UDS 安全访问授权
Security Access Authorization for UDS

编号 Document reference: NIO-TP.20.020-2019

文件名 File name: NIO-TP.20.020-2019_Security Access Authorization_v3.0.docx

版本 Version: 3.0

状态 Status: Release

日期 Date: 5/5/2019

	姓名 Name	部门 Dept.	日期 Date	签字 Signature
起草 Created:	Jun LUO	VE/EES	2016/06/10	
校对 Approved:	Peng Yu	VE/EES	2016/08/20	
	Chao Liang	VE/EES	2016/08/27	

保密条款 Confidentiality

Change History

Version	Date	Author	Changes	Distribution
V1.0	2016/06/10	Jun LUO	create	
V3.0	2019/05/05	Shanfei Zou	Update spec reference	

All update from last version to this version is in yellow.

Table of Contents

1 Introduction

1.1 Scope of this document

The goal of this document is to define the security access algorithm, and the used services, security levels implemented in both flash bootloader(level 4) and application software (level 1~3) by Electronic Control Units (ECUs) and is intended for use by electrical/electronic (E/E) engineering development teams and corporate suppliers in the design of Electronic Control Units designed for use in all NIO applications when Unified Diagnostic Service (UDS) diagnostic protocol is implemented, whatever the implemented physical bus type, e.g. CAN, LIN, Ethernet.

1.2 Order of Precedence

In case of a conflict between requirements stated in this document and other documents referenced and/or provided, the supplier is asked to request a NIO decision which requirement is valid. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies. In case the supplier fails to do so, NIO may decide which requirement is valid at any time during development

Nothing in this specification, however, supersedes applicable laws and regulations unless a specific exemption has been obtained.

1.3 Limitations

The document shall only be released and used by ECU support UDS diagnostic.

1.4 Document Conventions

In this document, the following terminology applies:

- “Shall” expresses an obligatory/mandatory requirement
- “Should” expresses a recommendation or an advice
- “Must” expresses a legal or normative requirement
- “Will” expresses a precautionary consideration or an additional/optional feature
- “May” expresses a permitted practice/method, not to be considered as a requirement

In this document, the following conventions are used for some parameters:

- M (“mandatory”): The parameter has to be supported.
- C (“conditional”): The parameter may be supported, based on certain criteria (e.g. function based in the ECU), need checking the detail criteria defined.

U (“user optional”): The parameter may or may not be present, depending on dynamic usage of the user.

2 Security Mechanism

Considering security, emissions, or safety reasons, there are restrictions for access data and/or diagnostic services (eg. Diagnostic services for downloading/uploading routines or writing data into a server and reading specific memory locations from a server are situations where security access may be required.). Improper routines or data downloaded into a server could potentially damage the ECUs or other vehicle components or risk the vehicle's compliance to emission, safety, or security standards. As defined in [HIS Security] the seed and key algorithm to be used for unlocking the ECU before performing restricted diagnostic action. Refer to NIO-TP.20.013-2019_General UDS Diagnostic Specification for detailed information for security access mechanism.

2.1 Security Level for ECU Access

Refer to NIO-TP.20.013-2019_General UDS Diagnostic Specification for ECU Security Access Level Concept. Regarding the different security/safety requirements, different Security Access Levels are defined.

At NIO the following predefined Security Access Levels are listed:

SA Level Level	Description/ Use Case	Engineering Development	Manufacturing/EOL	Service	Others/ Non-OEM
0	No Security Access (ECU unlock) required	X	X	X	
1	Basic Protected Services	X	X	X	
2	Higher level protected services for EOL procedure/ECU replacement (e.g. safety, security related)	X	X	X	
3	Special Vehicle Assembly Plant Functions (Optional for HV system)	X	X		
4	ECU Flash reprogramming	X	X	X	

Note: The UDS defines that only one security access level is active in the ECU at a time.

The SEED & Key length shall follow below rule:

Security Level	SEED length (Bytes)	KEY length (Bytes)	Used Service
1	4	4	27 01 / 27 02
2	4	4	27 03 / 27 04
3	4	4	27 05 / 27 06
4	4	4	27 07 / 27 08

2.2 NIO Standard Security Access Algorithm

As defined in [HIS Security] the seed and key algorithm to be used for unlocking the ECU to get the authorization of different Security Access Levels. Regarding different Security Levels, different algorithm is proposed for this purpose.

