
# GNTAPI

## Application Programming Interface in ANSI C

*January 2006*

SOFTIDEA

# Table of Contents

# General functions

## GNT_GetTokenInfo

retrieves the information about Token's resources from the Token. This includes file size and access rights of MEM1/2/3 areas, number of RSA slots and their attributes, etc.

### Syntax

```
DWORD GNT_GetTokenInfo(
   DWORD                dwTokenId,      token identifier
   PGNT_TOKEN_DATA      pTokenData      output
)
```

**dwTokenId** is the Token's identifier and can be obtained via GNT_GetTokenID function. It serves as a Token's handle.
**pTokenData** points to the structure that contains the information about Token's resources. See the definition of *GNT_TOKEN_DATA*.

### Example

```
BYTE bKeys;
...
GNT_GetTokenInfo(dwTokenId, pTokenData);
bKeys = pTokenData->bRSAKeys;
printf("The Token contains %d RSA keys.\n", bKeys);
...
```

### Return Values

See the *Table of return values*.

### See also

*GNT_GetTokenStatus*.

## GNT_GetTokenStatus

retrieves the login state, the production state and the firmware version from the Token.

### Syntax

```
DWORD GNT_GetTokenStatus (
   DWORD                dwTokenId,      token identifier
   PGNT_TOKEN_STATUS    pTokenStatus    output
)
```

**dwTokenId** is the Token's identifier and can be obtained via *GNT_GetTokenID* function. It serves as a Token's handle.
**pTokenStatus** points to the structure that contains the information about the Token's status. See the definition of *GNT_TOKEN_STATUS*.

### Example

```
...
GNT_GetTokenStatus(dwTokenId, pTokenStatus);
if ((pTokenStatus->LoginState & PIN1_LOGIN) != 0)
  printf("PIN1: logged in");
else
  printf("PIN1: not logged in");
```

 

```
...
```

**Return values**

See the *Table of return values*.

**See also**

*GNT_GetTokenInfo*.

## GNT_GetDriverVersion

returns the information about the version of Token's USB device driver.

**Syntax**

```
BOOLEAN GNT_GetDriverVersion (
   PGNT_TOKEN_STATUS      pVersionInfo        output
)
```

**pVersionInfo** points to the structure that contains the information about driver's version. See the definition of *GNT_TOKEN_STATUS*.

**Example**

```
...
GNT_TOKEN_STATUS    FrVer;
GNT_TOKEN_STATUS    DrvVer;
PGNT_TOKEN_STATUS pFwVer = &FwVer;
PGNT_TOKEN_STATUS pDrvVer = &DrvVer;
...
GNT_GetTokenStatus(dwTokenId, pFwVer);
GNT_GetDriverVersion(pDrvVer);
printf("Firmware version: %d.%d\n",pFwVer->HiVersion,
     pFwVer->LoVersion);
printf("Driver version: %d.%d\n",pDrvVer->HiVersion,
     pDrvVer->LoVersion);
...
```

**Return values**

Returns TRUE, if the version information was successfully retrieved, FALSE otherwise.

**See also**

*GNT_GetTokenStatus*.

## GNT_GetDevices

returns the number of available GNT Tokens, plugged to the computer's USB ports.

**Syntax**

```
DWORD GNT_GetDevices (
)
```

this function has no input parameters.

**Example**

```
...
DWORD dwTokens;
...
dwTokens = GNT_GetDevices();
printf("%d GNT Tokens available\n",(int)dwTokens);
```

 

```
...
```

**Return values**

Returns number of attached GNT Tokens.

**See also**

*GNT_GetTokenID*.

## GNT_GetTokenID

obtains the Token's identifier that is needed to address concrete Token by almost all of the GNT API functions.

**Syntax**

```
DWORD GNT_GetTokenID (
   DWORD                iTokenIndex        token identifier
)
```

**iTokenIndex** is the index of the Token, counting from 0 to the number of all available Tokens minus 1.

**Example**

```
DWORD dwDevCnt;
DWORD dwTokenId;
...
dwDevCnt = GNT_GetDevices();
for (int i=0;i<(int)dwDevCnt;i++)
{
  printf("Token No. %d ",i);
  dwTokenId = GNT_GetTokenID(i);
  printf("Token ID: 0x%.8x\n",dwTokenId);
  ...
}
...
```

**Return values**

Returns the Token identifier.

**See also**

*GNT_GetDevices*.

## GNT_GetSerialNum

obtains the serial number from the specified Token.

