Hochschule Offenburg
offenburg.university

ivESK Institut für verlässliche Embedded Systems und Kommunikationselektronik

E+I Elektrotechnik und Informationstechnik

Generic Cryptographic Interface

Documentation

Steve Wagner
EI-3nat

February 1, 2016

Prof. Dr. Axel Sikora
Dipl.-Phys. Andreas Walz
Institute of reliable Embedded Systems and Communication Electronics (ivESK)
Offenburg University of Applied Sciences

Statutory declaration

I declare that I have authored this thesis independently, that I have not used other than the declared sources / resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

Offenburg, _____ _____

Date Signature

Contents

TODO

1 Introduction

2 Motivation

3 Design

4 Initialisation of the interface

5 Context management

5.1 Definition

The contexts represent the state of stateful algorithms. It allows to avoid the hidden states in the interface, which mean that what is entering in the context as configuration for an algorithm will be used for the calculation and no parameters will be statically written in the functions of the interface.

5.2 Create a context

The princip of context is available for the following algorithms:

- Hash
- Signature
- Symmetric cipher
- Asymmetric cipher
- Diffie-Hellman

5.2.1 Hash context

Prototype:

```
/**
 * \fn              en_gciResult_t gciHashNewCtx( en_gciHashAlgo_t hashAlgo, GciCtxId_t*
    p_ctxID )
 * \brief           Create a new hash context and become an ID of it
 * \param [in]  hashAlgo    Algorithm of the hash context
 * \param [out] p_ctxID     Pointer to the context's ID
 * @return            en_gciResult_Ok on success
 * @return            en_gciResult_Err on error
 */
en_gciResult_t gciHashNewCtx( en_gciHashAlgo_t hashAlgo, GciCtxId_t* p_ctxID );
```

The context is only needed to save the Hash algorithm. For more informations of how to use a Hash context, see chapter 6.

5.2.2 Signature context (to generate a signature)

Prototype:

```
1
2  /**
3   * \fn              en_gciResult_t gciSignGenNewCtx( const st_gciSignConfig_t*
       p_signConfig, GciKeyId_t keyID, GciCtxId_t* p_ctxID )
4   * \brief           Create a new signature context and become an ID of it
5   * \param [in]  p_signConfig  Pointer to the configuration of the signature
6   * \param [in]  keyID     Key's ID
7   * \param [out] p_ctxID     Pointer to the context's ID
8   * @return            en_gciResult_Ok on success
9   * @return            en_gciResult_Err on error
10  */
11 en_gciResult_t gciSignGenNewCtx( const st_gciSignConfig_t* p_signConfig, GciKeyId_t keyID
       , GciCtxId_t* p_ctxID );
```

This context is used to save several different configuration to generate:

- RSA signature
- DSA signature
- ECDSA signature
- Cipher Message Authentication Code (CMAC)
- Hash Message Authentication Code (HMAC)

Only one configuration is possible for one context.
For more details about each configuration listed above see chapter 7

5.2.3 Signature context (to verify a signature)

Prototype:

```
1
2  /**
3   * \fn              en_gciResult_t gciSignVerifyNewCtx( const st_gciSignConfig_t*
       p_signConfig, GciKeyId_t keyID, GciCtxId_t* p_ctxID )
4   * \brief           Create a new signature context and become an ID of it
5   * \param [in]  p_signConfig  Pointer to the configuration of the signature
6   * \param [in]  keyID     Key's ID
7   * \param [out] p_ctxID     Pointer to the context's ID
8   * @return            en_gciResult_Ok on success
9   * @return            en_gciResult_Err on error
10  */
11 en_gciResult_t gciSignVerifyNewCtx( const st_gciSignConfig_t* p_signConfig, GciKeyId_t
       keyID, GciCtxId_t* p_ctxID );
```

This context is used to save several different configuration to verify:

- RSA signature
- DSA signature
- ECDSA signature
- Cipher Message Authentication Code (CMAC)
- Hash Message Authentication Code (HMAC)

Only one configuration is possible for one context.
For more details about each configuration listed above see chapter 7

