AN14060

How to use the PN76 family cryptographic features

Rev. 1.0 — 14 September 2023 | Application note

Document information

Information	Content
Keywords	Cryptography, MbedTLS, AES, GCM, PN7642, KeyStore
Abstract	Within this application note it is shown how to use the internally stored keys, in the PN7642 secure key store, with the mbedTLS APIs.

Revision history

Revision history

Rev	Date	Description
v.1.0	20230914	Initial version

1 Introduction

This document shows how to use the PN7642 cryptographic features, particularly AES, by using the provided SDK examples.

Within this document, only keys from the PN76s key store are used. The mbedTLS can also be used without using the internal keys of the key store, but this is considered to be straight-forward and not covered in here. Make sure to have the key store provisioned with at least one APP_MASTER_KEY or APP_FIXED_KEY, as the APP_ROOT_KEY cannot be used for cryptographic operations.

How to provision the key store is not covered in this document, and explained in another application note: [1] AN13720

The PN7642s cryptographic features are abstracted by ROM APIs (mbedTLS nsc API). All these APIs are described in the [2] PN7642 NFC Controller User API Documentation. On top of these APIs, NXP has created an abstraction layer (mbedTLS Crypto) which wraps these APIs to make them MbedTLS compatible.

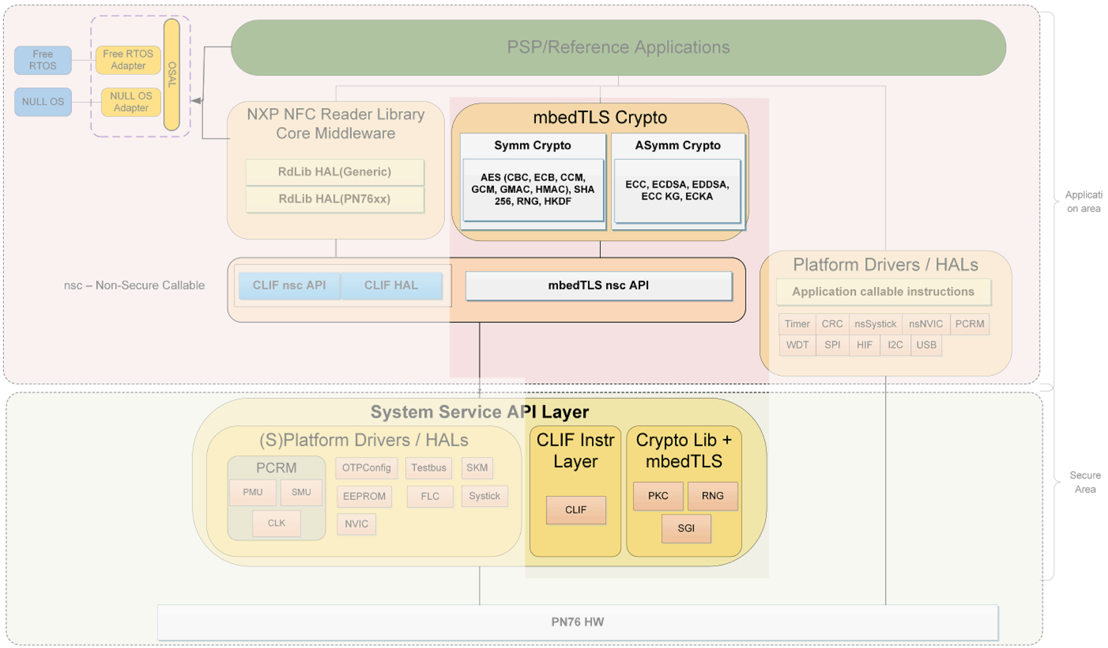

Figure 1. PN7642 system software architecture

Where to find the crypto interfaces in the [2] PN7642 NFC Controller User API Documentation:

Figure 2. PN7642 User API documentation crypto interfaces

2 General usage

2.1 Initialization of modules

Before the crypto modules and the key store can be used, they have to be initialized. Initialization is done by the following APIs. The codeblocks show a snip of the "pn_mbedtls_demo" example.

Both methods calls must be successful for further operation.