NIO shall distribute a secret code with 16 bits length for each security level of a specific ECU. That means one ECU shall get three (or four, if security level 3 is needed) secret codes from NIO. ECU supplier shall hardcoded these codes into Flash Memory together with the FBL SW. And the code for specific security level together with the seed along with diagnostic service ($27 $01/$27 $03/$27 $05/$27 07) sent by diagnostic tool shall be used to calculate the key for unlocking this security level. NIO diagnostic team shall define the rule of delivering and managing secret codes and be responsible for delivering secret codes to ECU suppliers according their actual development phase.

e.g. ECU1 has three security levels (level1, level2, level4). NIO dedicates secret code 0x1234 (high byte is 0x12,low byte is 0x34) to ECU1 for security level1, secret code 0x2345 (high byte is 0x23 and low byte is 0x45) to ECU1 for security level2 and secret code 0x3456 (high byte is 0x34 and low byte is 0x56) to ECU1 for security level4.

The algorithm for Security Access Level#**n** (where n = 1, 2, 3, 4) with secret code (code_level**n**):

1. As a response to **SecurityAccess.RequestSeed ($27 $xx** (where xx = 1, 3, 5, 7)**)** the ECU transmits 4 bytes seed randomly to the tester:

   [<byte1>,<byte2>,<byte3>,<byte4>]

2. Combine the 4 bytes seed and the ECU secret identifier as [<$byte_1$>, <$byte_2$>, <$byte_3$>, <$byte_4$>, < code_level**n**_high_byte>, < code_level**n**_low_byte>]
3. The tester (and ECU) have to use the CRC algorithm defined in [SAE-J1850] to calculate the CRC 8 value of [<$byte_1$>,<$byte_2$>,<$byte_3$>,<$byte_4$>, < code_level**n**_high_byte>, < code_level**n**_low_byte>] as <crc_1>
4. The tester (and ECU) has to calculate the CRC8 value of

   [<crc_1>,<$byte_2$>,<$byte_3$>,<$byte_4$>,<code_level**n**_high_byte>,<code_level**n**_low_byte>] as <crc_2>
   [<$byte_1$>,<crc_2>,<$byte_3$>,<$byte_4$>,<code_level**n**_high_byte>,<code_level**n**_low_byte>] as <crc_3>
   [<$byte_1$>,<$byte_2$>,<crc_3>,<$byte_4$>,<code_level**n**_high_byte>,<code_level**n**_low_byte>] as <crc_4>
   [<$byte_1$>,<$byte_2$>,<$byte_3$>,<crc_4>,<code_level**n**_high_byte>,<code_level**n**_low_byte>] as <crc_5>
   [<$byte_1$>,< $byte_2$>,< $byte_3$>,<$byte_4$>,< crc_5>,<code_level1_low_byte>] as <crc_6>
   [<$byte_1$>,< $byte_2$>,< $byte_3$>,<$byte_4$>,< code_level**n**_high_byte >,< crc_6] as <crc_7>.

   in iterations with the CRC algorithm defined by [SAE-J1850].
5. If <crc_4> == 0 && <crc_5> == 0 && <crc_6> == 0 && <crc_7> == 0, then the tester (and ECU) shall use [<crc_2>,< crc_3>,< crc_4>,< crc_5>] as calculated key, else the tester (and ECU) shall use [<crc_4>,< crc_5>,< crc_6>,< crc_7>] as calculated key.
6. The calculated key is transmitted to the ECU by **SecurityAccess.Sendkey($27 $xx** (where xx = 2, 4, 6, 8) **)**.
7. Only if the key is calculated identically and correctly on both sides, the ECU is unlocked until it leaves current session. The ECU shall confirm this state by sending a positive response.

e.g. secret code 0x1234.

Seed: 0xFF,0xFF,0xFF,0xFF *Key:0x64,0xF2,0xFC,0x77*

Seed: 0xA3,0xB2,0x18,0x75 *Key:0xA5,0xBE,0xCD,0x8C*

2.3 All Zero Seed Avoidance

In the software, ECU supplier shall make the code to avoid all 0x00 seed generated. That means the ECU software shall re-generate a set of seeds when the random generated seeds are all zero.

