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

March 19, 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

List of Tables

1. Motivation

Through the bachelor thesis “Extension and Integration of an Abstract Interface to Cryptography Providers” done in the Institute of reliable Embedded Systems and Communication Electronics (ivEsk), this Generic Cryptographic Interface (GCI) has been designed to have a base of the main cryptographic algorithms. The interface has to be as generic as possible to use different kinds of cryptographic providers, which can be open-source cryptographic software libraries or hardware-based cryptographic modules, and to be used in different applications. The institute wants to use hardware-based cryptographic module with one of its particularities of key management. The interface shall have a key management too, so it can be possible to use it from the hardware-based cryptographic module. When a cryptographic algorithm of the interface is used, it has to be possible to configure it. The behavior of the algorithm should only be assigned by the parameters coming from the configuration. No hidden states shall be used in the interface’s function, meaning that all parameters written in input of the function, as configuration, will be used for the cryptographic algorithm and nothing else.

2. Interface

2.1. Initialization

The initialization allows to initialized some part of the project if needed. A user name and a password can be added if the initialization has to stay private.

```
en_gciResult_t gciInit( const uint8_t* p_user, size_t userLen, const uint8_t*
p_password, size_t passLen )
```

Direction	Type	Parameter	Definition
Input	uint8_t*	p_user	Pointer to the user name
Input	size_t	userLen	Length of the user name
Input	uint8_t	p_password	Pointer to the password
Input	size_t	passLen	Length of the password

Table 2.1.: Parameters for the initialization of the interface

2.2. Delete the initialization of the interface

When the project is going to the end, some parts have to be delete, i.e free the memory used. This is the use of this part

```
en_gciResult_t gciDeinit( void )
```

3. Context management

3.1. Definition

Several different configurations are possible for a cryptographic algorithm. These configurations should not have parameters, which are implicitly written in the functions of the interface, but all of them has to be configurable from the application part. Furthermore, data can be added to the algorithm at any time, meaning that the state of the algorithm, the configuration and the previously added data, have to be saved somewhere. The result can be computed, only when the application needs it, that's why should all the data and the configuration be saved somewhere too. That's why the principle of the context is used. It represents the state of the stateful algorithms. Through the contexts, these informations, the configuration and the data, are encapsulated and referred by an ID, which is used by the application when other data has to be added or when the result has to be computed. The contexts also allow to the application part to add data to an algorithm by only passing the ID and this data at any time. The interface knows that this information has to be added to the context with the referred ID. The context keeps the configuration and the data till it's removed or a result is computed. When a result is computed, the context cannot be used again, because the data are not saved in the interface, but added to a function from the provider and this one removes the data when the result are computed. The context should then be removed and created again. This can be a problem, but is solved and explained in the section 3.3. The cryptographic algorithms, which the principle of context is used, are:

- Hash, see section 4
- Signature, see section 5
- Message Authentication Code (MAC) 5
- Symmetric cipher, see section 7
- Asymmetric cipher, see section 7
- Diffie-Hellman, see section 8

3.2. Creation of a context

Sometimes some parameters of a context need to change after the creation of a context. That's why the ID of the context has to be initialized to -1. In this case a context will be created, else the parameters who change since the previous configuration will be saved in the context refered by the ID given in input.

3.3. 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.

The principle of the clone is used for:

- Hash algorithm, see section 4
- Signture/MAC algorithm, see section 5

3.4. 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 en_gciResult_t gciCtxRelease( GciCtxId_t ctxID );
```

Direction	Type	Parameter	Definition
Input	GciCtxId_t	ctxID	Context's ID

Table 3.1.: Parameters to delete a context

4. Hash functions

4.1. Configuration of hash algorithms

Algorithm	Parameter
MD5	en_gciHashAlgo_Md5
SHA1	en_gciHashAlgo_Sha1
SHA224	en_gciHashAlgo_Sha224
SHA256	en_gciHashAlgo_Sha256
SHA384	en_gciHashAlgo_Sha384
SHA512	en_gciHashAlgo_Sha512

Table 4.1.: Hash algorithms (en_gciHashAlgo_t)

4.2. Prototypes

4.2.1. Create a hash context

```
1 en_gciResult_t gciHashNewCtx( en_gciHashAlgo_t hashAlgo, GciCtxId_t* p_ctxID );
```

Direction	Type	Parameter	Definition
Input	en_gciHashAlgo_t	hashAlgo	Algorithm of the hash context
Output	GciCtxId_t*	p_ctxID	Pointer to the context's ID

Table 4.2.: Parameters for the creation of a hash context

4.2.2. Update a hash context

```
1 en_gciResult_t gciHashUpdate( GciCtxId_t ctxID, const uint8_t* p_blockMsg, size_t
    blockLen );
```

Direction	Type	Parameter	Definition
Input	GciCtxId_t	ctxID	Context's ID
Input	uint8_t*	p_blockMsg	Pointer to the block of the message
Input	size_t	blockLen	Block message's length
Return	en_gciResult_t en_gciResult_t	en_gciResult_Ok en_gciResult_Err	When the data has been added on success When error(s) occured

Table 4.3.: Parameters for the update of a new hash context

4.2.3. Clone a hash context

The explication of the use of the clone of a hash context is done in section 3.3

