif
BOOL SEEDKEYAPI ASAP1A_CCP_ComputeKeyFromSeed (BYTE
*Seed, unsigned short SizeSeed,
BYTE *Key,
unsigned short
MaxSizeKey,
unsigned short
*SizeKey);
// Seed: Pointer to seed data
// SizeSeed:Size of seed data (length of ‚Seed‘)
// Key: Pointer, where DLL should insert the calculated key data.
// MaxSizeKey: Maximum size of ‚Key‘.
// SizeKey: Should be set from DLL corresponding to the number of data
// inserted to ‚Key‘ (at most ‚MaxSizeKey‘)
// Result: The value FALSE (= 0) indicates that the key could not be
// calculated from seed data (e.g. ‚MaxSizeKey‘ is too small).
// TRUE (!= 0) indicates success of
key calculation. #ifdef_cplusplus
}
#endif
#endif //_SEEDKEY_H_
```

```
//
// Implementation of Seed&Key DLL for ASAP1a CCP V2.1
//
#include <windows.h>
#include <memory.h>
#define
SEEDKEYAPI_IMPL
#include
"..\seedkey.h" extern
"C" {
BOOL SEEDKEYAPI ASAP1A_CCP_ComputeKeyFromSeed (BYTE
*Seed, unsigned short SizeSeed,
BYTE *Key,
unsigned short
MaxSizeKey, unsigned
short *SizeKey)
{
/* ... implementation of
seed&key algorithm. The result
should be written in Key.
the result of the CCP 2.1 cmd GET_SEED.
*/
MessageBox(NULL, "Call to ASAP1A_CCP_ComputeKeyFromSeed",
"Seed&Key", MB_OK);
}
}
```

Vector provides a Microsoft Visual C++ development kit intended to create custom “Seed & Key” DLLs. You could find this example also on the CANape installation CD / DVD. The development kit includes the following files:

SeedKey1.h	Header File including the interface description. You should not modify this file.
seedkey1.cpp	Source code of the subroutines which calculate the key. Please insert your custom key calculation routines
seedkey1.dsp	Microsoft Visual C++ project file.
seedkey1.dsw	Microsoft Visual C++ workspace.
StdAfx.h	Include file for standard system include files.
StdAfx.cpp	Source file that includes just the standard includes
ReadMe.Txt	Includes the readme.

3 Seed & Key with XCP protocol

In XCP the four resources CAL, DAQ, STIM and PGM could be protected. The Seed & Key information for XCP will be included in the XCP Protocol Layer Specification (Part 2)**.** The interface to an external Seed & Key function will be defined in the XCP Interface Specification (Part 4).

3.1 XCP protocol commands for Seed & Key

Information from the protocol commands GET_STATUS, GET_SEED and UNLOCK are necessary for the Seed & Key functionality in XCP. The following information will be a short extract from the XCP Protocol Layer Specification (Part 2) document.

GET_STATUS:

Command

Position	Type	Description
0	byte	Command Code = OxFD

This command returns all current status information of the slave device. This includes the status of the resource protection, pending store requests and the general status of data acquisition and stimulation.

Positive Response

Position	Type	Description
0	byte	Packet ID: 0xFF
1	byte	Current session status
2	byte	Current resource protection status
3	byte	Reserved
4,5	word	Session configuration id

The Current Resource Protection Status parameter is a bit mask described below: Bit

7	6	5	4	3	2	1	0
X	X	X	PGM	STIM	DAQ	X	CAL/PAG

Resource protection status examples:

00000001 = 01h = CAL/PAG (CALibration/PAGing commands) 00000100 = 04h = DAQ (DAQ list commands (DIRECTION = DAQ))

00001000 = 08h = STIM (DAQ list commands (DIRECTION = STIM))

00010000 = 10h = PGM (ProGraMming commands)

GET_SEED:

Command

Position	Type	Description
0	byte	Command Code = 0xF8
1	byte	Mode 0 = (first part of) seed 1 = remaining part of seed
2	byte	Mode=0: Resource Mode=1: Don't care

With Mode = 0, the master requests the slave to transmit (the first part of) the seed. The slave answers with (the first part of) the seed and the total length of the seed.

With Mode = 1, the master has to request the remaining part(s) of the seed from the slave if the total length of the seed is bigger than MAX_CTO-2.

The master has to use GET_SEED(Mode=1) in a defined sequence together with GET_SEED(Mode=0). If the master sends a GET_SEED(Mode=1) directly without a previous GET_SEED(Mode=0), the slave returns an ERR_SEQUENCE as negative response.

See command GET_STATUS (resource protection status) for a description for the values of the resource parameter (CAL/PAG, DAQ, STIM, PGM) and the related commands.

Only one resource may be requested with one GET_SEED command. If more than one resource has to be unlocked, the (GET_SEED+UNLOCK) sequence has to be performed multiple times. If the master does not request any resource or requests multiple resources at the same time, the slave will respond with an ERR_OUT_OF_RANGE.

Positive Response

Position	Type	Description
0	byte	Packet ID: 0xFF
1	byte	Length of seed [BYTE] Length = 0 resource unprotected Mode = 0 : total length of seed Mode = 1 : remaining length of seed
2 .. MAX_CTO-1	byte	Seed

Length indicates the (remaining) number of seed bytes. If Length = 0, the resource is unprotected and no UNLOCK command is necessary.

A GET_SEED sequence returns the ´seed´ data for a **Seed & Key** algorithm computing the 'key´ to un- lock the requested resource category for authorized access (see the UNLOCK command).

The master has to calculate the key by calling an external function file. There's only 1 external function file which might contain from 1 up to 4 different algorithms, one algorithm for each of the resources CAL/PAG, DAQ, STIM or PGM.

The external function file supplier can enable/disable the use of each of these 4 algorithms. The master can get the information about the ability of the algorithms directly from the external function file.

The external function file supplier can compile different versions of the external function file by making different combinations of enabled algorithms.

The master gets the name of the external function file to be used for this slave, from the ASAM MCD 2MC description file. The API for communicating with the external function file is specified in Part 4 of the XCP specification.

UNLOCK:

Command

Position	Type	Description
0	byte	Command Code = 0xF7
1	byte	(remaining) Length of key in bytes
2 .. MAX_CTO-1	byte	Key

Unlocks the slave device's security protection using a ´key´ computed from the ´seed´ obtained by a previous GET_SEED sequence. See the description of the GET_SEED command.

Length indicates the (remaining) number of key bytes.

The master has to use UNLOCK in a defined sequence together with GET_SEED.

The master only can send an UNLOCK sequence if previously there was a GET_SEED sequence. The master has to send the first UNLOCK after a GET_SEED sequence with a Length containing the total length of the key.

If the total length of the key is bigger than MAX_CTO-2, the master has to send the remaining key bytes with (a) consecutive UNLOCK command(s) containing the remaining length of the key.

If the master does not respect this sequence, the slave returns an ERR_SEQUENCE as negative response.

The key is checked after completion of the UNLOCK sequence. If the key is not accepted, ERR_ACCESS_LOCKED will be returned. The slave device will then go to disconnected state. A repetition of an UNLOCK sequence with a correct key will have a positive response and no other effect.