API: phmbedcrypto_Init()

```
/*Initialize the crypto modules */
InitStatus = (PN76_Status_t)phmbedcrypto_Init();
if (InitStatus != PN76_STATUS_SUCCESS)
{
    PRINTF("Crypto initialization failure\r\n");
    while (1)
        ;
}
```

API: PN76_Sys_KeyStore_Init(&bKeyStoreStatus)

```
eKeyStoreStatus = PN76_Sys_KeyStore_Init(&bKeyStoreStatus);
/* bKeyStoreStatus 6th bit means fatal error. */
if ((eKeyStoreStatus != PN76_STATUS_SUCCESS) || ((bKeyStoreStatus & 0x40U) !=
 0U))
{
    PRINTF("Crypto initialization error\r\n");
    while (1)
        ; /* if Failed Do not go further */
}
```

2.2 Context initialization

Before any crypto operation can be used, the context has to be initialized.

```
/*!
 * @brief          This function initializes the specified AES context.
 *
 *                 It must be the first API called before using
 *                 the context.
 * \note           Only 1 AES context is supported at any time. Use mbedtls_aes_free/
 *                 mbedtls_ccm_free before initializing the next context.
 * \param ctx      The AES context to initialize. This must not be \c NULL.
 */
void mbedtls_aes_init( mbedtls_aes_context *ctx );
```

Figure 3. mbedTLS AES Init

The aes context as a member "key_index", which is very important, if a key of the key store shall be used. This variable, key_index, has to be set to the "KeyIndex" of the key to be used.

Default PN76xx KeyStore map The diagram below shows the keystore map on PN76xx device, after APP_ROOT_KEY (both 128-bit and 256-bit) are provisioned.

KeyIndex	KEY Data	Description	
00	APP_ROOT_KEY_128	128-bit APP_ROOT_KEY data	Keys stored in PN76xx Secure Key Store enclave. For storing 256-bit key, two KeyIndexes are used and must provide even numbered KeyIndex
01		Free for APP_MASTER_KEY	
02	APP_ROOT_KEY_256	256-bit APP_ROOT_KEY data	
03	APP_ROOT_KEY_256	256-bit APP_ROOT_KEY data	
04	:	Free for APP_MASTER_KEY	
05	:	Free for APP_MASTER_KEY	
:	:	Free for APP_MASTER_KEY	
:	:	Free for APP_MASTER_KEY	
:	:	Free for APP_MASTER_KEY	
15	:	Free for APP_MASTER_KEY	
16	:	Free for APP_FIXED_KEY	Encrypted fixed keys stored in PN76xx Secure Flash. 128-bit or 256-bit key can be stored in one KeyIndex.
17	:	Free for APP_FIXED_KEY	
:	:	Free for APP_FIXED_KEY	
26	:	Free for APP_FIXED_KEY	
27	:	Free for APP_ASYMM_KEY	Encrypted Asymmetric keys stored in PN76xx Secure Flash. 256-bit or 384-bit key can be stored in one KeyIndex.
:	:	Free for APP_ASYMM_KEY	
33	:	Free for APP_ASYMM_KEY	

Figure 4. PN7642 key store map

```
mbedtls_aes_init(&ctx);
ctx.key_index = AES128_KEY_POS;
```

Figure 5. mbedTLS AES context set key index

The member "key_index" of the context can be accessed and set by referencing to it by "ctx.key_index". Where "ctx" represents the name used by the declaration of the "mbedtls_aes_context" variable.

```
static int APP_AES_ECB(int mode)
{
    mbedtls_aes_context ctx;
    const uint8_t key[32] = {0}, src[16] = {0};
```

Figure 6. mbedTLS context declaration

Note: Setting of "key_index" is crucial for further operations.

The same concept has to be applied wherever a key from the KeyStore shall be used. For using GCM, the context has to be initialized using the GCM init function:

```
mbedtls_gcm_init(&ctx);
ctx.key_index = AES128_KEY_POS;
```

Figure 7. AES GCM set key index in context

2.3 Set keys

To prepare the crypto modules for operation, the key has to be set. For this, the according API has to be used.

```
/*!
 * @brief          This function sets the encryption key.
 *
 * \param ctx      The AES context to which the key should be bound.
 *                 It must be initialized.
 * \param key      The encryption key.
 *                 This must be a readable buffer of size \p keybits bits.
 *                 NULL if key stored in SKT is used
 * \param keybits  The size of data passed in bits. Valid options are:
 *                 <ul><li>128 bits</li>
 *                 <li>256 bits</li></ul>
 *
 * \return         \c 0 on success.
 * \return         #MBEDTLS_ERR_AES_INVALID_KEY_LENGTH on failure.
 * \return         #MBEDTLS_ERR_AES_BAD_INPUT_DATA on failure
 */
int mbedtls_aes_setkey_enc( mbedtls_aes_context *ctx, const unsigned char *key,
                    unsigned int keybits );
```

Figure 8. mbedTLS set key encryption

To highlight is the second parameter "key". It shall be set to "NULL" if we want to use a key from the key store. The key to be used is determined by the variable "key_index" of the AES context.