```
en_gciResult_t gciHashCtxClone( GciCtxId_t idSrc, GciCtxId_t* p_idDest );
```

Direction	Type	Parameter	Definition
Input	GciCtxId_t	idSrc	The context which will be cloned
Output	GciCtxId_t*	p_idDest	Pointer to the clone context ID

Table 4.4.: Parameters for the clone of a hash context

4.2.4. Finish a hash context

```
en_gciResult_t gciHashFinish( GciCtxId_t ctxID, uint8_t* p_digest, size_t* p_digestLen );
```

Direction	Type	Parameter	Definition
Input	GciCtxId_t	ctxID	Context's ID
Output	uint8_t*	p_digest	Pointer to the digest of the complete added message
Output	size_t*	p_digestLen	Pointer to the length of the digest in bytes

Table 4.5.: Parameters for the calculation of the digest

4.3. Steps to hash (Example)

```
#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;
```

```
10
11      /* MD5 context ID -> Always initialize to -1 */
12      GciCtxId_t md5CtxID = -1, md5CloneCtxID = -1;
13
14      /* Messages to hash */
15      uint8_t a_data1[10] = {"Hello!"};
16      uint8_t a_data2[30] = {"This is a Hash MD5 test"};
17      uint8_t a_data3[10] = {"Thank you."};
18
19      size_t data1Len = strlen(a_data1);
20      size_t data2Len = strlen(a_data2);
21      size_t data3Len = strlen(a_data3);
22
23      int i;
24
25      /* a MD5 digest is always 128 bits -> 16 bytes */
26      uint8_t a_digest[GCI_MD5_SIZE_BYTES];
27
28      /* Initialize the buffer */
29      memset(a_digest, 0, GCI_MD5_SIZE_BYTES);
30
31      size_t digestLen = 0;
32
33      /* Create a new hash MD5 context */
34      err = gciHashNewCtx(en_gciHashAlgo_MD5, &md5CtxID);
35
36      /* Error coming from the creation of a new MD5-Hash context */
37      if(err != en_gciResult_Ok)
38      {
39          printf("GCI Error in gciHashNewCtx: MD5");
40      }
41
42      /* Add the first data by updating the hash context */
43      err = gciHashUpdate(md5CtxID, a_data1, data1Len);
44
45      /* Error coming from the updating of the hash context with data1 */
46      if(err != en_gciResult_Ok)
47      {
48          printf("GCI Error in gciHashUpdate: MD5");
49      }
50
51      /* Add the second data by updating the hash context */
52      err = gciHashUpdate(md5CtxID, a_data2, data2Len);
53
54      /* Error coming from the updating of the hash context with data2 */
55      if(err != en_gciResult_Ok)
56      {
57          printf("GCI Error in gciHashUpdate: MD5");
58      }
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     /* Delete the contexts */
113     gciCtxRelease(md5CtxID);
114     gciCtxRelease(md5CloneCtxID);
115
116 }
```

5. Signature/MAC algorithms

5.1. Configuration of Signature / MAC algorithms

Algorithm	Parameter
RSA	en_gciSignAlgo_RSA
DSA	en_gciSignAlgo_DSA
ECDSA	en_gciSignAlgo_ECDSA
MAC ISO9797 Algorithm 1	en_gciSignAlgo_MAC_ISO9797_ALG1
MAC ISO9797 Algorithm 3	en_gciSignAlgo_MAC_ISO9797_ALG3
Cipher-based MAC	en_gciSignAlgo_CMAC
Hash-based MAC	en_gciSignAlgo_HMAC

Table 5.1.: Signature/MAC algorithms (en_gciSignAlgo_t)

Parameter	Type
algo	en_gciSignAlgo_t
hash	en_gciHashAlgo_t
signConfigRsa (If RSA is used as algorithm)	st_gciSignRsaConfig_t
signConfigCmac (If CMAC is used as algorithm)	st_gciSignCmacConfig_t

Table 5.2.: Configuration of Signature/MAC algorithms (st_gciSignConfig_t))

5.2. Prototypes

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 **??** 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 **??** for more details).

5.2.1. Creation of a context

There is two possibilities of use for the signature context. The first one is to generate a signature and the second one to verify a signature. It has been split because some known provider needs to do the difference between the twice possibilities.

```
en_gciResult_t gciSignGenNewCtx( const st_gciSignConfig_t* p_signConfig,
GciKeyId_t keyID, GciCtxId_t* p_ctxID )
```