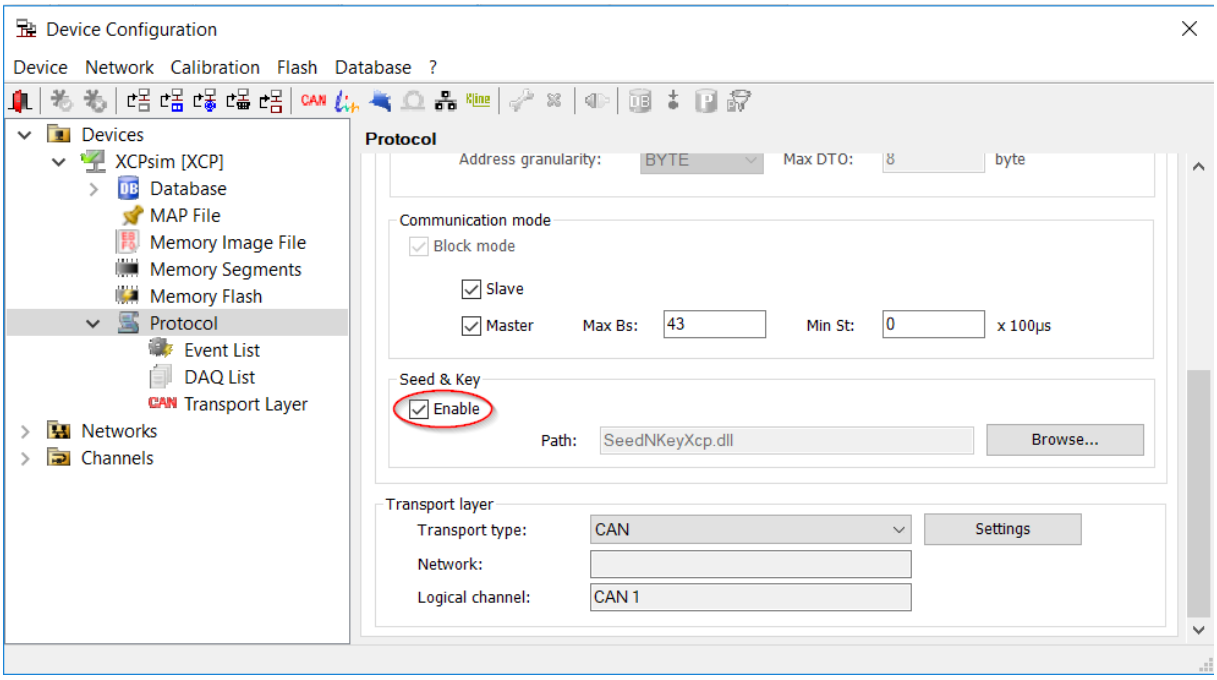
Positive Response

Position	Type	Description
0	byte	Packet ID: 0xFF
1	byte	Current resource protection status

The answer upon UNLOCK contains the Current Resource Protection Mask as described at GET_STATUS.

3.2 Seed & Key configuration in CANape

Enable or disable the Seed & Key functionality in the driver settings of the XCP device in CANape.



It is possible to setup the file name and the path of the Seed & Key DLL file in this dialog. This information will be consistent to the information in the a2l file. There's only 1 external function file which might contain from 1 up to 4 different algorithms, one algorithm for each of the resources CAL/PAG, DAQ, STIM or PGM. More information about the Seed & Key DLL file will be included in the "XCP – Part 4 – Interface Specification" document.

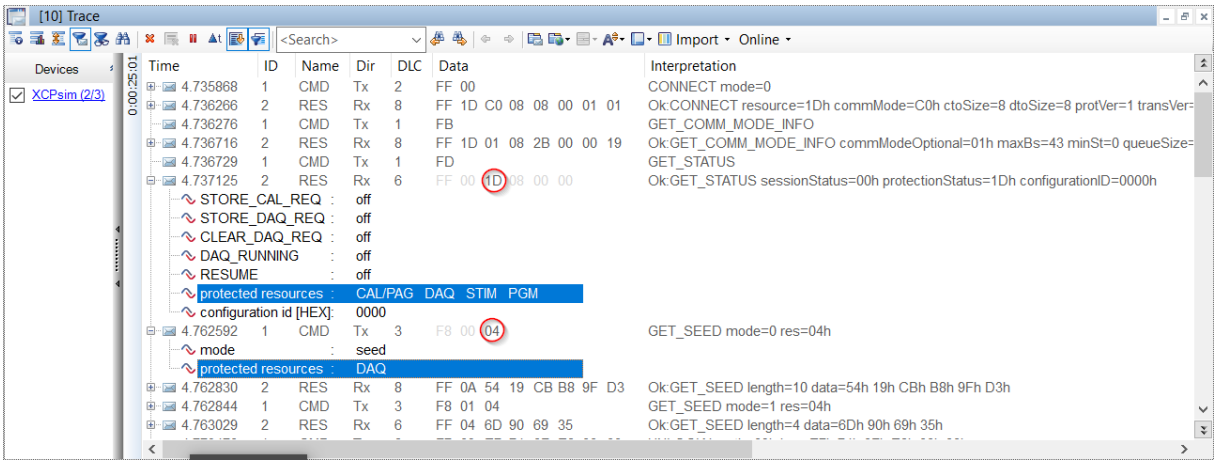
A2L file extract as example:

```
AML description and the IF_DATA section of an A2L file.

   /************************************************************/
   /* Definitions for XCP          */
   /************************************************************/

   struct Protocol_Layer {
…
     "SEED_AND_KEY_EXTERNAL_FUNCTION" char[256]; /* Name of the Seed&Key
     function */
    };
   };

…
  /begin IF_DATA XCP
…
   /begin PROTOCOL_LAYER
```

3.3 XCP Trace example for Seed & Key

In the following example the connection to an ECU will be established. This example based on the XCPsim demo project of CANape. After the connect command to an ECU CANape will normally unlock the CAL/PAG, DAQ and STIM resources. The PGM resource will be unlocked directly before a flash command will be send.

Connect Trace:



GET_STATUS:

The information in the Resource Protection Status will define which resources are protected.

4h 50m 11.196276s	CMD	Tx	1	FD	GET_STATUS
4h 50m 11.196425s	RES	Rx	6	FF 00 1D 08 00 00	Ok:GET_STATUS

Resource Protection Status = 1Dh = 00011101

CAL/PAG, DAQ, STIM and PGM are resource protected in this example.

GET_SEED:

The GET_SEED command will send a request to the device for one protected resource and will get back a seed. This seed will be needed for the calculation of the Key in a separate Seed & Key DLL file to un-protect the protected resource.

4h 50m 11.201877s	CMD	Tx	3	F8 00 01	GET_SEED mode=0 res=01h
4h 50m 11.202051s	RES	Rx	3h 6	FF 04 19 20 7F 53	Ok:GET_SEED length=4 data=19h 20h 7Fh 5

Resource = 01h = 00000001

In this example the GET_SEED command will send a request for a seed for the protected resource CAL/PAG. This seed will be needed for the calculation of the Key in a Seed & Key DLL file.

UNLOCK:

The return value from the Seed & Key DLL file will be send to the device by the UNLOCK command to enable the requested resource.

4h 50m 11.210434s	CMD	Tx	5	F7 03 61 D6 40 UNLOCK length=3 key=61h D6h 40h
4h 50m 11.211293s	RES	Rx	2	FF 1C Ok:UNLOCK status=1Ch

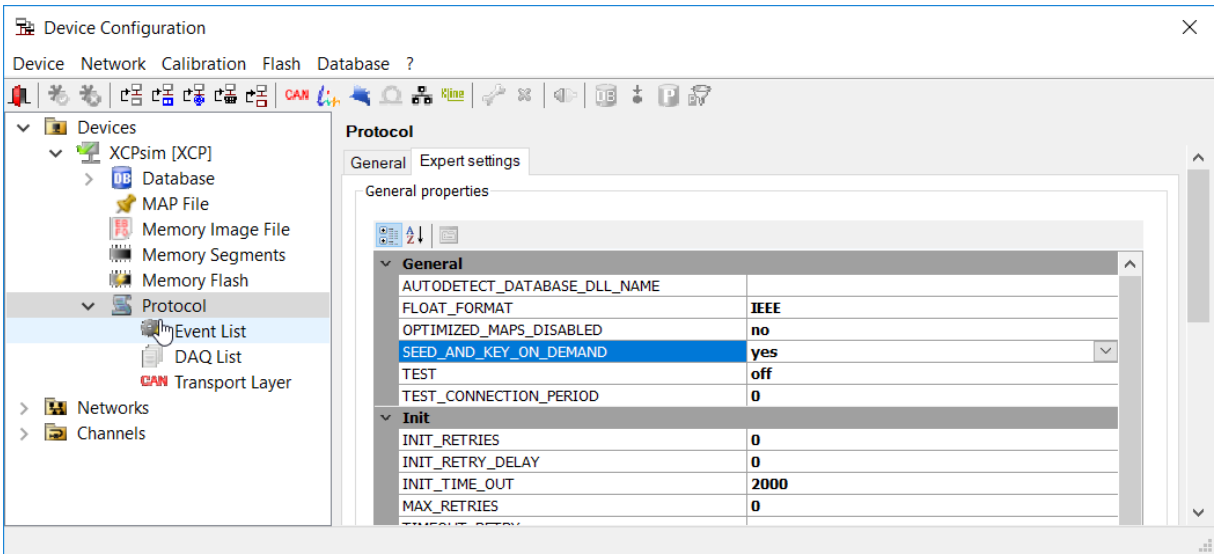
The positive response from the ECU will confirm unlock status of the protected resource. In this example the current resource protection status is 1Ch. That means that CAL is unlocked in the ECU.

