Security Access for CANoe

2014-09-22

SN-IND-1-003_SecurityAccess_CANoe.pdf

Author(s) Morizur, Pascale; Ernst, Oliver
Restrictions

Table of contents

1 About this Support Note 1
2 Overview 2
3 Seed&Key functionality 2
3.1 CANoe/DiVa Configuration 2
3.2 Use of Security DLL in CAPL 3
3.3 CANoe version up to 7.0 3
3.4 CANoe version 7.1 and later 4
4 Fingerprint functionality 6
5 DiVa configuration 6
6 Contacts 6

1 About this Support Note

In the table below you will find the icon conventions used throughout the Support Note.

Symbol	Utilization
i	This icon indicates notes and tips that facilitate your work.
⚠	This icon warns of dangers that could lead to damage.
▲	This icon indicates examples.

2 Overview

This Support Note gives you an overview how to handle Security Access in diagnostic projects based on CANoe.

Security Access might require, depending on the OEM and the ECU, two steps for setting the following functionality:

- Seed & Key: To access the ECU using an ECU specific DLL. This DLL calculates the Key in dependence of the Seed sent by the ECU. Alternatively for testing purpose on CANoe without DiVa, the DLL can be replaced by a CAPL code.
- Fingerprint: To be written in the ECU to document the name of the instance that had access to the ECU.

All folder paths in this document are described for Windows 7. If you use another OS, like Windows XP for instance, the path might be slightly different.

3 Seed&Key functionality

Depending on the OEM, there are two types of Seed & Key accesses:

- Simple access
- Level based access

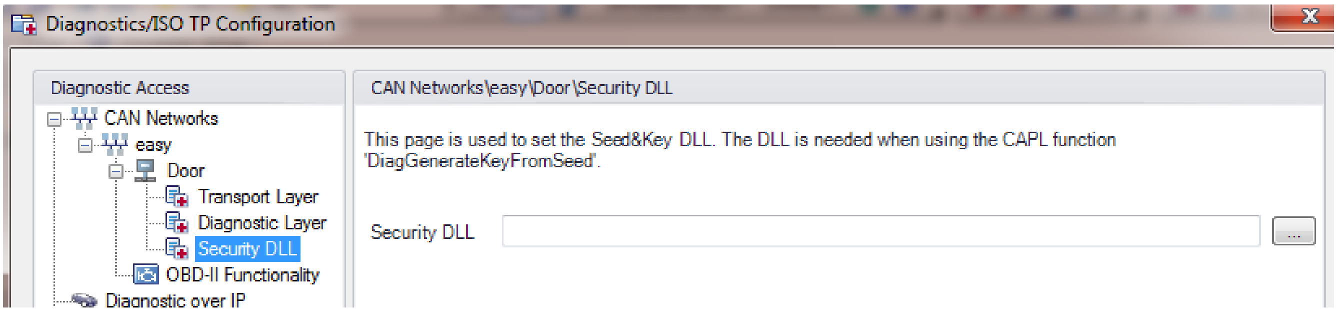
Most of the OEMs use a simple Security Access.

Please contact us in case you need detailed information how to handle your OEM’s specific security access requirements (e.g. for VAG or GM) in CANoe or DiVa.

3.1 CANoe/DiVa Configuration

Each ECU from a network has its own Security Access. The SeedKey DLL / ECU assignment shall be done in CANoe for each ECU using the “Diagnostic/ISO TP Configuration” menu under “Security DLL”, like indicated in the picture below for the “Door” ECU.