**Syntax**

```
DWORD GNT_GetSerialNum (
   DWORD                dwTokenId,         token identifier
   PGNT_SERIAL_NUM      pSerNum            output
)
```

**dwTokenId** is Token's identifier and can be obtained via *GNT_GetTokenID* function. It serves as a Token's handle.

**pSerNum** points to the structure where the serial number is stored. See the definition of the *GNT_SERIAL_NUM*.

**Example**

 

```
GNT_SERIAL_NUM SerNo;
PGNT_SERIAL_NUM pSerNo = &SerNo;
...
GNT_GetSerialNum(dwTokenId, pSerNo);
...
```

**Return values**

See the *Table of return values*.

**See also**

*GNT_GetTokenStatus*, *GNT_GetTokenInfo*.

 

# Login functions

## GNT_UserLogin

logs a user into the Token.

**Syntax**

```
DWORD GNT_UserLogin (
   DWORD       dwTokenId,    token identifier
   PASSWORD    PIN1,         parameter
   PASSWORD    PIN2          parameter
)
```

dwTokenId is the Token's identifier and can be obtained via *GNT_GetTokenID* function. It serves as a Token's handle.
PIN1 is the 16 bytes long 1st level user's access password. If PIN1 is provided, PIN2 should be NULL. The password input is not limited to characters, the whole range of byte values 0-255 is supported.
PIN2 is the 16 bytes long 2nd level user's access password. If PIN2 is provided, PIN1 should be NULL. The password input is not limited to characters, the whole range of byte values 0-255 is supported.

**Example**

```
DWORD dwRes;
...
dwRes = GNT_UserLogin(dwTokenId, "mypin1", NULL);
if (dwRes == GNT_STATUS_SUCCESS)
  printf("Login successful\n");
else
  printf("Login failed\n");
...
```

**Return values**

See the *Table of return values*.

**See also**

*GNT_AdminLogin*, *GNT_Logout*, *GNT_SetUserPIN1*.

## GNT_AdminLogin

logs an administrator into the Token.

**Syntax**

```
DWORD GNT_AdminLogin (
   DWORD       dwTokenId,    token identifier
   PASSWORD    APW           parameter
)
```

**dwTokenId** is the Token's identifier and can be obtained via *GNT_GetTokenID* function. It serves as a Token's handle.
**APW** is the 16 bytes long administrator's access password. The password input is not limited to characters, the whole range of byte values 0-255 is supported.

**Example**

```
DWORD dwRes;
...
dwRes = GNT_AdminLogin(dwTokenId, "current_apw");
if (dwRes == GNT_STATUS_SUCCESS)
  printf("Login successful\n");
else
  printf("Login failed\n");
...
```

**Return values**

See the *Table of return values*.

**See also**

*GNT_UserLogin*, *GNT_Logout*, *GNT_SetAdminPW*.

## GNT_Logout

logs all logged-in users out of the Token. It includes user's PIN1, PIN2 levels and administrator's level too.

**Syntax**

```
DWORD GNT_Logout (
  DWORD                dwTokenId        token identifier
)
```

**dwTokenId** is the Token's identifier and can be obtained via *GNT_GetTokenID* function. It serves as a Token's handle.

**Example**

```
GNT_UserLogin(dwTokenId,"userpin1",NULL);
GNT_UserLogin(dwTokenId,NULL,"userpin2");
... do some stuff ...
GNT_Logout(dwTokenId);
// now both PIN1 and PIN2 are logged out
```

**Return values**

See the *Table of return values*.

**See also**

*GNT_UserLogin*, *GNT_AdminLogin*.

## GNT_UserPIN2Logout

logs only user's PIN2 level out of the Token. User's PIN1 and administrator's login states are untouched. PIN2 level can be designed for higher security access, therefore it's convenient for the user to logout of this level if the higher security access is no more desired.

**Syntax**

```
DWORD GNT_UserPIN2Logout (
  DWORD                dwTokenId        token identifier
)
```

**dwTokenId** is the Token's identifier and can be obtained via *GNT_GetTokenID* function. It serves as a Token's handle.

**Example**

 

```
...
GNT_UserLogin(dwTokenId,NULL,"user_pin2");
... PIN2 level access operations
GNT_UserPIN2Logout(dwTokenId);
...
```

**Return values**

See the *Table of return values*.

**See also**

*GNT_UserLogin*, *GNT_Logout*.

## GNT_AdminLogout

logs an administrator out of the Token.