1Ch = 11100 = CAL is now unlocked. DAQ, STIM and PGM are locked at this moment. 18h = 11000 = CAL and DAQ are now unlocked.

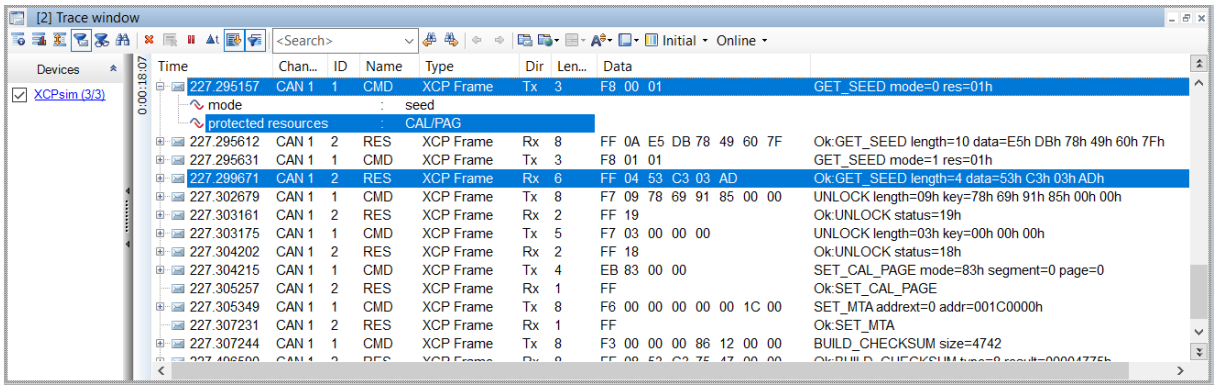
10h = 10000 = CAL, DAQ and STIM are now unlocked.

3.4 Expert settings for XCP Seed & Key

The option “SEED_AND_KEY_ON_DEMAND” will cause that the protected resources will be unlocked only if the resource will be used. A protected resource has to be unlocked only once per session.



Calibration example (resource CAL):



Before the SET_MTA command and the DOWNLOAD command the protected resource CAL/PAG will be unlocked.

3.5 How to create a XCP Seed & Key DLL file

The following explanation will be an abstract from the XCP Interface Specification Part 4

2 Interface to an external Seed & Key function

When calculating a Key from a Seed, the Master always has to use a user-defined algorithm. This algo- rithm is provided by the slave vendor. It contains functions to read out the provided privileges and to calculate a Key from a Seed.

The “SEED_AND_KEY_EXTERNAL_FUNCTION” parameter at the “PROTOCOL_LAYER” section in the ASAM MCD 2MC Description File, indicates the Name of the external function file the Master has to use. The parameter is an ASCII string that contains the name and the extension but does not contain the path to the file.

The integration of this function file is programming language and platform dependent. E.g. when using a Windows ® operating system, these “external functions” could be located in a MySeedNKey.DLL (Dynamically Linked Library). When using a UNIX ® operating system, these “external functions” could be located in a MySeedNKey.SO (Shared Object).

The mechanism required to include external functions files is tool specific. However, the included functions and calling parameters themselves are specified in this chapter.

To have an easy handling for XCP there is only one external function file which may contain all algorithms to unlock all privileges or only a subset. That means the supplier can generate different external function files with different privilege level combinations.

The privilege levels are described based on the “Resource Mask” of XCP and coded as defined there. The ECU needs one algorithm for each privilege (if protected).

The external function file contains 2 functions: one to get information about the available privileges of this function file and one to calculate a key from a seed for the requested privilege.

Function “XCP_GetAvailablePrivileges”:

Parameter name:	Data Type	XCP_ComputeKeyFromSeed	Remarks
Return Value:	DWORD	Error Code	
Parameter 1:	BYTE *	Available Privilege	returns the privileges with available unlock algorithms in this external function file

Function returns available privileges as XCP Resource Availability Mask. The following error codes can be returned: XcpSkExtFncAck: o.k.

Function: XCP_ComputeKeyFromSeed:

Parameter name:	Data Type	XCP_ComputeKeyFromSeed	Remarks
Return Value:	DWORD	Error Code	
Parameter 1:	BYTE *	Requested Privilege	=> from Tool, - input for external function – input for GetSeed command
Parameter 2:	BYTE	Byte Length Seed	from answer of GetSeed
Parameter 3:	BYTE *	Pointer to Seed	
Parameter 4:	BYTE *	Byte Length Key	input: max bytes memory for key output: byte length of key
Parameter 5:	BYTE *	Pointer to Key	

The external function “XCP_ComputeKeyFromSeed “ should calculate Key from Seed for the requested privilege

Key = f(Seed, RequestedPrivilege) (only one privilege can be unlocked at once)

Remark:

Parameter 4 “Byte Length Key” must be initialised with the maximum Length of Key reserved by the Master when calling the external Seed & Key function. This makes sure that the Seed & Key function will not write into other memory than reserved. It is recommended to reserve 255 bytes since this is the maximum length that is possible.

The following error codes can be returned:

XcpSkExtFncAck: o.k.
XcpSkExtFncErrPrivilegeNotAvailable: the requested privilege can not be unlocked with this function
XcpSkExtFncErrInvalidSeedLength: the seed length is wrong, key could not be computed
XcpSkExtFncErrUnsufficientKeyLength: the space for the key is too small

Example:

Example source code for a Windows ® -DLL can be downloaded from .
www.asam.net under “Standards/ ASAM MCD/ I. Current specifications”

4 Seed & Key with KWP protocol

KWP2000 devices use the service ‘SecurityAccess’ for unlocking resources. The proceeding compares to CCP/XCP:

Get seed value from ECU by using the service ‘SecurityAccess’ Calculate a key value

Send the key value to the ECU using the service ‘SecurityAccess’

Call the service ‘StartDiagnosticSession’ to enable a certain level of access rights.

4.1 KWP protocol commands for Seed & Key

The following extract will be a part of the International Standard ISO 14230-3 Keyword Protocol 2000.

- SecurityAccess service
- Parameter definition

The parameter ***accessMode (AM_)*** is used in the securityAccess service. It indicates to the server the step in progress for this service, the level of security the client wants to access and the format of seed and key. Values are defined in the table below for requestSeed and sendKey.