```
en_gciResult_t gciSignVerifyNewCtx( const st_gciSignConfig_t* p_signConfig,
GciKeyId_t keyID, GciCtxId_t* p_ctxID )
```

Direction	Type	Parameter	Definition
Input	st_gciSignConfig_t*	p_signConfig	Pointer to the configuration of the signature
Input	GciKeyId_t	keyID	ID of the key uses to sign
Output	GciCtxId_t*	p_ctxID	Pointer to the context's ID

Table 5.3.: Parameters for the creation of a signature/MAC context

RSA

The hash algorithm can be used if the updated data has to be hashed before to be signed. If not, this parameter should be configure as en_gciHashAlgo_None.

Parameter	Configuration	Prohibition(s)
algo	en_gciSignAlgo_Rsa	-
hash	en_gciHashAlgo_t	en_gciHash_Invalid
padding	en_gciPadding_t	en_gciPadding_Invalid en_gciPadding_None

Table 5.4.: Configuration of RSA Signature Scheme Algorithms (st_gciSignConfig_t)

DSA

Parameter	Configuration	Prohibition(s)
algo	en_gciSignAlgo_Dsa	-
hash	en_gciHashAlgo_t	en_gciHash_Invalid

Table 5.5.: Configuration of Digital Signature Algorithms (st_gciSignConfig_t)

ECDSA

Parameter	Configuration	Prohibition(s)
algo	en_gciSignAlgo_Ecdsa	-
hash	en_gciHashAlgo_t	en_gciHash_Invalid

Table 5.6.: Configuration of Elliptic Curve Digital Signature Algorithms (st_gciSignConfig_t)

Cipher-based MAC (CMAC)

Parameter	Configuration	Prohibition(s)
algo	en_gciSignAlgo_Cmac	-
hash	en_gciHashAlgo_t	en_gciHash_Invalid
block	en_gciBlockMode_t	en_gciBlockMode_Invalid en_gciBlockMode_None
padding	en_gciPadding_t	en_gciPadding_Invalid
iv.data iv.len	uint8_t* size_t	NULL value less or equal than 0

Table 5.7.: Configuration of Cipher-based MAC (st_gciSignConfig_t)

Hash-based MAC (HMAC)

Parameter	Configuration	Prohibition(s)
algo	en_gciSignAlgo_Hmac	-
hash	en_gciHashAlgo_t	en_gciHash_Invalid en_gciHash_None

Table 5.8.: Configuration of Hash-based MAC (st_gciSignConfig_t)

5.2.2. Update of the context

The update of the context for generating and verificate a signature is the same. The data have to be added to get a signature.

```
1 en_gciResult_t gciSignUpdate( GciCtxId_t ctxID, const uint8_t* p_blockMsg,
2 size_t blockLen )
```

Direction	Type	Parameter	Definition
Input	GciCtxId_t	ctxID	Context's ID
Input	uint8_t*	p_blockMsg	Pointer to the message to sign
Input	size_t	blockLen	Length of message

Table 5.9.: Parameters for the update of a signature/MAC context

5.2.3. Clone of signature/MAC algorithm

```
1 en_gciResult_t gciSignCtxClone( GciCtxId_t idSrc, GciCtxId_t* p_idDest )
```

Direction	Type	Parameter	Definition
Input	GciCtxId_t	idSrc	The context which will be cloned
Output	GciCtxId_t*	p_idDest	Pointer to the clone context ID

Table 5.10.: Parameters for the clone of a signature/MAC context

5.2.4. Calculation / Verification of the signature

After the data have been added to the context, if the context is to generate a signature, than the signature will be compted and returned. If the context if to verify a signature, than the signature to verify has to be added to the function. Internally the signature with the updated data will be computed but ot returned. Only if the the returning value of the function indicates if the signatures are the same or not.

```
en_gciResult_t gciSignGenFinish( GciCtxId_t ctxID, uint8_t* p_sign, size_t*
p_signLen )
```

Direction	Type	Parameter	Definition
Input	GciCtxId_t	ctxID	Context's ID
Input	uint8_t*	p_sign	Pointer to the generated signature
Input	size_t*	signLen	Pointer to the length of the generated signature

Table 5.11.: Parameters for the computation of a signature/MAC

```
en_gciResult_t gciSignVerifyFinish( GciCtxId_t ctxID, const uint8_t* p_sign,
size_t signLen )
```

Direction	Type	Parameter	Definition
Input	GciCtxId_t	ctxID	Context's ID
Input	uint8_t*	p_sign	Pointer to the signature to verify
Input	size_t	signLen	Length of the signature to verify

Table 5.12.: Parameters for the verification of a signature/MAC

5.3. Step to generate a signature

For this example the keys have already been added previously to the interface and an ID returned.

```

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

```

```c
int main(int argc , char *argv[])
{

    /* Configuration of a RSA signature */
    st_gciSignConfig_t signConfig = {.algo = en_gciSignAlgo_Rsa,
        .hash = en_gciHashAlgo_None
    };

    signConfig.un_signConfig.signConfigRsa.padding = en_gciPadding_Pkcs1_Emsa;


    /* Error Management */
    en_gciResult_t err;

    /* Messages to hash */
    uint8_t a_data1[10] = {"Hello!"};
    uint8_t a_data2[30] = {"This is a RSA Signature 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);

    /* Buffer for the signature */
    uint8_t a_signature[30];
    size_t signLen;


    int i;

    /* RSA context ID */
    GciCtxId_t rsaCtxID;

    /* Init of the signature */
    memset(a_signature, 0, 30);
    signLen = 0;

    /* Creation of the signature context with the RSA private key */
    err = gciSignGenNewCtx(&signConfig, rsaPrivKeyID, &rsaCtxID);

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

    /* First update of the signature */
    err = gciSignUpdate(rsaCtxID, a_data1, data1Len);

    /* Error coming from the update of a RSA-signature context */
    if(err != en_gciResult_Ok)
    {
        printf("GCI Error in gciSignUpdate: RSA");
    }

    /* Second update of the signature */
    err = gciSignUpdate(rsaCtxID, a_data2, data2Len);

    /* Error coming from the update of a RSA-signature context */
    if(err != en_gciResult_Ok)
```