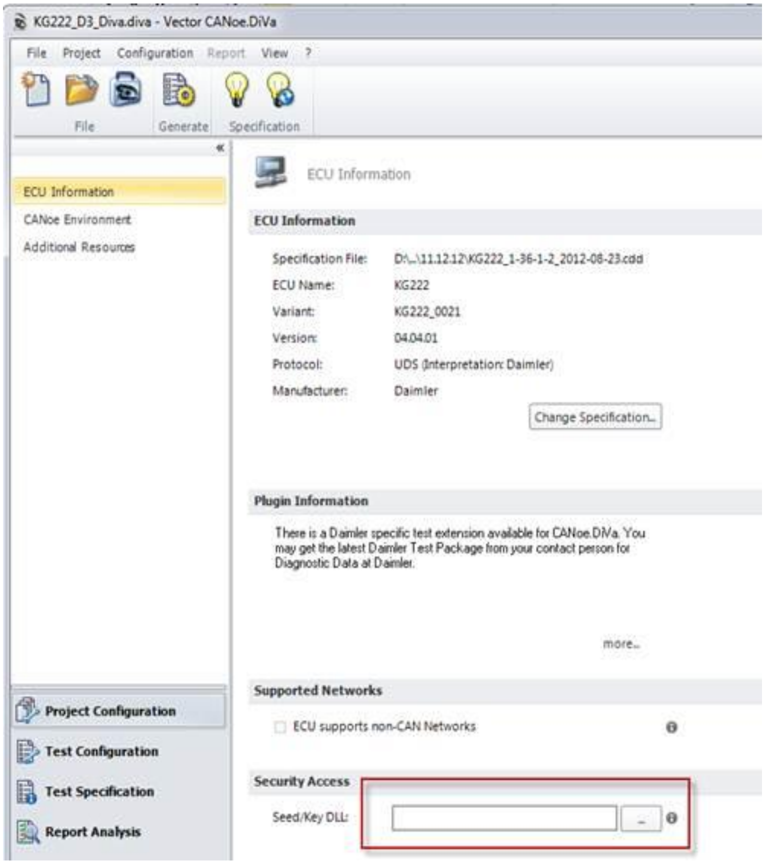
Please note that if you use DiVa, the SeedKey DLL in the “Diagnostic/ISO TP Configuration” will be automatically set as soon as the DiVa project is imported.



This page is used to set the Seed&Key DLL. The DLL is needed when using the CAPL function 'DiagGenerateKeyFromSeed'.

Security DLL

Please note that if you use DiVa, the SeedKey DLL shall only be configured in DiVa, like indicated below.



ECU Information

ECU Information

Specification File: D:\...\11.12.12\KG222_1-36-1-2_2012-08-23.cdd
ECU Name: KG222
Variant: KG222_0021
Version: 04.04.01
Protocol: UDS (Interpretation: Daimler)
Manufacturer: Daimler

Plugin Information

There is a Daimler specific test extension available for CANoe.DiVa. You may get the latest Daimler Test Package from your contact person for Diagnostic Data at Daimler.

Supported Networks

□ ECU supports non-CAN Networks

Security Access

Seed/Key DLL:

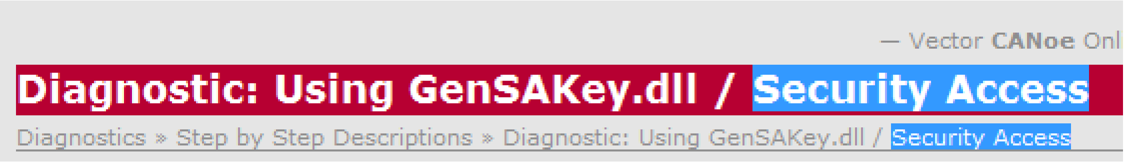
3.2 Use of Security DLL in CAPL

In CAPL, the Security DLL of the ECU that has been last set to active using `DiagSetTarget` - for instance `DiagSetTarget(„Door“)` - is used.

3.3 CANoe version up to 7.0

Previous CANoe versions of maximal 7.0 use GenSAKey.dll. This wrapper-DLL calculates the Key the ECU is expecting depending on the received seed.

More information on Security Access, an example and a link to the GenSAKey.dll manual can be obtained using CANoe Help:



— Vector CANoe Onl

Diagnostic: Using GenSAKey.dll / Security Access

Diagnostics » Step by Step Descriptions » Diagnostic: Using GenSAKey.dll / Security Access

3.4 CANoe version 7.1 and later

Starting with CANoe7.1 it is possible to use a self-made Security DLL without GenSAKey.dll. The access is assured through new C++ functions that are mandatory in your SeedKey DLL.

Please note that if the Security Access is not successful, the Key “0x00” will be transmitted by CANoe. You can observe this in the trace window. In this case you should check your SeedKey DLL.

More information on Security Access with SeedKey DLL might be obtained using CANoe Help.

There are different functions (APIs) that might be implemented by the SeedKey DLL to integrate the OEM Seed & Key algorithm. We recommend the use of one of the two following:

- `GenerateKeyEx`
- `GenerateKeyExOpt`

Both only differ in the parameter `ipOptions`, which is only part of `GeneratekeyExOpt`. This parameter enables the access to different Security Levels in case of Level based Security Access (see also 3). However if your OEM uses a Simple Security Access, you might also use the `GeneratekeyExOpt` API and set the `ipOptions` parameter to an empty or dummy String.