Hex	Description	Cvt	Mnemonic
00	reservedByDocument This value is reserved by this document.	M	RBD
01	requestSeed RequestSeed with the level of security defined by the vehicle manufacturer.	M	RSD
02	sendKey SendKey with the level of security defined by the vehicle manufacturer.	M	SK
03, 05, 07–7F	requestSeed RequestSeed with different levels of security defined by the vehicle manufacturer.	U	RSD
04, 06, 08 - 80	sendKey SendKey with different levels of security defined by the vehicle manufacturer.	U	SK
81 – FA	vehicleManufacturerSpecific This range of values is reserved for vehicle manufacturer specific use.	U	VMS
FB – FE	systemSupplierSpecific This range of values is reserved for system supplier specific use.	U	SSS
FF	reservedByDocument This value is reserved by this document.	M	RBD

Table 6.3.1.1 - Definition of accessMode values

The values of the parameter ***key*** are not defined in this document.

The parameter ***securityAccessStatus (SAS_)*** may be used to receive the status of the server security system. The value is defined in the table below.

Part 3 Implementation Recommended Practice addition:

The values of the parameter ***seed*** are not defined in this document except for the values **"$00...00" (all numbers are $00)** which indicates to the client (tester) that the server (ECU) is not locked and

"$FF...FF" (all numbers are $FF) which shall not be used because this value may occur if the server's (ECU's) memory has been erased.

Hex	Description	Mnemonic
34	securityAccessAllowed	SSA

Table 6.3.1.2 - Definition of securityAccessStatus value

- Message data bytes

Data Byte	Parameter Name	Cvt	Hex Value	Mnemonic
#1	securityAccess Request#1 Service Id	M	27	SA#1
#2	accessMode=[requestSeed: shall be an odd number greater than $01 if additional levels of security are supported, ($01 default), vehicleManufactu-rerSpecific, systemSupplierSpecific]	M	xx=[01, 03, : 7F, 81-F9, FB-FD]	AM_...

Table 6.3.2.1 - securityAccess Request#1 Message

Data Byte	Parameter Name	Cvt	Hex Value	Mnemonic
#1	securityAccess Positive Response#1 Service Id	S	67	SA#1PR
#2	accessMode=[requestSeed: shall be an odd number greater than $01 if additional levels of security are supported, ($01 default), vehicleManufacturerSpecific, systemSupplierSpecific]	M	xx=[01, 03, : 7F, 81-F9, FB-FD]	AM_...
#3 : #n	seed#1 (High Byte) : seed#m (Low Byte)	C : C	xx : xx	SEED

Table 6.3.2.2 - securityAccess Positive Response#1 Message C = condition: accessMode must be set to "requestSeed".

Data Byte	Parameter Name	Cvt	Hex Value	Mnemonic
#1	negativeResponse#1 Service Id	S	7F	NR
#2	securityAccess Request Service Id	S	27	SA
#3	responseCode=[KWP2000ResponseCode {section 4.4 }]	M	xx=[00-FF]	RC_...

Table 6.3.2.3 - SecurityAccess Negative Response#1 Message

Data Byte	Parameter Name	Cvt	Hex Value	Mnemonic
#1	securityAccess Request#2 Service Id	M	27	SA#2
#2	accessMode=[sendKey: shall be an even number one greater than accessMode in request#1 if additional levels of security are supported, ($02 default), vehicleManufacturerSpecific, systemSupplierSpecific]	M	xx=[02, 04, 80, 82-FA, FC-FE]	AM_...
#3	key#1 (High Byte)	C	xx	KEY
#n	key#m (Low Byte)	C	xx	

Table 6.3.2.4 - SecurityAccess Request#2 Message C = condition: accessMode must be set to "sendKey".

Data Byte	Parameter Name	Cvt	Hex Value	Mnemonic
#1	securityAccess Positive Response#2 Service Id	S	67	SA#2PR
#2	accessMode=[sendKey: shall be an even number one greater than AccessMode in request#1 if additional levels of security are supported, ($02 default) vehicleManufacturerSpecific, system-SupplierSpecific	M	xx=[02, 04, : 80, 82-FA, FC-FE]	AM_...
#3	securityAccessStatus=[securityAccessAllowed]	M	34	SAA

Table 6.3.2.5 - SecurityAccess Positive Response#2 Message

Data Byte	Parameter Name	Cvt	Hex Value	Mnemonic
#1	negativeResponse#2 Service Id	S	7F	NR
#2	securityAccess Request Service Id	S	27	SA
#3	responseCode=[KWP2000ResponseCode {section 4.4 }]	M	xx=[00-FF]	RC_...

Table 6.3.2.6 - SecurityAccess Negative Response#2 Message

- Message description

This mode is intended to be used to implement the data link security measures defined in SAE J2186 (E/E Data Link Security).

The client shall request the server to "unlock" itself by sending the service securityAccess request#1. The server shall respond by sending a "seed" using the service securityAccess positive response#1. The client shall respond by returning a "key" number back to the server using the service securityAccess request#2. The server shall compare this "key" to one internally stored. If the two numbers match, then the server shall enable ("unlock") the client's access to specific KWP 2000 services and indicate that with the service securityAccess positive response#2. If upon two (2) attempts of a service securityAc- cess request#2 by the client, where the two keys do not match, then the server shall insert a ten (10) second time delay before allowing further attempts.

The ten (10) second time delay shall also be required before the server responds to a service securi-tyAccess request#1 from the client after server power-on.

If a device supports security, but is already unlocked when a securityAccess request#1 is received, that server shall respond with a securityAccess positive response#1 service with a seed of "$00 00". A client shall use this method to determine if a server is locked by checking for a non-zero seed.

The security system shall not prevent normal diagnostic or vehicle communications between the client and the servers. Proper "unlocking" of the server is a prerequisite to the client's ability to perform some of the more critical functions such as reading specific memory locations within the server, downloading information to specific locations, or downloading routines for execution by the controller. In other words, the only access to the server permitted while in a "locked" mode is through the server specific software. This permits the server specific software to protect itself from unauthorized intrusion.

Servers which provide security shall support reject messages if a secure mode is requested while the server is locked.

Some servers could support multiple levels of security, either for different functions controlled by the server, or to allow different capabilities to be exercised. These additional levels of security can be accessed by using the service securityAccess requests#1 and #2 with an accessMode value greater than the default value. The second data byte of the "requestSeed" shall always be an odd number, and the second data byte of the service to "sendKey" shall be the next even number.

Part 3 Implementation Recommended Practice addition:

Some diagnostic sessions in a server (ECU) require a successful security access sequence in order to support/perform secured functions. In such case the following sequence of services shall be required:

startDiagnosticSession(diagnosticMode) service {section 6.1.1} securityAccess(...) service

The accessMode parameters of the securityAccess service depend on the diagnosticMode enabled (session started)

in the server (ECU).

- Message flow example

time	client (Tester)	server (ECU)
P3 P2	securityAccess.Request#1[...]	securityAccess.PositiveResponse#1[...]
P3 P2	securityAccess.Request#2[...]	securityAccess.PositiveResponse#2[...]

Table 6.3.4.1 - Message flow example of securityAccess services

Note: In case of a securityAccess negative response#1 it is the vehicle manufacturer's responsibility to decide how to proceed.

- Implementation example of securityAccess

- Test specification securityAccess

This section specifies the conditions to be fulfilled to successfully unlock the server (ECU) if in a "locked“ state: requestSeed = $01; sendKey = $02; seed = $3675; key = $C98B {e.g. 2's complement of seed value }

If the server sends negative response messages the client (tester) and the server (ECU) shall follow the rules specified in section 6.3.3 Message description.