5.2.4 Cipher context

Prototype:

```
1
2  /**
3   * \fn              en_gciResult_t gciCipherNewCtx( const st_gciCipherConfig_t*
       p_ciphConfig, GciKeyId_t keyID, GciCtxId_t* p_ctxID )
4   * \brief           Create a new symmetric cipher context
5   * \param [in]  p_ciphConfig  Pointer to the configuration of the symmetric cipher
6   * \param [in]  keyID     Key's ID
7   * \param [out] p_ctxID     Pointer to the context's ID
8   * @return            en_gciResult_Ok on success
9   * @return            en_gciResult_Err on error
10  */
11 en_gciResult_t gciCipherNewCtx( const st_gciCipherConfig_t* p_ciphConfig, GciKeyId_t
       keyID, GciCtxId_t* p_ctxID );
```

This context is use to encrypt a plaintext or decrypt a ciphertext.

The cipher algorithm which could be used are:

- Symmetric stream cipher RC4
- Symmetric block cipher AES
- Symmetric block cipher DES
- Symmetric block cipher 3DES
- Asymmetric cipher RSA

For more details of each configuration see chapter 9

5.2.5 Diffie-Hellmann context

Prototype:

```
1
2  /**
3   * \fn              en_gciResult_t gciDhNewCtx( const st_gciDhConfig_t* p_dhConfig,
       GciCtxId_t* p_ctxID )
4   * \brief           Create a new Diffie-Hellman context
5   * \param [in]  p_dhConfig    Pointer to the configuration of the Diffie-Hellman
6   * \param [out] p_ctxID     Pointer to the context's ID
7   * @return            en_gciResult_Ok on success
8   * @return            en_gciResult_Err on error
9   */
10 en_gciResult_t gciDhNewCtx( const st_gciDhConfig_t* p_dhConfig, GciCtxId_t* p_ctxID );
```

This context is used to created :

- Diffie-Hellman
- Elliptic Curve Diffie-Hellman
- Shared secret for Diffie-Hellman
- Shared secret for Elliptic Curve Diffie-Hellman

Furthermore, the private key generated must be saved internally.

For more information about the configuration and the generation of the Diffie-Hellman/Elliptic Curve Diffie-Hellman key pair see chapter 10

For more informations about the generation of the shared secret see chapter 11

5.3 Clone an existing context

One of the inconvenient of the interface comes from the finish part, where the last calculation is done. For the hash algorithm and the signature algorithm, when the digest/signature is calculated, no more updates could be done with the last configuration and with the previous updates.

The solution of this problem is the clone of the context.

When the digest/signature has to be calculated but the configuration and the previous updates should be kept, the clone of the hash/signature context allows to copy the configuration and the previous updates. Two contexts are identical but one is use for the calculation of the digest/signature and the other one for futur updates.

5.3.1 Hash context

Prototype:

```
1
2  /*!
3   * \fn              en_gciResult_t gciHashCtxClone( GciCtxId_t idSrc, GciCtxId_t*
       p_idDest )
4   * \brief           Clone a context
5   * \param [in]  idSrc     The context which will be cloned
6   * \param [out] p_idDest    Pointer to the context ID where the source context is cloned
7   * @return            en_gciResult_Ok on success
8   * @return            en_gciResult_Err on error
9   */
10 en_gciResult_t gciHashCtxClone( GciCtxId_t idSrc, GciCtxId_t* p_idDest );
```

5.3.2 Both Signature context

Prototype:

```
1
2  /*!
3   * \fn              en_gciResult_t gciSignCtxClone( GciCtxId_t idSrc, GciCtxId_t*
       p_idDest )
4   * \brief           Clone a context
5   * \param [in]  idSrc     The context which will be cloned
6   * \param [out] p_idDest    Pointer to the context ID where the source context is cloned
7   * @return            en_gciResult_Ok on success
8   * @return            en_gciResult_Err on error
9   */
10 en_gciResult_t gciSignCtxClone( GciCtxId_t idSrc, GciCtxId_t* p_idDest );
```

5.4 Delete an existing context

When the context is not needed anymore, it can be removed and be used for an other configuration, which can be completely different as the previous one.