The parameter "keybits" is set to the length of the key.

For encryption and decryption different keys can be used and a different "setkey" API must be used.

In the 'pn_mbedtls_demo' example the key input must be changed to **NULL** as showing in the following figures:

```
if (mode == MBEDTLS_AES_ENCRYPT)
{
    result    = mbedtls_aes_setkey_enc(&ctx, NULL, keySize);
    refOutput = s_AesEcbEncRef[i];
}
else
{
    result    = mbedtls_aes_setkey_dec(&ctx, NULL, keySize);
    refOutput = s_AesEcbDecRef[i];
}
```

Figure 9. SDK example set key AES ECB

```
result = mbedtls_gcm_setkey(&ctx, MBEDTLS_CIPHER_ID_AES, NULL, keySize);
```

Figure 10. SDK example set key AES GCM

2.4 AES encryption and decryption

After the context has been initialized, see Section 2.2, and the keys have been set, see Section 2.3, the encryption and decryption method is ready to be used.

```
/*!
 * @brief          This function performs an AES single-block encryption or
 *                 decryption operation.
 *
 *                 It performs the operation defined in the \p mode parameter
 *                 (encrypt or decrypt), on the input data buffer defined in
 *                 the \p input parameter.
 *
 *                 mbedtls_aes_init(), and either mbedtls_aes_setkey_enc() or
 *                 mbedtls_aes_setkey_dec() must be called before the first
 *                 call to this API with the same context.
 *
 * \param ctx      The AES context to use for encryption or decryption.
 *                 It must be initialized and bound to a key.
 * \param mode     The AES operation: #MBEDTLS_AES_ENCRYPT or
 *                 #MBEDTLS_AES_DECRYPT.
 * \param input    The buffer holding the input data.
 *                 It must be readable and at least \c 16 Bytes long.
 * \param output   The buffer where the output data will be written.
 *                 It must be writeable and at least \c 16 Bytes long.

 * \return         \c 0 on success.
 */
int mbedtls_aes_crypt_ecb( mbedtls_aes_context *ctx,
                    int mode,
                    const unsigned char input[16],
                    unsigned char output[16] );
```

Figure 11. mbedTLS AES encryption or decryption

In the "pn_mbedtls_demo" example, the method "mbedtls_aes_crypt_ecb" is called first for encryption and after for decryption.

```
result = mbedtls_aes_crypt_ecb(&ctx, mode, src, output);

if ((result != 0) || (0 != memcmp(output, refOutput, sizeof(output))))
{
    PRINTF("Failed\r\n");

    APP_DumpArray("Reference result", refOutput, sizeof(output));
    APP_DumpArray("Actual result", output, sizeof(output));
    break;
}
else
{
    PRINTF("Pass\r\n");
    result = 0;
}
```

Figure 12. mbedTLS AES ECB

The "memcmp" in the if-else statement is comparing the output against the previously set reference data. If it is matching, "Pass" will be printed. Else the reference value and an actual result is printed for comparison.

2.5 GCM encrypt, tag, and decrypt

For GCM, the same concept is applied. After initializing the context, see Section 2.2 "Context initialization", and setting the keys, see Section 2.3, the functions mbedtls_gcm_crypt_and_tag(…) and mbedtls_gcm_auth_decrypt(…) can be used with the keys from the key store:

```
/* Encrypt and Tag. */
result = mbedtls_gcm_crypt_and_tag(&ctx, MBEDTLS_GCM_ENCRYPT, sizeof(plainData), iv, sizeof(iv), add,
                                   sizeof(add), plainData, cipherData, sizeof(tag), tag);
```

Figure 13. GCM encryption and tagging

```
/* GCM Auth and Decrypt */
result = mbedtls_gcm_auth_decrypt(&ctx, sizeof(plainData), iv, sizeof(iv), add, sizeof(add), tag, sizeof(tag),
                                  cipherData, decryptPlainData);
```

Figure 14. GCM authenticate and decrypt

3 Preparing 'pn_mbedtls_demo' example

This chapter explains how to work with the mbedTLS example "pn_mbedtls_demo" and using the PN76 key store.