- Message flow

- Message flow if server (ECU) is in a "locked" state (seed ¹ $0000) STEP#1 securityAccess(AM_RSD)

time	Client (tester) Request Message	Hex
P3	securityAccess.ReqSId[	27

	accessMode = requestSeed]	01

Time	Server (ECU) Positive Response Message	Hex	Server (ECU) Negative Response Message	Hex
P2	securityAccess.PosRspSId[	67	negativeResponse Service Identi-	7F
	accessMode = requestSeed	01	fier	27
	seed (High Byte)	36	securityAccess.ReqSId[	xx
	seed (Low Byte)]	75	responseCode { refer to section 4.4 }]	
	goto STEP#2		return(responseCode)	

STEP#2 securityAccess(AM_SK, KEY)

time	Client (tester) Request Message	Hex
P3	securityAccess.ReqSId[	27
	accessMode = sendKey	02
	key (High Byte) { 2's complement of seed }	C9
	key (Low Byte) { 2's complement of seed }]	8B

Time	Server (ECU) Positive Response Message	Hex	Server (ECU) Negative Response Message	Hex
P2	securityAccess.PosRspSId[accessMode = sendKey securityAccessAllowed]	67 02 34	negativeResponse Service Identifier securityAccess.ReqSId[responseCode { refer to section 4.4 }]	7F 27 xx
	return(main)		return(responseCode)	

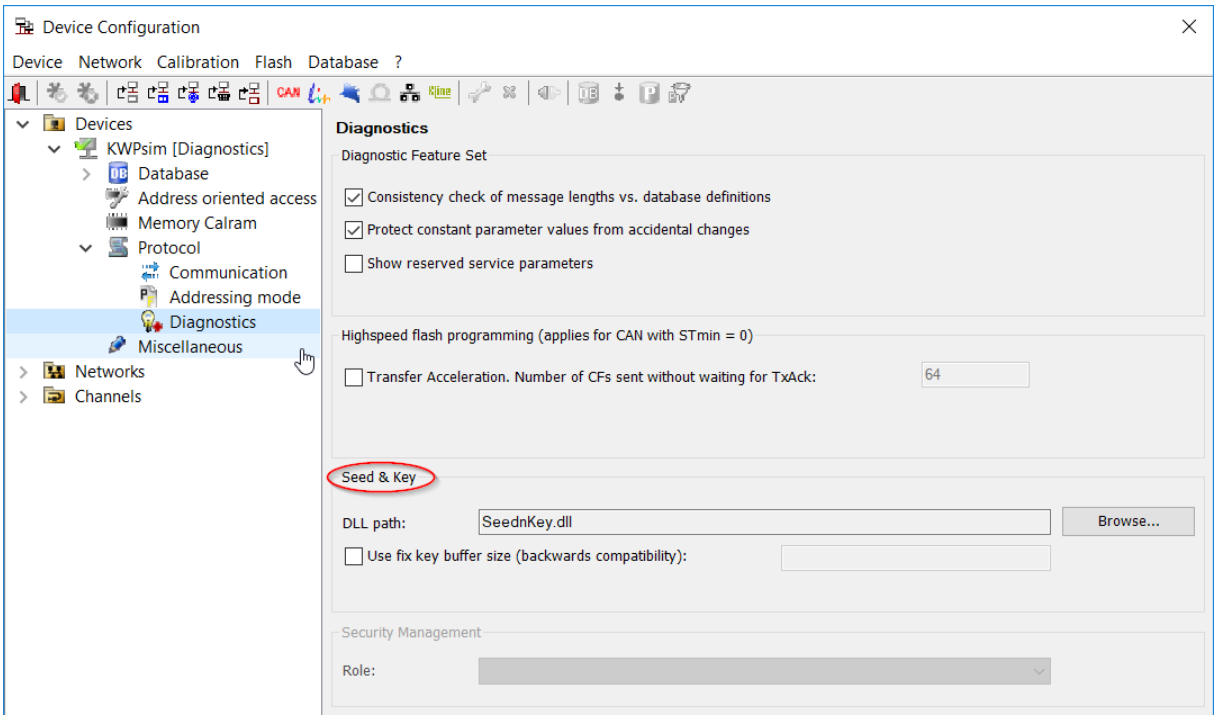
- Message flow if server (ECU) is in an "unlocked" state (seed = $0000) STEP#1 securityAccess(AM_RSD)

time	Client (tester) Request Message	Hex
P3	securityAccess.ReqSId[	27
	accessMode = requestSeed]	01

Time	Server (ECU) Positive Response Message	Hex	Server (ECU) Negative Response Message	Hex
P2	securityAccess.PosRspSId[	67	negativeResponse Service Identi-	7F
	accessMode = requestSeed	01	fier	27
	seed (High Byte)	00	securityAccess.ReqSId[	xx
	seed (Low Byte)]	00	responseCode { refer to section 4.4 }]	
	return(main)		return(responseCode)	

4.2 Seed & Key configuration in CANape

Enable or disable the Seed & Key functionality in the driver settings of the KWP device in CANape.



4.3 KWP Trace example for Seed & Key

-

4.4 How to create a KWP Seed & Key DLL file

In KWP there is on more function parameter necessary in the Seed & Key DLL file than for CCP. Because of this the example could look like the following:

```
unsigned long_cdecl KWP2000_ComputeKeyFromSeed(unsigned char Mode,
unsigned char *Seed, unsigned short SizeSeed, unsigned char *Key, unsigned
short MaxSizeKey, unsigned short *SizeKey);
```

Vector provides a Microsoft Visual C++ development kit intended to create custom “Seed & Key” DLLs. You could find this example also on the CANape installation CD / DVD. The development kit includes the following files:

SeedKey1.h	Header File including the interface description. You should not modify this file.
seedkey1.cpp	Source code of the subroutines which calculate the key. Please insert your custom key calculation routines
seedkey1.dsp	Microsoft Visual C++ project file.
seedkey1.dsw	Microsoft Visual C++ workspace.
StdAfx.h	Include file for standard system include files.
StdAfx.cpp	Source file that includes just the standard includes

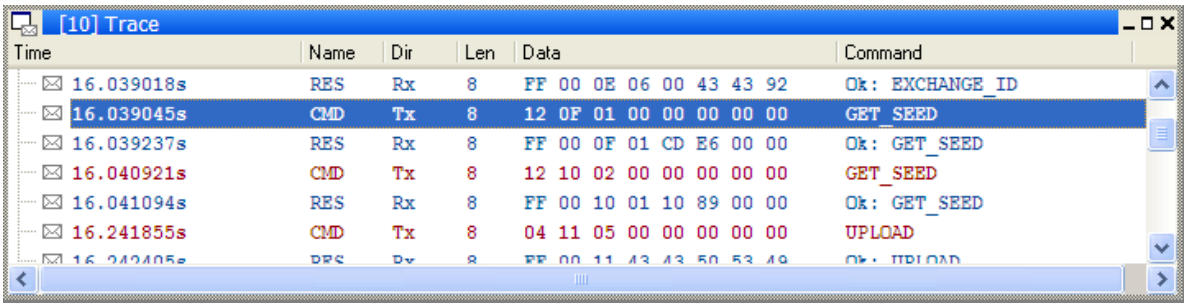
ReadMe.Txt	Includes the readme.

5 Problems with CCP and Seed & Key:

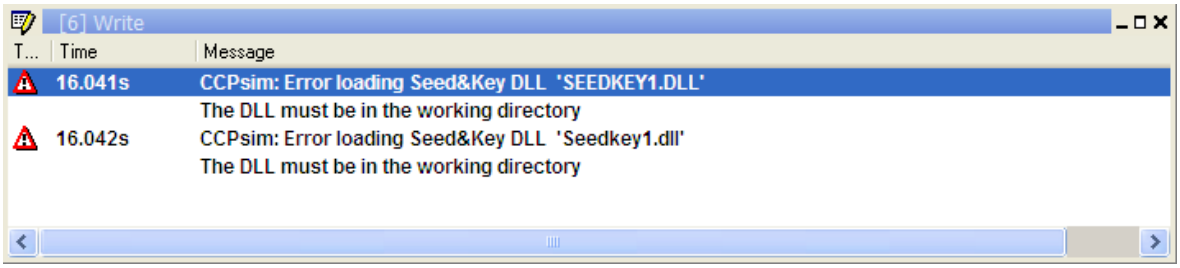
There could be different reasons why the Seed & Key functionality in CCP could fail. The following exam- ples try to show a way to fix some known problems which have been occurred with CANape in the past.