**Syntax**

```
DWORD GNT_AdminLogout (
   DWORD          dwTokenId      token identifier
)
```

**dwTokenId** is the Token's identifier and can be obtained via *GNT_GetTokenID* function. It serves as a Token's handle.

**Example**

```
...
GNT_AdminLogin(dwTokenId,"current_apw");
... some admin related operations
GNT_AdminLogout(dwTokenId);
...
```

**Return values**

See the *Table of return values*.

**See also**

*GNT_AdminLogin*, *GNT_Logout*.

## GNT_SetUserPIN1

modifies the user's PIN1. A user has to be logged in with current PIN1 or enter correct PIN2.

Syntax

```
DWORD GNT_SetUserPIN1 (
   DWORD          dwTokenId,     token identifier
   PASSWORD       NewPIN1,       input
   PASSWORD       PIN2           parameter
)
```

**dwTokenId** is the Token's identifier and can be obtained via *GNT_GetTokenID* function. It serves as a Token's handle.
**NewPIN1** is the new user's PIN1 password.
**PIN2** is the current user's PIN2 password. It is required only when a user is currently not logged with PIN1 password, otherwise this parameter could be NULL. See example below.

**Example**

```
DWORD dwRes;
```

 

```
...
dwRes = GNT_UserLogin(dwTokenId,"current_pin1",NULL);
if (dwRes == GNT_STATUS_SUCCESS)
{
  GNT_SetUserPIN1(dwTokenId,"new_pin1",NULL);
}
else
{
  GNT_SetUserPIN1(dwTokenId,"new_pin1","current_pin2");
}
...
```

### Return values

See the *Table of return values*.

### See also

*GNT_UserLogin*, *GNT_SetUserPIN2*, *GNT_SetAdminPW*.

## GNT_SetUserPIN2

modifies the user's PIN2.

### Syntax

```
DWORD GNT_SetUserPIN2 (
   DWORD       dwTokenId,    token identifier
   PASSWORD    NewPIN2,      input
   PASSWORD    PIN2          parameter
)
```

**dwTokenId** is the Token's identifier and can be obtained via *GNT_GetTokenID* function. It serves as a Token's handle.
**NewPIN2** is the new user's PIN2 password.
**PIN2** is the currently valid PIN2 password.

### Example

```
...
GNT_SetUserPIN1(dwTokenId,"new_pin2","current_pin2");
GNT_UserLogin(dwTokenId,NULL,"new_pin2");
...
```

### Return values

See the *Table of return values*.

### See also

*GNT_UserLogin*, *GNT_SetUserPIN1*, *GNT_SetAdminPW*.

## GNT_SetAdminPW

modifies the administrator's access password.

### Syntax

```
DWORD GNT_SetAdminPW (
   DWORD       dwTokenId,    token identifier
   PASSWORD    NewAPW,       input
   PASSWORD    APW           parameter
)
```

 

**dwTokenId** is the Token's identifier and can be obtained via *GNT_GetTokenID* function. It serves as a Token's handle.
**NewAPW** is the new administrator's password.
**APW** is the currently valid administrator's password.

### Example

```
...
GNT_SetAdminPW(dwTokenId,"new_apw","current_apw");
GNT_AdminLogin(dwTokenId,"new_apw");
...
```

### Return values

See the *Table of return values*.

### See also

*GNT_AdminLogin*, *GNT_SetUserPIN1*, *GNT_SetUserPIN2*.

 

# Cryptographic functions

## GNT_GetRandom

obtains a sequence of random bytes from the Token.

**Syntax**

```
DWORD GNT_GetRandom (
   DWORD          dwTokenId,     token identifier
   PVOID          pBuffer,       output
   DWORD          dwLen          parameter
)
```

**dwTokenId** is the Token's identifier and can be obtained via *GNT_GetTokenID* function. It serves as a Token's handle.
**pBuffer** points to the buffer in computer's memory, where the random data are copied to.
**dwLen** is the length of random data.

**Example**

```
...
// obtain 31 random bytes into 'buff'
GNT_GetRandom(dwTokenId,buff,31);
...
```

**Return values**

See the *Table of return values*.

## GNT_GenerateKeyPairRSA

generates an RSA keypair in the Token.