The prerequisites to work with the AES module by using keys from the KeyStore are:

- mbedTLS initialized
- KeyStore initialized
- KeyStore provisioned with an APP_MASTER_KEY or APP_FIXED_KEY.

Per default the example is executing a lot of crypto operations, in this application note only the AES operation is shown. Other methods are not executed.

```
/*!
 * @brief Main function
 */
int main(void)
{
    int errors = 0;
    /* Board pin init */
    BOARD_InitBootPins();
    BOARD_InitBootClocks();
    BOARD_InitDebugConsole();

    PRINTF("\r\nPN mbedcrypto example started\r\n");

    APP_InitMbedCrypto();

    errors += APP_AES_ECB(MBEDTLS_AES_ENCRYPT);
    errors += APP_AES_ECB(MBEDTLS_AES_DECRYPT);
```

Figure 15. Example main(void)

In the method APP_AES_ECB(…) the reference data has to be changed. The default key is all '0'. The key of the reference data has to be changed to the actual provisioned key within the key store. The CT (cipher text) and PT (plain text) can be left to all '0'. But, the array has to be changed to the actual new encrypted/decrypted message. See Section 4 how to generate new reference data.

```
/* clang-format off */
const uint8_t s_AesEcbDecRef[][16] =
{
        /*
         * AES-ECB-128
         * KEY=9AA0255E371836EE0BD2C3CEDACB9542
         * CT=00000000000000000000000000000000
         */
        {0x33, 0x2f, 0x99, 0xe4, 0x78, 0xc7, 0xb0, 0xef, 0x10, 0xde, 0xcb, 0xb0, 0xb2, 0x72, 0x5b, 0xd3},
        /*
         * AES-ECB-256
         * KEY=207D74CF3EED13AE1373D61E134592F226AE1112590461623CF76EB27EF9B55C
         * CT=00000000000000000000000000000000
         */
        {0xef, 0xd4, 0x08, 0x07, 0x32, 0x5b, 0xb1, 0xdf, 0xfe, 0x40, 0xcb, 0x24, 0xa5, 0x81, 0xc4, 0xa3},
};

const uint8_t s_AesEcbEncRef[][16] =
{
        /*
         * AES-ECB-128
         * KEY=9AA0255E371836EE0BD2C3CEDACB9542
         * PT=00000000000000000000000000000000
         */
        {0x8c, 0x2f, 0x4e, 0x29, 0xef, 0x2a, 0x19, 0x16, 0x60, 0x45, 0x81, 0x51, 0x27, 0xdf, 0x9c, 0x3b},
        /*
         * AES-ECB-256
         * KEY=207D74CF3EED13AE1373D61E134592F226AE1112590461623CF76EB27EF9B55C
         * PT=00000000000000000000000000000000
         */
        {0x5a, 0x14, 0x62, 0x39, 0xd5, 0xa8, 0x60, 0x5f, 0x2d, 0xef, 0x68, 0x3e, 0x72, 0xda, 0x38, 0x13},
};
/* clang-format on */
```

Figure 16. Example reference data

In Figure 16, the reference data has been newly encrypted and decrypted with the key that shall be used.

Not only the reference data has to be updated. Also, the key to be used has to be set. After the context initialization (see Section 2.2), the keys have to be set according to their length.

```
mbedtls_aes_init(&ctx);
if(keySize == 128) {
    ctx.key_index = AES128_KEY_POS;
}
else if(keySize == 256) {
    ctx.key_index = AES256_KEY_POS;
}
else {
    /* no valid key set */
}
```

Figure 17. mbedTLS example set key_index

After these small modifications, the example uses the key from the PN76 KeyStore. The defined "AES128_KEY_POS" and "AES256_KEY_POS" represent the index of the keys in the key store.

After these modifications, the example is ready and using the keys of the KeyStore. Let the example run, if everything has been set correct it should pass:

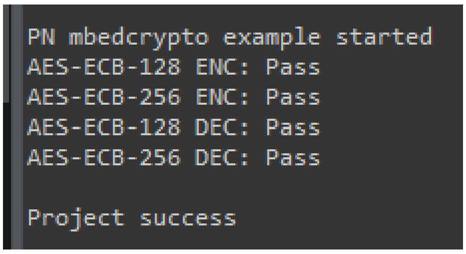

Figure 18. AES example pass

4 Annex A: Python code for new reference data

The 'pn_mbedtls_demo' does not use keys from the key store. It passes plain keys directly. Those are all zero. By using keys from the key store, which are not all zero, the reference data has to be updated to match.