```
67      {
68          printf("GCI Error in gciSignUpdate: RSA");
69      }
70
71      /* Third update of the signature */
72      err = gciSignUpdate(rsaCtxID, a_data3, data3Len);
73
74      /* Error coming from the update of a RSA-signature context */
75      if(err != en_gciResult_Ok)
76      {
77          printf("GCI Error in gciSignUpdate: RSA");
78      }
79
80      /* Generation of the signature */
81      err = gciSignGenFinish(rsaCtxID, a_signature, &signLen);
82
83      /* Error coming from the generation of a RSA-signature */
84      if(err != en_gciResult_Ok)
85      {
86          printf("GCI Error in gciSignGenFinish: RSA");
87      }
88
89      else
90      {
91          printf("GCI Info: Signature = ");
92          for(i = 0; i < signLen; i++)
93          {
94              printf("%d", a_signature[i]);
95          }
96      }
97
98      /* Delete the context */
99      gciCtxRelease(rsaCtxID);
100
101 }
```

6. Generation of key pair

6.1. Configuration of a key pair

Algorithm	Parameter
RSA	en_gciKeyPairType_RSA
DSA	en_gciKeyPairType_DSA
ECDSA	en_gciKeyPairType_ECDSA

Table 6.1.: Key pair types (en_gciKeyPairType_t)

Parameter	Type
keyType	en_gciKeyPairType_t
hash	en_gciHashAlgo_t
keyPairParamRsa (If RSA is used as key pair type)	st_gciRsaKeyGenConfig_t*
keyPairParamEcdsa (If ECDSA is used as key pair type)	st_gciNamedCurve_t
keyPairParamDsa (If DSA is used as key pair type)	st_gciDsaDomainParam_t*

Table 6.2.: Configuration of key pair types (st_gciKeyPairConfig_t))

6.2. Prototype

```
1 en_gciResult_t gciKeyPairGen( const st_gciKeyPairConfig_t* p_keyConf,
2 GciKeyId_t* p_pubKeyID, GciKeyId_t* p_privKeyID );
```

Direction	Type	Parameter	Definition
Input	st_gciKeyPairConfig_t*	p_KeyConfig	Pointer to the configuration of the key pair
Output	GciKeyId_t*	p_pubKeyID	Pointer to the ID of the public key
Output	GciKeyId_t*	p_privKeyID	Pointer to the ID of the private key

Table 6.3.: Parameters for the generation of a key pair

RSA

Parameter	Configuration	Exception(s)
keyType	en_gciKeyPairType_Rsa	-
modulusLen	size_t	less or equal than 0

Table 6.4.: Configuration of RSA key pair (st_gciKeyPairConfig_t)

Digital Signature Algorithm (DSA)

Parameter	Configuration	Exception(s)
keyType	en_gciKeyPairTypeDsa	-
param	st_gciDsaDomainParam_t	less or equal than 0

Table 6.5.: Configuration of DSA key pair (st_gciKeyPairConfig_t)

Elliptic Curve Digital Signature Algorithm (ECDSA)

Parameter	Configuration	Exception(s)
keyType	en_gciKeyPairType_Ecdsa	-
param	en_gciNamedCurve_t	en_gciNamedCurve_Invalid

Table 6.6.: Configuration of ECDSA key pair (st_gciKeyPairConfig_t)

6.3. Steps to generate a key pair

```
#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;

    /* Configuration of a RSA key pair */
    st_gciKeyPairConfig_t rsaConf = {.keyType = en_gciKeyPairType_Rsa
    };
    /* Length of the RSA modulus */
    size_t rsaModLen = 1024;
    rsaConf.un_keyPairParam.keyPairParamRsa->modulusLen = &rsaModLen;


    /* ID of the RSA key pair */
    GciKeyId_t rsaPubKeyID = -1;
    GciKeyId_t rsaPrivKeyID = -1;

    /* Generate the RSA key pair */
```

```
25      err = gciKeyPairGen(&rsaConf, &rsaPubKeyID, &rsaPrivKeyID);
26
27      /* Error coming from the generation of a RSA key pair */
28      if(err != en_gciResult_Ok)
29      {
30          printf("GCI Error in gciKeyPairGen: RSA");
31      }
32  }
```

7. Cipher algorithms

7.1. Configuration of cipher algorithms

Algorithm	Parameter
RC4	en_gciCipherAlgo_Rc4
DES	en_gciCipherAlgo_Des
3DES	en_gciCipherAlgo_3des
AES	en_gciCipherAlgo_Aes
RSA	en_gciCipherAlgo_Rsa

Table 7.1.: Cipher algorithms (en_gciCipherAlgo_t)

Parameter	Type
algo	en_gciCipherAlgo_t
blockMode (If DES, 3DES or AES is used as algorithm)	en_gciBlockMode_t
padding (If RSA is used as algorithm)	en_gciPadding_t
iv (If DES, 3DES or AES is used as algorithm)	st_gciBuffer_t

Table 7.2.: Configuration of cipher algorithms (st_gciCipherConfig_t))

7.2. Prototypes

7.2.1. Creation of a context