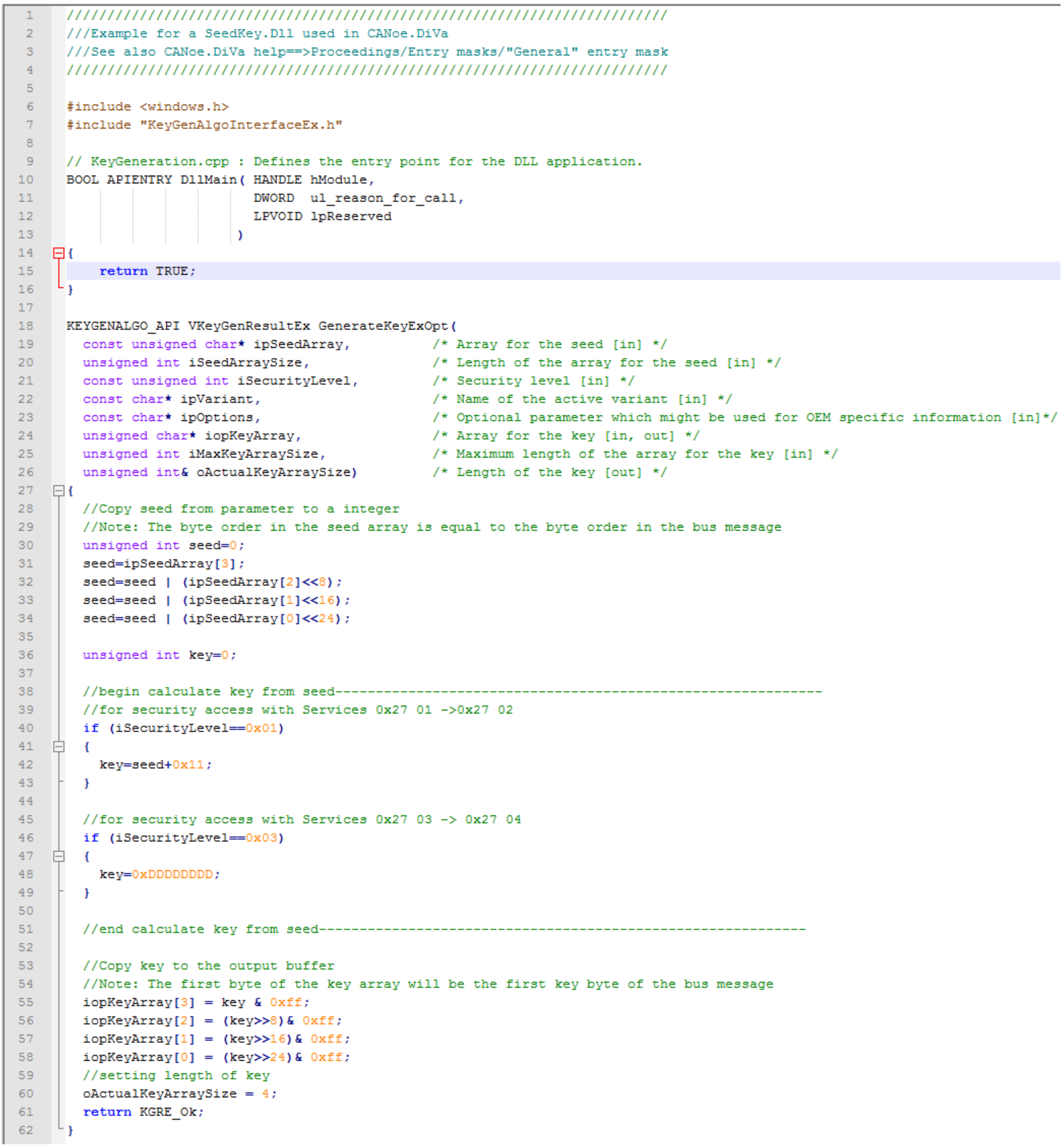
The CAPL-Function `DiagGenerateKeyFromSeed` automatically adapts itself to the API implemented in the SeedKey DLL. If your SeedKey DLL uses `GenerateKeyEx`, the CAPL-function `DiagGenerateKeyFromSeed` shall be used with the `ipOtpions`-Parameter set to an empty or dummy String.