Online calculators can be of great help, but a very short python script can do the same. In this chapter, a short python script to generate those reference data is shown.

In the below codeblock the generation of the reference data for ECB 128-bit keys is shown:

```
def mbedtls_aes_ecb():
    print("AES ECB 128")
    bKey = bytes.fromhex('9AA0255E371836EE0BD2C3CEDACB9542')
    bIv = bytes.fromhex('000000000000000000000000')
    bPlainText = bytes.fromhex('00000000000000000000000000000000')
    bCipherText = bytes.fromhex('00000000000000000000000000000000')

    c = cipher.AES.new(bKey,
                       cipher.MODE_ECB,
                       bIv)

    enc = c.encrypt(bPlainText)
    print("ECB encrypt:", bArrayToHex(enc))

    dec = c.decrypt(bCipherText)
    print("ECB decrypt:", bArrayToHex(dec))
```

To generate reference data with a 256-bit key, the variable "bKey" has to be changed accordingly to the 256-bit key.

In the below codeblock the generation of the reference data for GCM with a 128bit key is shown:

```
def mbedtls_gcm_128():
    print("AES GCM 128")
    bKey = bytes.fromhex('9AA0255E371836EE0BD2C3CEDACB9542')
    bIv = bytes.fromhex('000000000000000000000000')
    bAd = bytes.fromhex('00000000000000000000000000000000')
    bMsg =
 bytes.fromhex('0000000000000000000000000000000000000000000000000000000000000000')

    c = cipher.AES.new(bKey,
                       cipher.MODE_GCM,
                       bIv,
                       bAd)
    enc = c.encrypt(bMsg)
    print("GCM Cipher Text Reference:", bArrayToHex(enc[0]))
    print("GCM Tag Reference:", bArrayToHex(enc[1]))
```

The method "bArrayToHex" is to beautify the output and make it more readable and easier to copy:

```
def bArrayToHex(list_val):
    result = ''.join(' 0x{:02x}'.format(x) for x in list_val)
    return (result)
```

5 Annex B: FAQ

Q: What key is used if in the context a key index is set (ctx.keyIndex = 0x01), but the key parameter is not NULL?

A: The given key in the parameter is taken. To use a key from the key store, the key-index in the context has to be valid and the parameter must be NULL. Both conditions have to be met.

Q: Can I use APP_ROOT_KEYs for cryptographic operations?

A: No, the APP_ROOT_KEY cannot be used for cryptographic operations. Any other key can be used. The APP_ROOT_KEY is purely for authentication at the secure key store and deriving keys from it.

Q:Can the key store be used also for asymmetric keys?

A:Yes, the key store can store asymmetric keys from index 27 onwards. These keys can be used for cryptographic operations the same way symmetric keys are used.

6 Abbreviations

Table 1. Abbreviations

Acronym	Description
AES	Advanced Encryption Standard
API	application programming interface
CT	chiper text
ECB	electronic code book mode
GCM	Galois/Counter Mode
PT	plain text
SDK	software development kit

7 References

[1] AN13720 PN7642 Secure Key Mode demo application
[2] PN7642 NFC Controller User API Documentation

8 Note about the source code in the document

Example code shown in this document has the following copyright and BSD-3-Clause license:

Copyright 2023 NXP Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials must be provided with the distribution.
3. Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

9 Legal information

9.1 Definitions

Draft — A draft status on a document indicates that the content is still under internal review and subject to formal approval, which may result in modifications or additions. NXP Semiconductors does not give any representations or warranties as to the accuracy or completeness of information included in a draft version of a document and shall have no liability for the consequences of use of such information.

9.2 Disclaimers

Limited warranty and liability — Information in this document is believed to be accurate and reliable. However, NXP Semiconductors does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information. NXP Semiconductors takes no responsibility for the content in this document if provided by an information source outside of NXP Semiconductors.

In no event shall NXP Semiconductors be liable for any indirect, incidental, punitive, special or consequential damages (including - without limitation - lost profits, lost savings, business interruption, costs related to the removal or replacement of any products or rework charges) whether or not such damages are based on tort (including negligence), warranty, breach of contract or any other legal theory.

Notwithstanding any damages that customer might incur for any reason whatsoever, NXP Semiconductors' aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms and conditions of commercial sale of NXP Semiconductors.