5.1 Error loading CCP Seed & Key DLL file

A problem with the Seed & Key DLL file will exist when the command GET_SEED will be send but the command UNLOCK is missing in the protocol trace.



A message in the write window displays that the Seed & Key DLL file could not been found.

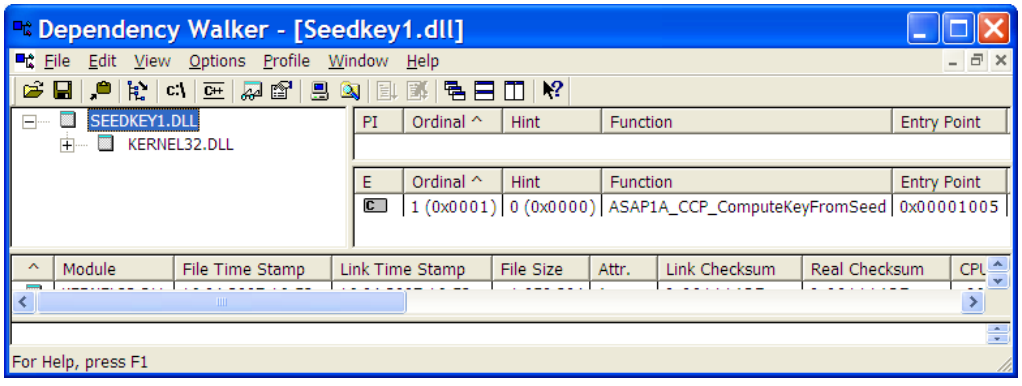


Reason 1:

File name or path of the Seed & Key DLL file is not correct. Please verify file name and path in the driver settings of the device. You could find more information in chapter 2.2 (Seed & Key configuration in CANape) of this document.

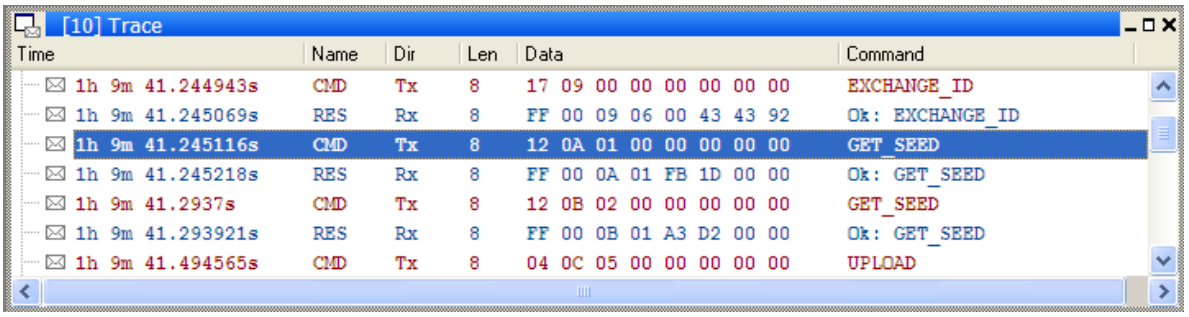
Reason 2:

A dependent system resource file could not been found on this computer. You could verify this by ope- ning the Seed & Key DLL file with a tool like DependencyWalker. Try to find the dependent system re- source files on your computer. In the following example Kernel32.DLL is dependent DLL files which must be found on the computer.

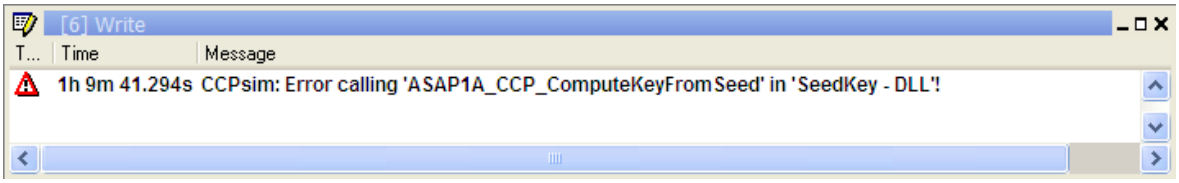


5.2 Error calling function in the CCP Seed & Key DLL file

A problem with the function in the Seed & Key DLL file will exist when the command GET_SEED will be send but the command UNLOCK is missing in the protocol trace.



A message in the write window will display that there is a problem with the function in the Seed & Key DLL file.

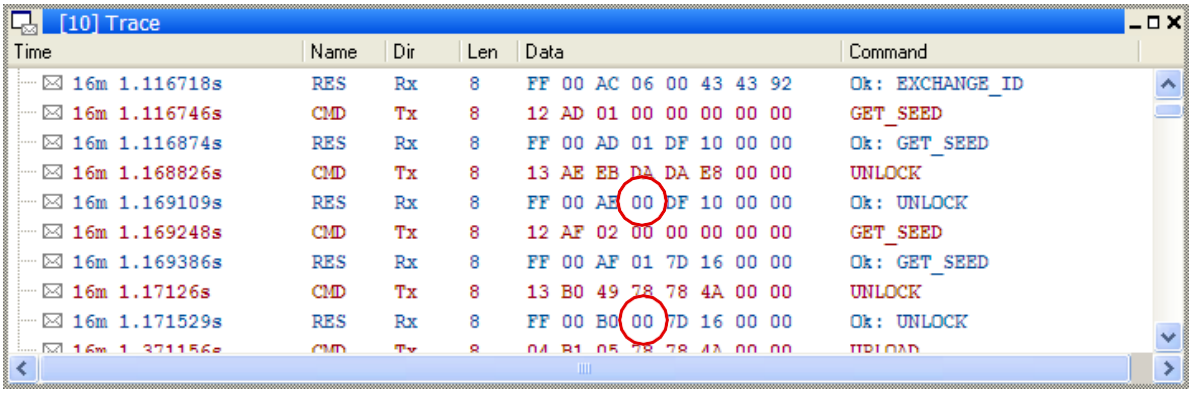


Reason 1:

The reason could be an implementation error in this function. Wrong or no return values available.

5.3 Unlocking of the resource failed

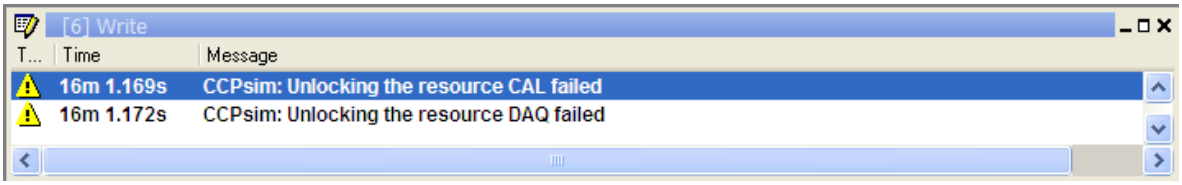
The response to the UNLOCK command from the ECU will inform about the Current Privilege Status (Resource Mask). In this example no resource will be unlocked.



Reason 1:

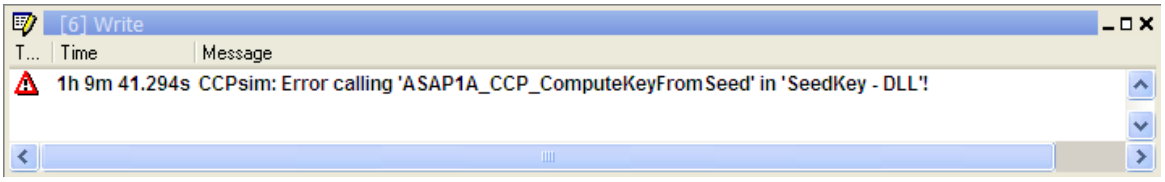
Wrong key in the UNLOCK command. Check the seed and the calculated key in the protocol trace. Per- haps the wrong Seed & Key DLL file will be included in this CANape project.

The following messages in the write window will occur.