(Because in ISO standard, all zero seed response by ECU means the ECU have already been unlocked in the requested security level)

For verification, supplier shall show this part of software to NIO for checking this function implementation.

3 Delay timer handing

After three failure attempts of a "SecurityAccess sendKey request" in a row, the ECU shall be locked for "SecurityAccess" for 10 seconds. Any SecurityAccess request during this time shall be rejected with the negative response code "Required time delay not expired" (NRC 37).
When the 10s wait time is elapsed, another try is allowed. In case this try is invalid, the ECU shall wait again 10s before accepting another "Request Seed" request
Note: see FAA-Flag handling of "Implementation Rules" in Section3.7.3 in < NIO-TP.20.013-2019_General UDS Diagnostic Specification> for details.

4 Implementation

The security access Level1, Level2 shall be implemented in Application and only security level4 shall be implemented in Flash bootloader software. And the ECUs which have special vehicle assembly plant functions shall implement security Level3 in Application.

5 Appendices

5.1 Glossary

Acronym	Meaning
CAN	Controller Area Network
ECU	Electronic Control Unit
EE	Electrical Engineering
ISO	International Organization for Standardization
LIN	Local Interconnect Network
ECU	Electronic Control Unit
SA	Security Access ('None' means services no need unlock ECU)

5.2 Related Documents

No	Document name
[1]	NIO-TP.20.013-2019_General UDS Diagnostic Specification
[2]	NIO-TP.20.015-2019_Flash Programming Specification

6 Example code of CRC8 for Key Calculation

```
unsigned char crc8(unsigned char *data, int length) //function of calculate the Key
	{
		unsigned char t_crc;
		int f, b;
		t_crc = 0xFF;
		for (f = 0; f < length; f++)
		{
			t_crc ^= data[f];
			for (b = 0; b < 8; b++)
			{
				if ((t_crc & 0x80) != 0)
				{
					t_crc <<= 1;
					t_crc ^= 0x1D;
				}
				else
				{
					t_crc <<= 1;
				}
			}

		}
		return ~t_crc;
	}

  //-----------------------------------------------------------------------------
  //-----------------------------------------------------------------------------
bool __declspec(dllexport) __cdecl ASAP1A_CCP_ComputeKeyFromSeed(char *seed, unsigned
short sizeSeed, char * key, unsigned short maxSizeKey, unsigned short *sizeKey)
	{
		int seedlength = 6;
		unsigned char buf_byte[6];
		unsigned char crc_byte[7];
		seed[4] = 0x12;//high byte of secret code(example)
		seed[5] = 0x34;//low byte of secret code(example)

		buf_byte[0] = seed[0];
		buf_byte[1] = seed[1];
		buf_byte[2] = seed[2];
		buf_byte[3] = seed[3];
		buf_byte[4] = seed[4];
		buf_byte[5] = seed[5];

		crc_byte[0] = crc8(buf_byte,seedlength);
```

```
                buf_byte[0] = crc_byte[0];
                crc_byte[1] = crc8(buf_byte,seedlength);

                buf_byte[0] = seed[0];
                buf_byte[1] = crc_byte[1];
                crc_byte[2] = crc8(buf_byte, seedlength);

                buf_byte[1] = seed[1];
                buf_byte[2] = crc_byte[2];
                crc_byte[3] = crc8(buf_byte, seedlength);

                buf_byte[2] = seed[2];
                buf_byte[3] = crc_byte[3];
                crc_byte[4] = crc8(buf_byte, seedlength);

                buf_byte[3] = seed[3];
                buf_byte[4] = crc_byte[4];
                crc_byte[5] = crc8(buf_byte, seedlength);

                buf_byte[4] = seed[4];
                buf_byte[5] = crc_byte[5];
                crc_byte[6] = crc8(buf_byte, seedlength);

if (crc_byte[3] == 0 && crc_byte[4] == 0 && crc_byte[5] == 0 && crc_byte[6] == 0)
                {
                        key[0] = crc_byte[1];
                        key[1] = crc_byte[2];
                        key[2] = crc_byte[3];
                        key[3] = crc_byte[4];
                }else
                {
                        key[0] = crc_byte[3];
                        key[1] = crc_byte[4];
                        key[2] = crc_byte[5];
                        key[3] = crc_byte[6];
                }
                *sizeKey = 4;
                // If the return value is false the flash tool stops
                return true;
}
```