It is possible to check which API is contained in your DLL using for instance the following free of charge tool:
http://en.wikipedia.org/wiki/Dependency_Walker

You can find an example how to use GenerateKeyEx in the form of a complete Visual Studio Project:

`\Demo_CAN_CN\Diagnostics\UDSSim\SeedKey`

This is an example how to use `GenerateKeyExOpt`:



```
//////////////////////////////////////////////////////////////////////////
///Example for a SeedKey.Dll used in CANoe.DiVa
///See also CANoe.DiVa help==>Proceedings/Entry masks/"General" entry mask
//////////////////////////////////////////////////////////////////////////

#include <windows.h>
#include "KeyGenAlgoInterfaceEx.h"

// KeyGeneration.cpp : Defines the entry point for the DLL application.
BOOL APIENTRY DllMain( HANDLE hModule,
                       DWORD  ul_reason_for_call,
                       LPVOID lpReserved
                     )
{
    return TRUE;
}

KEYGENALGO_API VKeyGenResultEx GenerateKeyExOpt(
  const unsigned char* ipSeedArray,          /* Array for the seed [in] */
  unsigned int iSeedArraySize,               /* Length of the array for the seed [in] */
  const unsigned int iSecurityLevel,         /* Security level [in] */
  const char* ipVariant,                     /* Name of the active variant [in] */
  const char* ipOptions,                     /* Optional parameter which might be used for OEM specific information [in]*/
  unsigned char* iopKeyArray,                /* Array for the key [in, out] */
  unsigned int iMaxKeyArraySize,             /* Maximum length of the array for the key [in] */
  unsigned int& oActualKeyArraySize)         /* Length of the key [out] */
{
  //Copy seed from parameter to a integer
  //Note: The byte order in the seed array is equal to the byte order in the bus message
  unsigned int seed=0;
  seed=ipSeedArray[3];
  seed=seed | (ipSeedArray[2]<<8);
  seed=seed | (ipSeedArray[1]<<16);
  seed=seed | (ipSeedArray[0]<<24);

  unsigned int key=0;

  //begin calculate key from seed------------------------------------------------------------
  //for security access with Services 0x27 01 ->0x27 02
  if (iSecurityLevel==0x01)
  {
    key=seed+0x11;
  }

  //for security access with Services 0x27 03 -> 0x27 04
  if (iSecurityLevel==0x03)
  {
    key=0xDDDDDDDD;
  }

  //end calculate key from seed------------------------------------------------------------

  //Copy key to the output buffer
  //Note: The first byte of the key array will be the first key byte of the bus message
  iopKeyArray[3] = key & 0xff;
  iopKeyArray[2] = (key>>8)& 0xff;
  iopKeyArray[1] = (key>>16)& 0xff;
  iopKeyArray[0] = (key>>24)& 0xff;
  //setting length of key
  oActualKeyArraySize = 4;
  return KGRE_Ok;
}
```

4 Fingerprint functionality

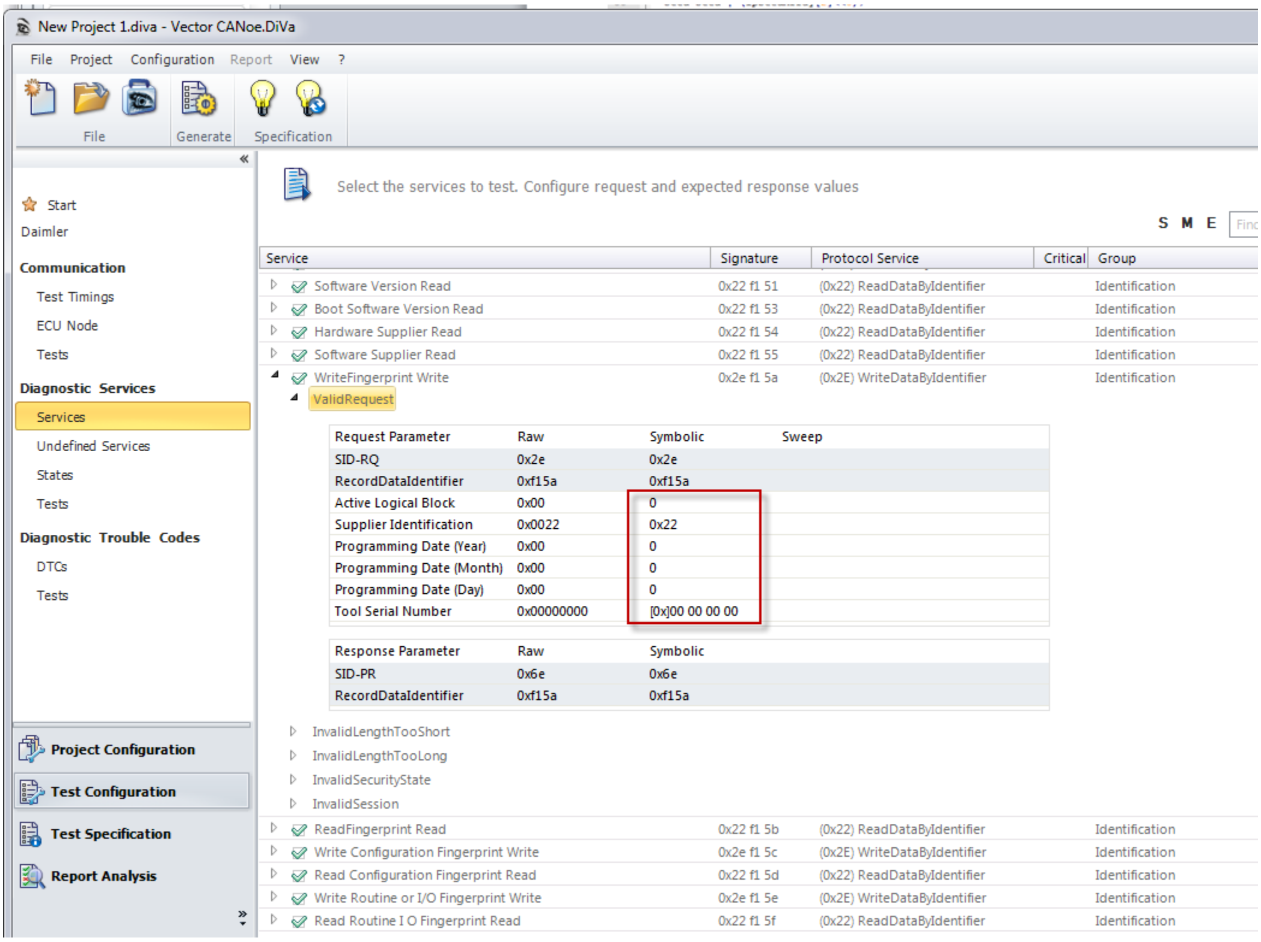
Once the Key has been accepted, some OEMs require additionally that a Fingerprint is written in the ECU to document who had access to the ECU memory before the Security Access is successful and the ECU is unlocked.