5.4 Several functions in CCP Seed & Key DLL file

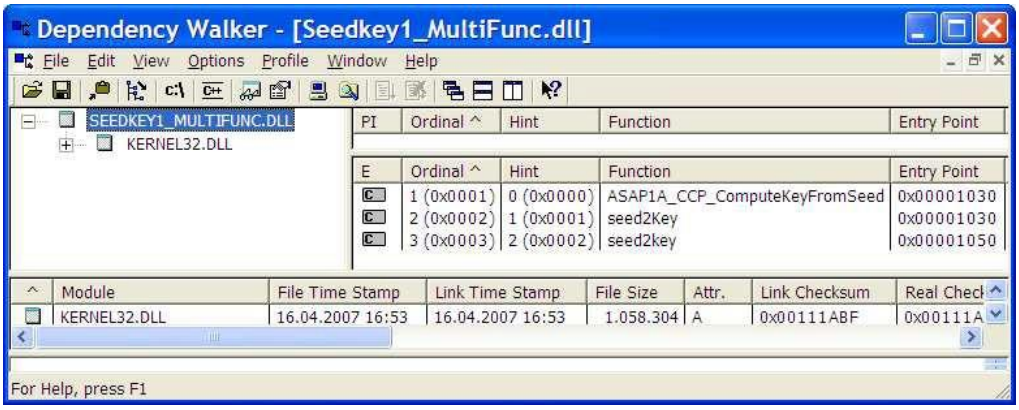
A message in the write window will display that there is a problem with the function in the Seed & Key DLL file. But this Seed & Key DLL file will be working with another measurement and calibration tool like CANape without a problem.



Reason 1:

More than one function will be defined in the Seed & Key DLL file. CANape will use the ASAP1A_CCP_ComputeKeyFromSeed function for example. The other measurement and calibration tool will use another function from this Seed & Key DLL file.

You could open the Seed & Key DLL file with a tool like DependencyWalker to verify if several functions will be defined.



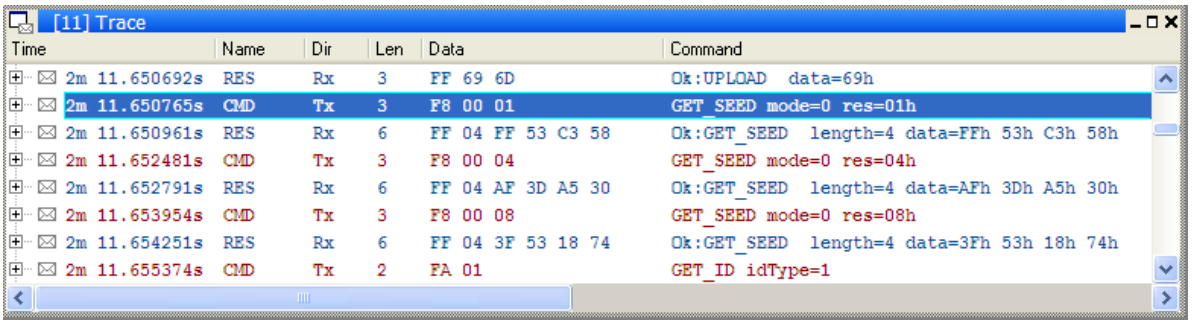
A wrapper DLL file could be used to switch between the internal functions in this Seed & Key DLL file.

6 Problems with XCP and Seed & Key:

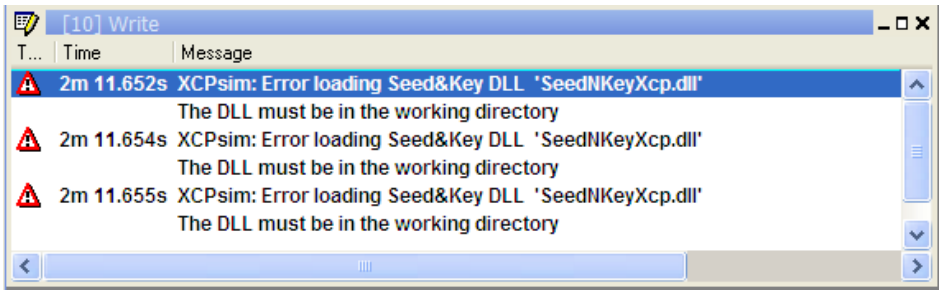
There could be different reasons why the Seed & Key functionality in XCP could fail. The following exam- ples try to show a way to fix some known problems which have been occurred with CANape in the past.

6.1 Error loading XCP Seed & Key DLL file

A problem with the Seed & Key DLL file will exist when the command GET_SEED will be send but the command UNLOCK is missing in the protocol trace.



A message in the write window displays that the Seed & Key DLL file could not been found.

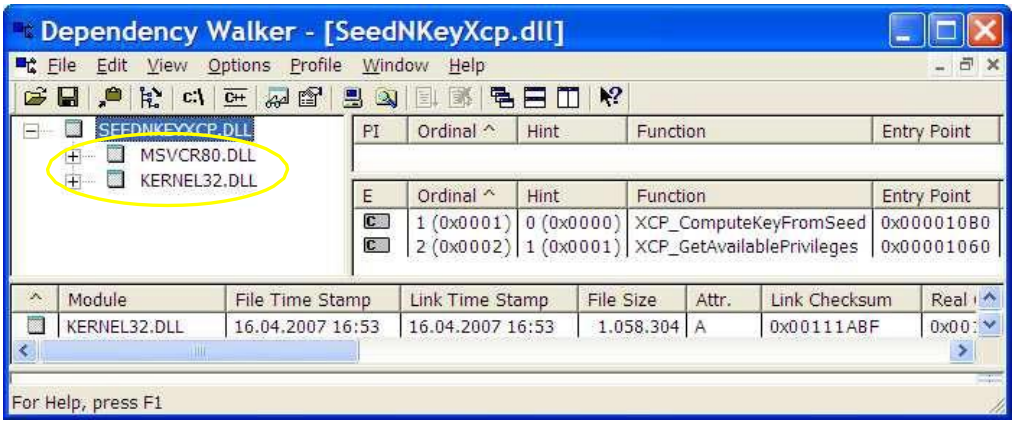


Reason 1:

File name or path of the Seed & Key DLL file is not correct. Please verify file name and path in the driver settings of the device. You could find more information in chapter 3.2 (Seed & Key configuration in CANape) of this document.

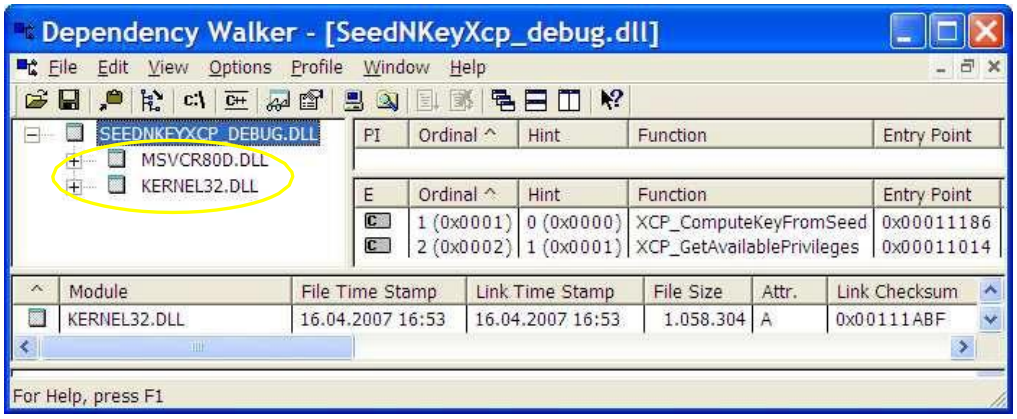
Reason 2:

A dependent system resource file could not been found on this computer. You could verify this by opening the Seed & Key DLL file with a tool like DependencyWalker. Try to find the dependent system resource files on your computer. In the following example MSVCR80.DLL and Kernel32.DLL are dependent DLL files which must be found on the computer.