```
1 en_gciResult_t gciCipherNewCtx( const st_gciCipherConfig_t* p_ciphConfig,
2 GciKeyId_t keyID, GciCtxId_t* p_ctxID )
```

Direction	Type	Parameter	Definition
Input	st_gciCipherConfig_t*	p_ciphConfig	Pointer to the configuration of the cipher
Input	GciKeyId_t	keyID	ID of the key uses to encrypt/decrypt
Output	GciCtxId_t*	p_ctxID	Pointer to the context's ID

Table 7.3.: Parameters for the creation of a cipher context

For a symmetric cipher, the key, represented here by the ID, should be the same for the two parts of the communication for the encryption and the decryption of data.

For an asymmetric cipher, the key, reprensented here by the ID, must be the public key of the key pair for the encryption and the private key for the decryption of data.

Symmetric stream cipher

Parameter	Configuration	Prohibition(s)
algo	en_gciSignAlgo_Rc4	-
blockMode	en_gciBlockMode_None	-
padding	en_gciPadding_None	-
iv	NULL	-

Table 7.4.: Configuration of symmetric stream ciphers (st_gciCipherConfig_t)

Symmetric block cipher

Parameter	Configuration	Prohibition(s)
algo	en_gciSignAlgo_Des en_gciSignAlgo_3Des en_gciSignAlgo_Aes	en_gciCipherAlgo_Invalid en_gciCipherAlgo_None en_gciCipherAlgo_Rc4 en_gciCipherAlgo_Rsa
blockMode	en_gciBlockMode_t	en_gciBlockMode_Invalid en_gciBlockMode_None
padding	en_gciPadding_None	-
iv	st_gciBuffer_t	less or equal than 0

Table 7.5.: Configuration of symmetric block ciphers (st_gciCipherConfig_t)

Asymmetric cipher

Parameter	Configuration	Prohibition(s)
algo	en_gciSignAlgo_Rsa	-
blockMode	en_gciBlockMode_None	-
padding	en_gciPadding_t	en_gciPadding_Invalid en_gciPadding_None
iv	NULL	-

Table 7.6.: Configuration of asymmetric ciphers (st_gciCipherConfig_t)

7.2.2. Encryption a plaintext

```
1 en_gciResult_t gciCipherEncrypt( GciCtxId_t ctxId, const uint8_t* p_plaintxt,
2 size_t pltxtLen, uint8_t* p_ciphtxt, size_t* p_cptxtLen )
```

Direction	Type	Parameter	Definition
Input	GciCtxId_t	ctxID	Context's ID
Input	uint8_t*	p_plaintext	Pointer to the plaintext to encrypt
Input	size_t	pltxtLen	Length of the plaintext to encrypt
Output	uint8_t*	p_ciphtxt	Pointer to the ciphertext
Output	size_t*	p_cptxtLen	Pointer to the length of the ciphertext

Table 7.7.: Parameters for the encryption of a plaintext

7.2.3. Decryption a ciphertext

```
en_gciResult_t gciCipherDecrypt( GciCtxId_t ctxId, const uint8_t* p_ciphtxt,
size_t cptxtLen, uint8_t* p_plaintxt, size_t* p_pltxtLen );
```

Direction	Type	Parameter	Definition
Input	GciCtxId_t	ctxID	Context's ID
Input	uint8_t*	p_ciphtxt	Pointer to the ciphertext to decrypt
Input	size_t	cptxtLen	Length of the ciphertext to decrypt
Output	uint8_t*	p_plaintext	Pointer to the plaintext
Output	size_t*	p_pltxtLen	Pointer to the length of the plaintext

Table 7.8.: Parameters for the decryption of a ciphertext

7.3. Step to encrypt and decrypt data

For this example the keys have already been added previously to the interface and an ID returned.

```

#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;

    int i;

    uint8_t a_pltxt[20] = {"Data to encrypt"};
    size_t pltxtLen = strlen(a_pltxt);

    uint8_t a_ciphtxt[50];
    memset(a_ciphtxt, 0, 50);
    size_t ciphtxtLen = 0;

    /* Configuration of a RSA cipher uses for encryption */
    st_gciCipherConfig_t rsaConfEnc = {.algo = en_gciCipherAlgo_Rsa,
```

```
        .blockMode = en_gciBlockMode_None,
        .padding = en_gciPadding_Pkcs1_Emsa,
        .iv = NULL
    };

    /* Context's ID */
    GciCtxId_t rsaEncCtxID = -1;
    GciCtxId_t rsaDecCtxID = -1;

    /* Creation of the cipher context */
    err = gciCipherNewCtx(&rsaConfEnc, rsaPubKeyID, &rsaEncCtxID);

    /* Error coming from the creation of a RSA cipher context */
    if(err != en_gciResult_Ok)
    {
        printf("GCI Error in gciCipherNewCtx: RSA");
    }

    /* Encrytion of the plaintext */
    err = gciCipherEncrypt(rsaEncCtxID, a_pltxt, pltxtLen, a_ciphtxt, &ciphtxtLen);

    /* Error coming from the encryption with a RSA cipher context */
    if(err != en_gciResult_Ok)
    {
        printf("GCI Error in gciCipherEncrypt: RSA");
    }

    else
    {

        printf("Ciphertext:");

        for(i=0; i<ciphtxtLen; i++)
        {
            printf("%d", a_ciphtxt[i]);
        }
    }


    /* Configuration of a RSA cipher uses for decryption */
    st_gciCipherConfig_t rsaConfDec = {.algo = en_gciCipherAlgo_Rsa,
        .blockMode = en_gciBlockMode_None,
        .padding = en_gciPadding_Pkcs1_Emsa,
        .iv = NULL
    };

    memset(a_pltxt, 0, 50);
    pltxtLen = 0;


    /* Creation of the cipher context */
    err = gciCipherNewCtx(&rsaConfDec, rsaPrivKeyID, &rsaDecCtxID);

    /* Error coming from the creation of a RSA cipher context */
    if(err != en_gciResult_Ok)
    {
        printf("GCI Error in gciCipherNewCtx: RSA");
    }