Prototype:

```
1
2 /*!
3  * \fn              en_gciResult_t gciCtxRelease( GciCtxId_t ctxID )
4  * \brief           Release a context
5  * \param [in] ctxID      Context's ID
6  * @return            en_gciResult_Ok on success
7  * @return            en_gciResult_Err on error
8  */
9 en_gciResult_t gciCtxRelease( GciCtxId_t ctxID );
```

6 Hash functions

6.1 Algorithm of hash

```
1
2  /**
3   * \enum           en_GciHashAlgo
4   * \brief          Enumeration for Hash algorithms
5   */
6  typedef enum en_GciHashAlgo
7  {
8    /** Invalid Hash */
9    en_gciHashAlgo_Invalid,
10   /** MD5 Hash */
11   en_gciHashAlgo_MD5,
12   /** SHA1 Hash */
13   en_gciHashAlgo_SHA1,
14   /** SHA224 Hash */
15   en_gciHashAlgo_SHA224,
16   /** SHA256 Hash */
17   en_gciHashAlgo_SHA256,
18   /** SHA384 Hash */
19   en_gciHashAlgo_SHA384,
20   /** SHA512 Hash */
21   en_gciHashAlgo_SHA512,
22   /** No hash algorithm used */
23   en_gciHashAlgo_None=0xFF
24 } en_gciHashAlgo_t;
```

6.2 Prototypes

6.2.1 Create a hash context

```
1
2  /**
3   * \fn              en_gciResult_t gciHashNewCtx( en_gciHashAlgo_t hashAlgo, GciCtxId_t*
       p_ctxID )
4   * \brief           Create a new hash context and become an ID of it
5   * \param [in]  hashAlgo    Algorithm of the hash context
6   * \param [out] p_ctxID     Pointer to the context's ID
7   * @return            en_gciResult_Ok on success
8   * @return            en_gciResult_Err on error
9   */
10 en_gciResult_t gciHashNewCtx( en_gciHashAlgo_t hashAlgo, GciCtxId_t* p_ctxID );
```

6.2.2 Update a hash context

```
1
2  /**
3   * \fn              en_gciResult_t gciHashUpdate( GciCtxId_t ctxID, const uint8_t*
       p_blockMsg, size_t blockLen )
4   * \brief           Add block of the message
5   * \param [in]  ctxID     Context's ID
6   * \param [in]  p_blockMsg    Pointer to the block of the message
7   * \param [in]  blockLen    Block message's length
8   * @return            en_gciResult_Ok on success
9   * @return            en_gciResult_Err on error
10  */
11 en_gciResult_t gciHashUpdate( GciCtxId_t ctxID, const uint8_t* p_blockMsg, size_t
       blockLen );
```

6.2.3 Clone a hash context

```
1
2  /*!
3   * \fn              en_gciResult_t gciHashCtxClone( GciCtxId_t idSrc, GciCtxId_t*
       p_idDest )
4   * \brief           Clone a context
5   * \param [in]  idSrc     The context which will be cloned
6   * \param [out] p_idDest    Pointer to the context ID where the source context is cloned
7   * @return            en_gciResult_Ok on success
8   * @return            en_gciResult_Err on error
9   */
10 en_gciResult_t gciHashCtxClone( GciCtxId_t idSrc, GciCtxId_t* p_idDest );
```

6.2.4 Finish a hash context

```
1
2  /**
3   * \fn              en_gciResult_t gciHashFinish( GciCtxId_t ctxID, uint8_t* p_digest,
       size_t* p_digestLen )
4   * \brief           Get the digest of the message after adding all the block of the
       message
5   * \param [in]  ctxID     Context's ID
6   * \param [out] p_digest    Pointer to the digest of the complete message added
7   * \param [out] p_digestLen   Pointer to the length of the digest in bytes
8   * @return            en_gciResult_Ok on success
9   * @return            en_gciResult_Err on error
10  */
11 en_gciResult_t gciHashFinish( GciCtxId_t ctxID, uint8_t* p_digest, size_t* p_digestLen );
```

6.3 Steps to hash (Example)

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "crypto_iface.h"

int main(int argc , char *argv[])
{
    /* Error Management */
    en_gciResult_t err;

    /* MD5 context ID */
    GciCtxId_t md5CtxID, md5CloneCtxID;