Fingerprint is written using a dedicated Diagnostic Instance based on the UDS-Service `WriteDataByIdentifier (0x2E)`. This Diagnostic Instance is part of the CDD describing the concerned ECU.

Some OEMs have Security Level dependent Fingerprints. In this case the CDD contains accordingly different Diagnostic Instances for Fingerprint purposes.

5 DiVa configuration

As the ECU is waiting for a successful Fingerprint before the ECU is unlocked, DiVa shall be manually configured to send a proper Fingerprint during the tests. This can be done using the right mouse button as indicated in the picture below:



Select the services to test. Configure request and expected response values

Service	Signature	Protocol Service	Critical	Group
Software Version Read	0x22 f1 51	(0x22) ReadDataByIdentifier		Identification
Boot Software Version Read	0x22 f1 53	(0x22) ReadDataByIdentifier		Identification
Hardware Supplier Read	0x22 f1 54	(0x22) ReadDataByIdentifier		Identification
Software Supplier Read	0x22 f1 55	(0x22) ReadDataByIdentifier		Identification
WriteFingerprint Write	0x2e f1 5a	(0x2E) WriteDataByIdentifier		Identification

ValidRequest

Request Parameter	Raw	Symbolic	Sweep
SID-RQ	0x2e	0x2e	
RecordDataIdentifier	0xf15a	0xf15a	
Active Logical Block	0x00	0	
Supplier Identification	0x0022	0x22	
Programming Date (Year)	0x00	0	
Programming Date (Month)	0x00	0	
Programming Date (Day)	0x00	0	
Tool Serial Number	0x00000000	[0x]00 00 00 00	

Response Parameter	Raw	Symbolic
SID-PR	0x6e	0x6e
RecordDataIdentifier	0xf15a	0xf15a

InvalidLengthTooShort

InvalidLengthTooLong

InvalidSecurityState

InvalidSession

Service	Signature	Protocol Service	Critical	Group
ReadFingerprint Read	0x22 f1 5b	(0x22) ReadDataByIdentifier		Identification
Write Configuration Fingerprint Write	0x2e f1 5c	(0x2E) WriteDataByIdentifier		Identification
Read Configuration Fingerprint Read	0x22 f1 5d	(0x22) ReadDataByIdentifier		Identification
Write Routine or I/O Fingerprint Write	0x2e f1 5e	(0x2E) WriteDataByIdentifier		Identification
Read Routine I O Fingerprint Read	0x22 f1 5f	(0x22) ReadDataByIdentifier		Identification

6 Contacts

Please find the contacts of Vector Informatik GmbH and all subsidiaries worldwide via:
http://www.vector.com/vi_addresses_en.html