    /* Decryption of the ciphertext */
```

```
83      err = gciCipherDecrypt(rsaDecCtxID, a_ciphtxt, ciphtxtLen, a_pltxt, &pltxtLen);
84
85      /* Error coming from the decryption with a RSA cipher context */
86      if(err != en_gciResult_Ok)
87      {
88          printf("GCI Error in gciCipherDecrypt: RSA");
89      }
90
91      else
92      {
93
94          printf("Plaintext:");
95
96          for(i=0; i<pltxtLen; i++)
97          {
98              printf("%d", a_pltxt[i]);
99          }
100     }
101
102 }
```

8. Generation of Diffie-Hellmann key pair

8.1. Configuration of a Diffie-Hellmann key pair

Algorithm	Parameter
Diffie-Hellman	en_gciDhType_Dh
Elliptic Curve Diffie-Hellman	en_gciDhType_Ecdh

Table 8.1.: Diffie-Hellman type (en_gciDhType_t)

Parameter	Type
type	en_gciDhType_t
dhParamDomain (If Diffie-Hellman is used as Diffie-Hellman type)	st_gciDhDomainParam_t
dhParamCurveName (If Elliptic Curve Diffie-Hellman is used as Diffie-Hellman type)	en_gciNamedCurve_t

Table 8.2.: Configuration of Diffie-Hellman algorithms (st_gciDhConfig_t))

8.2. Prototypes

8.2.1. Creation of a context

```
1 en_gciResult_t gciDhNewCtx( const st_gciDhConfig_t* p_dhConfig, GciCtxId_t*
2 p_ctxID )
```

Direction	Type	Parameter	Definition
Input	st_gciDhConfig_t*	p_dhConfig	Pointer to the configuration of a Diffie-Hellman type
Output	GciCtxId_t*	ctxID	Pointer to the context's ID

Table 8.3.: Parameters for the creation of a Diffie-Hellman context

Diffie-Hellmann (DH)

Parameter	Configuration	Prohibition(s)
type	en_gciDhType_Dh	-
dhParamDomain	st_gciDhDomainParam_t	NULL
dhParamCurveName	NULL	-

Table 8.4.: Configuration of Diffie-Hellman algorithms (st_gciDhConfig_t)

Elliptic Curve Diffie Hellmann (ECDH)

Parameter	Configuration	Prohibition(s)
type	en_gciDhType_Ecdh	-
dhParamDomain	NULL	-
dhParamCurveName	en_gciNamedCurve_t	en_gciNamedCurve_Invalid

Table 8.5.: Configuration of Elliptic curve Diffie-Hellman algorithms (st_gciDhConfig_t)

8.2.2. Generate a key pair

When a Diffie-Hellman key pair is generated, only the ID of the public key is returned. The private key is saved in the context and is not possible to get it.

```
1 en_gciResult_t gciDhGenKey( GciCtxId_t ctxID, GciKeyId_t* p_pubKeyID )
```

Direction	Type	Parameter	Definition
Input	GciCtxId_t	ctxID	Context's ID
Output	GciKeyId_t*	p_pubKeyID	Pointer to the ID of the generated public key

Table 8.6.: Parameters for the generation of Diffie-Hellman key pairs

8.2.3. Calculation of a shared secret key

```
1 en_gciResult_t gciDhCalcSharedSecret( GciCtxId_t ctxID, GciKeyId_t pubKeyID,
2 GciKeyId_t* p_secretKeyID )
```

Direction	Type	Parameter	Definition
Input	GciCtxId_t	ctxID	Context's ID
Input	GciKeyId_t	pubKeyID	ID of the public key
Output	GciKeyId_t*	p_secretKeyID	Pointer to the ID of the calculated secret key

Table 8.7.: Parameters for the calculation of Diffie-Hellman secret keys

8.3. Steps to generate a Diffie-Hellmann key pair and secret key

For this example the keys have already been added previously to the interface and an ID returned.

```
1
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include <string.h>
5  #include "crypto_iface.h"
6
7  int main(int argc , char *argv[])
8  {
9
10     /* Error management */
11     en_gciResult_t err;
12
13     int i;
14
15     /* Configuration of the DH */
16     st_gciDhConfig_t dhConf = {.type = en_gciDhType_Dh};
17
18     /* To generate the domain parameter */
19     dhConf.un_dhParam.dhParamDomain = NULL;
20
21     /* Context's ID */
22     GciCtxId_t dhCtxID = -1;
23
24     /* DH key ID */
25     GciKeyId_t dhPubKeyID = -1;
26     GciKeyId_t dhSecKeyID = -1;
27
28     /* Creation of a new context */
29     err = gciDhNewCtx(&dhConf, &dhCtxID);
30
31     /* Error coming from the creation of a DH context */
32     if(err != en_gciResult_Ok)
33     {
34         printf("GCI Error in gciDhNewCtx: DH");
35     }
36
37     /* Generate the key pair */
38     err = gciDhGenKey(dhCtxID, &dhPubKeyID);
39
40     /* Error coming from the generation of DH key pair */
41     if(err != en_gciResult_Ok)
42     {
43         printf("GCI Error in gciDhGenKey: DH");
44     }
45
46     err = gciDhCalcSharedSecret(dhCtxID, dhPeerPubKeyID, &dhSecKeyID);
47
48     /* Error coming from the calculation of the DH secret key */
49     if(err != en_gciResult_Ok)
50     {
51         printf("GCI Error in gciDhCalcSharedSecret: DH");
52     }
53
54 }
```

9. Random Number Generator

9.1. Seed a pseudo-random number