    /* Messages to hash */
    uint8_t a_data1[10] = {"Hello!"};
    uint8_t a_data2[30] = {"This is a Hash MD5 test"};
    uint8_t a_data3[10] = {"Thank you."};

    size_t data1Len = strlen(a_data1);
    size_t data2Len = strlen(a_data2);
    size_t data3Len = strlen(a_data3);

    int i;

    /* a MD5 digest is always 128 bits -> 16 bytes */
    uint8_t a_digest[GCI_MD5_SIZE_BYTES];

    /* Initialize the buffer */
    memset(a_digest, 0, GCI_MD5_SIZE_BYTES);

    size_t digestLen = 0;

    /* Create a new hash MD5 context */
    err = gciHashNewCtx(en_gciHashAlgo_MD5, &md5CtxID);

    /* Error coming from the creation of a new MD5-Hash context */
    if(err != en_gciResult_Ok)
    {
        printf("GCI Error in gciHashNewCtx: MD5");
    }

    /* Add the first data by updating the hash context */
    err = gciHashUpdate(md5CtxID, a_data1, data1Len);

    /* Error coming from the updating of the hash context with data1 */
    if(err != en_gciResult_Ok)
    {
        printf("GCI Error in gciHashUpdate: MD5");
    }

    /* Add the second data by updating the hash context */
    err = gciHashUpdate(md5CtxID, a_data2, data2Len);

    /* Error coming from the updating of the hash context with data2 */
    if(err != en_gciResult_Ok)
    {
        printf("GCI Error in gciHashUpdate: MD5");
    }
```

```
59
60      /* Clone the context */
61      err = gciHashCtxClone(md5CtxID, &md5CloneCtxID);
62      if(err != en_gciResult_Ok)
63      {
64          printf("GCI Error in gciHashCtxClone: MD5");
65      }
66
67      /* Get the digest of this message */
68      err = gciHashFinish(md5CtxID, a_digest, &digestLen);
69      if(err != en_gciResult_Ok)
70      {
71          printf("GCI Error in gciHashFinish: MD5");
72      }
73
74      else
75      {
76          printf("GCI Info: Digest1 = ");
77          for(i = 0; i < GCI_MD5_SIZE_BYTES; i++)
78          {
79              printf("%d", a_digest[i]);
80          }
81      }
82
83      /* Initialize the buffer */
84      memset(a_digest, 0, GCI_MD5_SIZE_BYTES);
85
86      /* Add the third data by updating the hash context */
87      err = gciHashUpdate(md5CloneCtxID, a_data3, data3Len);
88
89      /* Error coming from the updating of the hash context with data3 */
90      if(err != en_gciResult_Ok)
91      {
92          printf("GCI Error in gciHashUpdate: MD5");
93      }
94
95      /* Get the digest of this message */
96      err = gciHashFinish(md5CloneCtxID, a_digest, &digestLen);
97      if(err != en_gciResult_Ok)
98      {
99          printf("GCI Error in gciHashFinish: MD5");
100     }
101
102     else
103     {
104         printf("\r\nGCI Info: Digest2 = ");
105         for(i=0; i<GCI_MD5_SIZE_BYTES; i++)
106         {
107             printf("%d", a_digest[i]);
108         }
109
110     }
111
112     printf("\r\n");
113
114     /* Delete the contexts */
115     gciCtxRelease(md5CtxID);
116     gciCtxRelease(md5CloneCtxID);
117
118 }
```

7 Signature/MAC algorithms

The complete structure of the signatures/MAC's configuration:

```
1
2  /**
3   * \struct          st_gciSignConfig
4   * \brief         Structure for the configuration of a signature
5   */
6  typedef struct st_gciSignConfig
7  {
8    /**
9     * en_gciSignAlgo_Invalid
10    * en_gciSignAlgo_RSA
11    * en_gciSignAlgo_DSA
12    * en_gciSignAlgo_ECDSA
13    * en_gciSignAlgo_MAC_ISO9797_ALG1
14    * en_gciSignAlgo_MAC_ISO9797_ALG3
15    * en_gciSignAlgo_CMAC_AES
16    * en_gciSignAlgo_HMAC
17    * en_gciSignAlgo_RSASSA_PSS
18    * en_gciSignAlgo_RSASSA_PKCS
19    * en_gciSignAlgo_RSASSA_X509
20    * en_gciSignAlgo_ECDSA_GFP
21    * en_gciSignAlgo_ECDSA_GF2M
22    * en_gciSignAlgo_None = 0xFF
23    */
24   en_gciSignAlgo_t algo;
25
26
27   /**
28    * en_gciHashAlgo_Invalid
29    * en_gciHashAlgo_MD5
30    * en_gciHashAlgo_SHA1
31    * en_gciHashAlgo_SHA224
32    * en_gciHashAlgo_SHA256
33    * en_gciHashAlgo_SHA384
34    * en_gciHashAlgo_SHA512
35    * en_gciHashAlgo_None=0xFF
36    */
37   en_gciHashAlgo_t hash;
38
39   /**
40    * \union          un_signConfig
41    * \brief        Union for the configuration of each signature
42    */
43   union un_signConfig
44   {
45     /** RSA Configuration */
46     st_gciSignRsaConfig_t signConfigRsa;
47
48     /** CMAC Configuration */
49     st_gciSignCmacConfig_t signConfigCmac;
```

```
50
51    } un_signConfig;
52
53 } st_gciSignConfig_t;
```

7.1 MAC / Signature configuration

Two differents use of the signature are available.
The first one is the generation of a signature, which will sign the datas updated (see section 7.2 for more details.

The second one is the verification of a signature, which will sign the datas updated but will at the end compared with this entered in the function (see section 7.3 for more details).

7.1.1 RSA

Test RSA gen sign with LibTomCrypt before writing this part

7.1.2 Digital Signature Algorithm (DSA)

Test DSA gen sign with LibTomCrypt before writing this part

7.1.3 Elliptic Curve Digital Signature Algorithm (ECDSA)

Test ECDSA gen sign with LibTomCrypt before writing this part

7.1.4 Block-Cipher-Based Message Authentication Code (CBC-MAC / CMAC)

Test CMAC gen sign with LibTomCrypt before writing this part

7.1.5 keyed-Hash Message Authentication Code (HMAC)

For the HMAC, only the Hash has to be specified:

```
1
2 /* HMAC Configuration */
3 st_gciSignConfig_t hmacConfig;
4
5 /* HMAC Algorithm */
6 hmacConfig.algo = en_gciSignAlgo_HMAC;
7
8 /* Hash MD5 Algorithm */
9 hmacConfig.hash = en_gciHashAlgo_MD5;
```

7.2 Steps to generate a signature/MAC

Sign gen example with HMAC

Implement gciSignCtxClone for the example

7.3 Steps to verify a signature/MAC

Sign vry example with HMAC

Implement gciSignCtxClone for the example

8 Generation of key pair

8.1 Configuration of a key pair

8.1.1 RSA

8.1.2 Digital Signature Algorithm (DSA)

8.1.3 Elliptic Curve Digital Signature Algorithm (ECDSA)

8.2 Steps to generate a key pair

9 Cipher algorithms

9.1 Configuration of a symmetric cipher

9.2 Configuration of an asymmetric cipher

9.3 Encrypt a plaintext

9.4 Decrypt a ciphertext

10 Generation of Diffie-Hellmann key pair

10.1 Configuration of a Diffie-Hellmann key pair

10.1.1 Diffie-Hellmann (DH)

10.1.2 Elliptic Curve Diffie Hellmann (ECDH)

10.2 Steps to generate a Diffie-Hellmann key pair

11 Calculation of a Diffie-Hellmann shared secret

11.1 Steps to calculate a shared secret

12 Pseudo-Random Number Generator

12.1 Generation of a pseudo-random number

12.2 Seed a pseudo-random number

13 Key management

13.1 Save a key and get an ID

13.2 Get of a saved key with its ID

13.3 Delete a key

Bibliography

[1] Christof Paar and Jan Pelzl. *Understanding cryptography: a textbook for students and practitioners*. Springer Berlin Heidelberg, Berlin; Heidelberg [u.a.], 2. corr. printing edition, 2010.