Right to make changes — NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Suitability for use — NXP Semiconductors products are not designed, authorized or warranted to be suitable for use in life support, life-critical or safety-critical systems or equipment, nor in applications where failure or malfunction of an NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors and its suppliers accept no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is at the customer's own risk.

Applications — Applications that are described herein for any of these products are for illustrative purposes only. NXP Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

Customers are responsible for the design and operation of their applications and products using NXP Semiconductors products, and NXP Semiconductors accepts no liability for any assistance with applications or customer product design. It is customer's sole responsibility to determine whether the NXP Semiconductors product is suitable and fit for the customer's applications and products planned, as well as for the planned application and use of customer's third party customer(s). Customers should provide appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP Semiconductors does not accept any liability related to any default, damage, costs or problem which is based on any weakness or default in the customer's applications or products, or the application or use by customer's third party customer(s). Customer is responsible for doing all necessary testing for the customer's applications and products using NXP Semiconductors products in order to avoid a default of the applications and the products or of the application or use by customer's third party customer(s). NXP does not accept any liability in this respect.

Terms and conditions of commercial sale — NXP Semiconductors products are sold subject to the general terms and conditions of commercial sale, as published at http://www.nxp.com/profile/terms, unless otherwise agreed in a valid written individual agreement. In case an individual agreement is concluded only the terms and conditions of the respective agreement shall apply. NXP Semiconductors hereby expressly objects to applying the customer's general terms and conditions with regard to the purchase of NXP Semiconductors products by customer.

Export control — This document as well as the item(s) described herein may be subject to export control regulations. Export might require a prior authorization from competent authorities.

Suitability for use in non-automotive qualified products — Unless this document expressly states that this specific NXP Semiconductors product is automotive qualified, the product is not suitable for automotive use. It is neither qualified nor tested in accordance with automotive testing or application requirements. NXP Semiconductors accepts no liability for inclusion and/or use of non-automotive qualified products in automotive equipment or applications.

In the event that customer uses the product for design-in and use in automotive applications to automotive specifications and standards, customer (a) shall use the product without NXP Semiconductors' warranty of the product for such automotive applications, use and specifications, and (b) whenever customer uses the product for automotive applications beyond NXP Semiconductors' specifications such use shall be solely at customer's own risk, and (c) customer fully indemnifies NXP Semiconductors for any liability, damages or failed product claims resulting from customer design and use of the product for automotive applications beyond NXP Semiconductors' standard warranty and NXP Semiconductors' product specifications.

Translations — A non-English (translated) version of a document, including the legal information in that document, is for reference only. The English version shall prevail in case of any discrepancy between the translated and English versions.

Security — Customer understands that all NXP products may be subject to unidentified vulnerabilities or may support established security standards or specifications with known limitations. Customer is responsible for the design and operation of its applications and products throughout their lifecycles to reduce the effect of these vulnerabilities on customer's applications and products. Customer's responsibility also extends to other open and/or proprietary technologies supported by NXP products for use in customer's applications. NXP accepts no liability for any vulnerability. Customer should regularly check security updates from NXP and follow up appropriately.

Customer shall select products with security features that best meet rules, regulations, and standards of the intended application and make the ultimate design decisions regarding its products and is solely responsible for compliance with all legal, regulatory, and security related requirements concerning its products, regardless of any information or support that may be provided by NXP.

NXP has a Product Security Incident Response Team (PSIRT) (reachable at PSIRT@nxp.com) that manages the investigation, reporting, and solution release to security vulnerabilities of NXP products.

NXP B.V. - NXP B.V. is not an operating company and it does not distribute or sell products.

9.3 Licenses

Purchase of NXP ICs with NFC technology — Purchase of an NXP Semiconductors IC that complies with one of the Near Field Communication (NFC) standards ISO/IEC 18092 and ISO/IEC 21481 does not convey an implied license under any patent right infringed by implementation of any of those standards. Purchase of NXP Semiconductors IC does not include a license to any NXP patent (or other IP right) covering combinations of those products with other products, whether hardware or software.

9.4 Trademarks

Notice: All referenced brands, product names, service names, and trademarks are the property of their respective owners.

NXP — wordmark and logo are trademarks of NXP B.V.

EdgeVerse — is a trademark of NXP B.V.

Tables

Figures

Contents

Date of release: 14 September 2023
Document identifier: AN14060