```
en_gciResult_t gciRngSeed( const uint8_t* p_sdBuf, size_t sdLen )
```

Direction	Type	Parameter	Definition
Input	uint8_t*	p_sdBuf	Pointer to the buffer of the seed
Input	size_t	sdLen	Pointer to the buffer of randomcharacters

Table 9.1.: Parameters for the seed of pseudo-random numbers

9.2. Generation of a random number

```
en_gciResult_t gciRngGen( int rdmNb, uint8_t* p_rdmBuf )
```

Direction	Type	Parameter	Definition
Input	int	rdmNb	Number of random numbers to generate
Output	uint8_t*	p_secretKeyID	Pointer to the buffer of random characters

Table 9.2.: Parameters for the generation of random numbers

10. Key management

The hardware-based cryptographic modules use a key management. This institute wants to use this key management, that's why the interface shall have a key management too.

10.1. Save a key and get an ID

There is two possibilities about the value of the given ID. Either the ID is previously initialized to -1, so will the smallest value free for an ID returned, or the ID can be initialized to a value greater or equal to 0 to be the same as a context ID for example, so if this value is free, the key will be saved at this specified ID.

```
en_gciResult_t gciKeyPut( const st_gciKey_t* p_key, GciKeyId_t* p_keyID )
```

Direction	Type	Parameter	Definition
Input	st_gciKey_t*	p_key	Pointer to the key to save
Output	GciKeyId_t*	p_keyID	Pointer to the ID of the saved key

Table 10.1.: Parameters for to put a key

10.2. Get of a saved key with its ID

```
en_gciResult_t gciKeyGet( GciKeyId_t keyID, st_gciKey_t* p_key )
```

Direction	Type	Parameter	Definition
Input	GciKeyId_t	p_key	ID of the key to get
Output	st_gciKey_t*	p_keyID	Pointer to the saved key

Table 10.2.: Parameters for to get a key

10.3. Delete a key