Reason 3:

The Seed & Key DLL file will be a debug file. The dependent MSVCR80D.DLL file must be installed on your computer. Normally the debug resource files are only installed on computers with an installed soft- ware environment tool like Microsoft Visual Studio. The size of this debug Seed & Key DLL file will be greater than a normal Seed & Key DLL files.

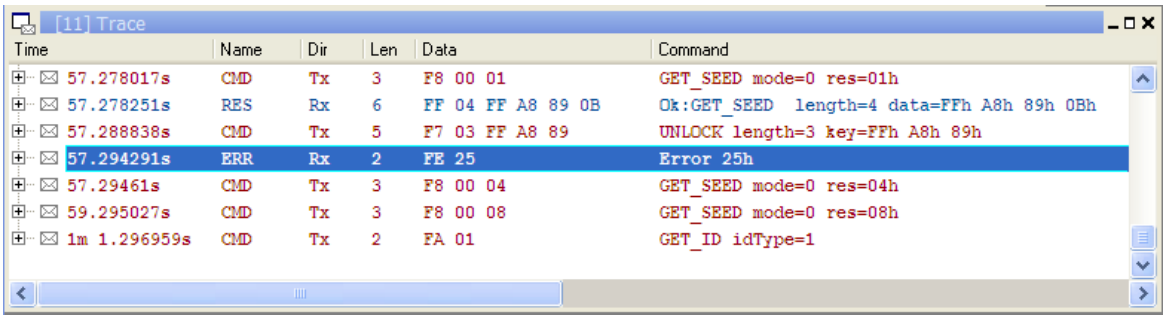


Reason 4:

The Seed & Key DLL file will not return with the correct number of information to CANape. An implemen- tation error of the internal function could be the reason. When the Seed & Key DLL file will be crashed, it is possible that CANape will also be crashed. This could not been handled by CANape.

6.2 No Access message from ECU

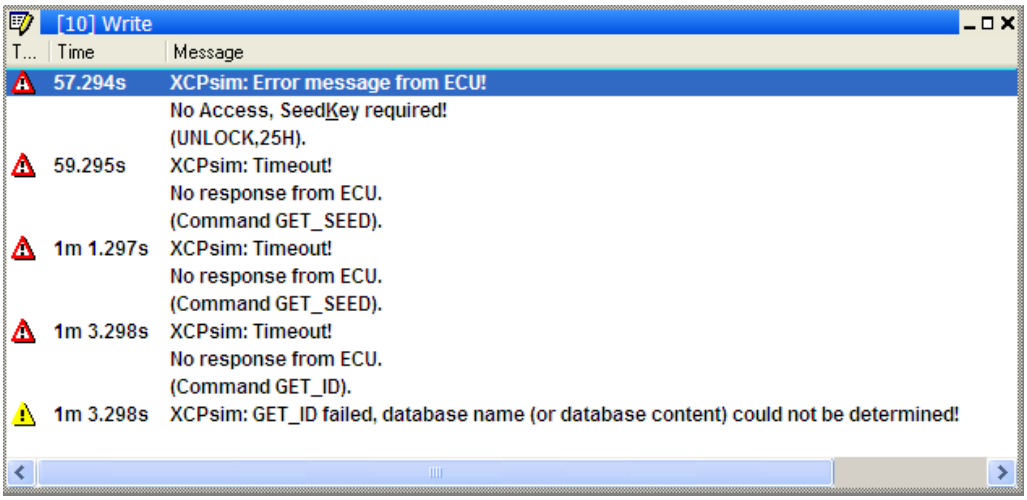
In this example the ECU will send an error response for the UNLOCK command.



Reason 1:

Wrong key in the UNLOCK command. Check the seed and the calculated key in the protocol trace. Per- haps the wrong Seed & Key DLL file will be included in this CANape project.

Only the first message in the write window will be the correct reason why the connection in this example fails. The other messages are only subsequent errors.



7 Seed & Key for Logger

The XCP or CCP communication between Vector loggers and devices could be Seed & Key protected. Loggers are not able to use a Seed & Key DLL file to unlock such devices. Because of this a specific function is needed. This chapter will describe the Vector logger configuration for Seed & Key protected devices.

For the measurement of CCP or XCP data with a Vector logger on a Seed & Key protected device the following files are needed:

- DBC or Fibex file with CCP/XCP information
- SKB file with information of the Seed & Key algorithm

Both files could be created within CANape and afterwards added to the logger configuration. During the compilation of the logger configuration all the information is bounded the the created COD file. This COD file could be loaded to the Vector logger. Starting the logger the connection to the Seed & Key protected device will be established automatically via CCP or XCP.

7.1 Supported logger

The following Vector loggers are supporting Seed & Key:

- GL1000
- Multilog

Please be sure to have a CCP or XCP license for each logger. This license will include the support of CCP, XCP and Seed & Key.

7.2 The creation of DBC files (for CAN)

Two steps to create a DBC file with CANape 8.0:

1. Please select all the signals you want to log in the measurement configuration (**F4**) in CANape. The measurement mode has to be cyclic, polling mode is not allowed.

1. Please open the Logger configuration dialog via the menue '**Tools |Logger configuration…**' and select one device before creating the corresponding DBC file.

CANape will record all the information from the current connection to the device and save this to the created DBC file. The DBC file is saved in the current working directory of CANape. The CCP or XCP connect information in the DBC fiel is exactly the same like in CANape. Because of this the Vector logger will later send the same protocoll commands to the device like CANape.

7.3 The creation of Fibex files (for XCP on FlexRay)

Five steps to create a Fibex file with CANape 8.0:

1. Please select all the signals you want to log in the measurement configuration (**F4**) in CANape. The measure- ment mode has to be cyclic, polling mode is not allowed
2. Connect CANape to the Device (menue **Calibration | Go Online**).
3. Starting the measurement for a few seconds (menue **Measurement | Start** or **F9**).
4. Stop the measurement in CANape. (menue **Measurement | Stop** or **ESC**).
5. Please open the Logger configuration dialog via the menue '**Tools |Logger configuration…**' and select one device before creating the corresponding DBC file.

CANape will record all the information from the current connection to the device and save this to the created Fibex file. The Fibex file is saved in the current working directory of CANape. The XCP connect information in the DBC fiel is exactly the same like in CANape. Because of this the Vector logger will later send the same protocoll commands to the device like CANape.

Additionally the created Fibex file will include also the information from the source Fibex file included in CANape as network description file of the device.

7.4 The creation of SKB files (for Seed & Key)

The created DBC or Fibex will include enough information for a logger configuration for devices without Seed & Key protection. If the device is Seed & Key protected a SKB file with the Seed & Key algorithm is also needed for the logger configuration. The SKB file could be created in CANape like the following:

Start CANape:

- If a CANape project for the connected ECU exists, open this project.
- If no CANape project exists yet, create a new project based on the ECUs A2L file.

Open the Functions Editor via Tools|Functions and scripts… or the appropriate button from the toolbar

Create a new function in the section Seed & Key algorithm; select New… from the shortcut menu. The function is stored to an SKS file.

When the function is created, the following four parameters are automatically included:

- var seed[]

Key from ECU

- var seedLength

Length of the key from ECU

- var key[]

Response key to ECU

- var keyLength

Length of response key to ECU

1. Insert the ECU specific Seed & Key algorithm.
2. Select Export… from the shortcut menu of the SKS file.
3. Click on [SeedKey Binary (.skb)].
4. Select an export directory, where the SKB file is created.

Note
The SKS files contain the encryption algorithms in plain text and will always be saved in the sub-folder **SeedAndKey** in the CANape working directory. After the SKB file was created the re- spective SKS file can be deleted or moved for safety reasons. Otherwise it will be exported, too, when the project is exported by the project manager.

8 Contacts

For support related questions please address to the support contact for your country https://www.vector.com/int/en/company/contacts/support-contact/.