```
en_gciResult_t gciKeyDelete( GciKeyId_t keyID  )
```

Direction	Type	Parameter	Definition
Input	GciKeyId_t	p_key	ID of the key to delete

Table 10.3.: Parameters for to delete a key

Bibliography

[1] S. Blake-Wilson, N. Bolyard, V. Gupta, C. Hawk, and B. Moeller. Elliptic curve cryptography (ecc) cipher suites for transport layer security (tls). RFC 4492, RFC Editor, May 2006. `http://www.rfc-editor.org/rfc/rfc4492.txt`.

[2] 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.

Appendix

Appendix A.

Other parameters

A.1. Global types

Name	Type	Definition
GciCtxId_t	int	Context's ID

Table A.1.: Context's ID (st_gciCtxId_t)

Name	Type	Definition
GciKeyId_t	int	Key's ID

Table A.2.: Key's ID (st_gciKeyId_t)

Name	Type	Definition
len	size_t	Length of the big number (in bytes)
data	uint8_t*	Big numbre (data)

Table A.3.: Parameters of a big number (st_gciBigInt_t)

Name	Type	Definition
len	size_t	Length of the buffer (in bytes)
data	uint8_t*	Buffer (data)

Table A.4.: Parameters of a buffer (st_gciBuffer_t)

A.2. Managements

Name	Parameter
No errors	en_gciResult_Ok
Error(s)	en_gciResult_Err

Table A.5.: Error management (en_gciResult_t)

Name	Parameter
Invalid Information	en_gciInfo_Invalid
Name of the available curves	en_gciInfo_CurveName

Table A.6.: Information management (en_gciInfo_t)

A.3. Block modes

Name	Parameter
Invalid Block mode	en_gciBlockMode_Invalid
CBC	en_gciBlockMode_Cbc
ECB	en_gciBlockMode_Ecb
CFB	en_gciBlockMode_Cfb
OFB	en_gciBlockMode_Ofb
GCM	en_gciBlockMode_Gcm
No block mode	en_gciBlockMode_None

Table A.7.: Block modes (en_gciBlockMode_t)

A.4. Paddings

Name	Parameter
Invalid Padding	en_gciPadding_Invalid
ISO 9797	en_gciPadding_Iso9797
PKCS1 V1.5	en_gciPadding_Pkcs1_V1_5
PKCS1 EMSA	en_gciPadding_Pkcs1_Emsa
PKCS5	en_gciPadding_Pkcs5
PKCS7	en_gciPadding_Pkcs7
No padding	en_gciPadding_None

Table A.8.: Paddings (en_gciPadding_t)

A.5. Domain parameters

Name	Type	Definition
p	st_gciBigInt_t	Prime number
q	st_gciBigInt_t	Divisor
g	st_gciBigInt_t	Generator

Table A.9.: DSA domain parameters (st_gciDsaDomainParam_t)

Name	Type	Definition
p	st_gciBigInt_t	Prime number
g	st_gciBigInt_t	Generator

Table A.10.: Diffie-Hellman domain parameters (st_gciDhDomainParam_t)

Name	Type	Definition
a	st_gciBigInt_t	Coefficient a
b	st_gciBigInt_t	Coefficient b
g	st_gciEcPoint_t	Generator
p	st_gciBigInt_t	Prime number group
N	st_gciBigInt_t	Elliptic curve group
h	st_gciBigInt_t	Subgroup of ellitptic curve group generated by g (generator)

Table A.11.: Coordinate of Elliptic Curves (st_gciEcDomainParam_t)

A.6. Elliptic curves

Name	Type	Definition
x	st_gciBigInt_t	x-coordinate
y	st_gciBigInt_t	y-coordinate

Table A.12.: Coordinate of Elliptic Curves (st_gciEcPoint_t)

Name	Parameter
Invalid Curve	en_gciNamedCurve_Invalid
SECT163K1	en_gciNamedCurve_Sect163K1
SECT163R1	en_gciNamedCurve_Sect163R1
SECT163R2	en_gciNamedCurve_Sect163R2
SECT193R1	en_gciNamedCurve_Sect193R1
SECT193R2	en_gciNamedCurve_Sect193R2
SECT233K1	en_gciNamedCurve_Sect233K1
SECT233R1	en_gciNamedCurve_Sect233R1
SECT239K1	en_gciNamedCurve_Sect239K1
SECT283K1	en_gciNamedCurve_Sect283K1
SECT283R1	en_gciNamedCurve_Sect283R1
SECT409K1	en_gciNamedCurve_Sect409K1
SECT409R1	en_gciNamedCurve_Sect409R1
SECT571K1	en_gciNamedCurve_Sect571K1
SECT571R1	en_gciNamedCurve_Sect571R1
SECT160K1	en_gciNamedCurve_Sect160K1
SECT160R1	en_gciNamedCurve_Sect160R1
SECT160R2	en_gciNamedCurve_Sect160R2
SECT192K1	en_gciNamedCurve_Sect192K1
SECT192R1	en_gciNamedCurve_Sect192R1
SECT224K1	en_gciNamedCurve_Sect224K1
SECT224R1	en_gciNamedCurve_Sect224R1
SECT256K1	en_gciNamedCurve_Sect256K1
SECT256R1	en_gciNamedCurve_Sect256R1
SECT384R1	en_gciNamedCurve_Sect384R1
SECT512R1	en_gciNamedCurve_Sect512R1
BRAINPOOLP256R1	en_gciNamedCurve_BrainpoolP256R1
BRAINPOOLP384R1	en_gciNamedCurve_BrainpoolP384R1
BRAINPOOLP512R1	en_gciNamedCurve_BrainpoolP512R1

Table A.13.: Elliptic curves [1] (en_gciNamedCurve_t)

A.7. Configurations

Name	Type	Definition
padding	st_gcigciPadding_t	Padding

Table A.14.: Configuration of RSA Signature Scheme Algorithms (st_gciSignRsaConfig_t))

Parameter	Type	Definition
block	en_gciBlockMode_t	Block mode
padding	en_gciPadding_t	Padding
iv	en_gciBuffer_t	Initial Vector

Table A.15.: Configuration of cipher-based MAC (st_gciSignCmacConfig_t))

Parameter	Type	Definition
modulusLen	size_t	Length of the modulus

Table A.16.: Configuration of RSA key (st_gciRsaKeyGenConfig_t))

A.8. Keys

Name	Type	Definition
n	st_gciBigInt_t	Prime number
e	st_gciBigInt_t	Public exponent

Table A.17.: RSA public key (st_gciRsaPubKey_t)

Name	Type	Definition
p	st_gciBigInt_t	First prime number
q	st_gciBigInt_t	Second prime number
dP	st_gciBigInt_t	dP = d mod (p-1)
dQ	st_gciBigInt_t	dQ = d mod (q-1)
qP	st_gciBigInt_t	qP = $q^{-}1$ mod p

Table A.18.: RSA CRT private key (st_gciRsaCrtPrivKey_t)

Name	Type	Definition
n	st_gciBigInt_t	Prime number
d	st_gciBigInt_t	Private exponent
crt	st_gciRsaCrtPrivKey_t*	Private CRT

Table A.19.: RSA private key (st_gciRsaPrivKey_t)

Name	Type	Definition
param	st_gciDsaDomainParam_t*	DSA domain parameters
key	st_gciBigInt_t	Big numbers of the key

Table A.20.: DSA public/private key (st_gciDsaKey_t)

Name	Type	Definition
param	st_gciDhDomainParam_t*	Diffie-Hellman domain parameters
key	st_gciBigInt_t	Big numbers of the key

Table A.21.: Diffie-Hellman public/private key (st_gciDhKey_t)

Name	Type	Definition
curve	st_gciNamedCurve_t*	Elliptic curves
coord	st_gciEcPoint_t	Coorindate (x,y) of the curve

Table A.22.: Elliptic curve Diffie-Hellman public key (st_gciEcdhPubKey_t)

Name	Type	Definition
curve	st_gciNamedCurve_t*	Elliptic curves
key	st_gciBigInt_t	Big numbers of the key

Table A.23.: Elliptic Curve Diffie-Hellman private key (st_gciEcdhPrivKey_t)

Name	Type	Definition
curve	st_gciNamedCurve_t*	Elliptic curves
coord	st_gciEcPoint_t	Coorindate (x,y) of the curve

Table A.24.: Elliptic curve DSA public key (st_gciEcsaPubKey_t)

Name	Type	Definition
curve	st_gciNamedCurve_t*	Elliptic curves
key	st_gciBigInt_t	Big numbers of the key

Table A.25.: Elliptic Curve DSA private key (st_gciEcdhPrivKey_t)