**Syntax**

```
DWORD GNT_GenerateKeyPairRSA (
   DWORD          dwTokenId,     token identifier
   PGNT_RSA_KEY   pRSAKey,       output
   BYTE           bRSAslot       parameter
)
```

**dwTokenId** is the Token's identifier and can be obtained via *GNT_GetTokenID* function. It serves as a Token's handle.
**pRSAKey** points to the buffer in computer's memory where the generated RSA key is exported after it is generated. See the definition of *GNT_RSA_KEY*. The RSA key can be exported out of the Token only if the relevant RSA slot is marked as GNT_RSA_ATTR_EXPORTABLE_BY_GENER_ONLY or GNT_RSA_ATTR_EXPORTABLE.
**bRSAslot** is the index of RSA slot, where the generated RSA key is stored.

**Example**

```
...
GNT_GenerateKeyPairRSA(dwTokenId,NULL,1);
// generate with export of the complete RSA key
GNT_GenerateKeyPairRSA(dwTokenId,pRSAKey,5);
...
```

**Return values**

See the *Table of return values*.

### See also

*GNT_ExportRSAKey*, *GNT_ImportRSAKey*.

## GNT_ExportRSAKey

exports an RSA key from the specified slot.

### Syntax

```
DWORD GNT_ExportRSAKey (
   DWORD          dwTokenId,      token identifier
   PGNT_RSA_KEY   pRSAKey,        output
   BOOLEAN        bCompleteKey,   parameter
   BYTE           bRSAslot        parameter
)
```

**dwTokenId** is the Token's identifier and can be obtained via *GNT_GetTokenID* function. It serves as a Token's handle.
**pRSAKey** points to the buffer in computer's memory where the RSA key is copied to. The RSA key can be exported out of the Token only if the relevant RSA slot is marked as 'exportable' and a user is logged in with corresponding access password. See the definition of *GNT_RSA_KEY*.
**bCompleteKey** is the boolean parameter that defines whether complete RSA key (*bCompleteKey = TRUE)* or only public key will be exported (*bCompleteKey = FALSE*).
**bRSAslot** is the index of RSA slot, where the exported RSA key is stored.

### Example

```
...
// export complete RSA key from the slot 1
GNT_ExportRSAKey(dwTokenId,pRSAKey,TRUE,1);
...
// export only public key from the slot 12
GNT_ExportRSAKey(dwTokenId,pRSAKey,FALSE,12);
...
```

### Return values

See the *Table of return values*.

### See also

*GNT_ImportRSAKey*, *GNT_GenerateKeyPairRSA*.

## GNT_ImportRSAKey

imports complete RSA key into a Token.

Syntax

```
DWORD GNT_ImportRSAKey (
   DWORD          dwTokenId,      token identifier
   PGNT_RSA_KEY   pRSAKey,        input
   BYTE           bRSAslot        parameter
)
```

**dwTokenId** is the Token's identifier and can be obtained via *GNT_GetTokenID* function. It serves as a Token's handle.
**pRSAKey** points to the buffer in computer's memory where the RSA key is copied from. The RSA key can be imported into the Token only if the relevant RSA slot is marked as

 

'importable' and a user is logged in with corresponding access password. See the definition of *GNT_RSA_KEY*.

**bRSAslot** is the index of RSA slot, where the imported RSA key is copied into.

**Example**

```
...
GNT_ImportRSAKey(dwTokenId,pRSAKey,4);
...
```

**Return values**

See the *Table of return values*.

**See also**

*GNT_ExportRSAKey*, *GNT_GenerateKeyPairRSA*.

## GNT_EraseRSAKey

erases an RSA key from the specified Token's slot.

**Syntax**

```
DWORD GNT_EraseRSAKey (
   DWORD          dwTokenId,     token identifier
   BYTE           bRSAslot       parameter
)
```

**dwTokenId** is the Token's identifier and can be obtained via *GNT_GetTokenID* function. It serves as a Token's handle.

**bRSAslot** is the index of RSA slot to be erased.

**Example**

```
...
GNT_EraseRSAKey(dwTokenId,2);
...
```

**Return values**

See the *Table of return values*.

**See also**

*GNT_ImportRSAKey*, *GNT_GenerateKeyPairRSA*.

## GNT_Hash

calculates a hash value from input data.

**Syntax**

```
DWORD GNT_Hash (
   DWORD                  dwTokenId,     token identifier
   BYTE *                 pbInData,      input
   DWORD                  dwInLen,       input
   BYTE *                 pHashValue,    output
   PGNT_CRYPT_OPTIONS     pCrOpts        parameter
)
```

**dwTokenId** is the Token's identifier and can be obtained via *GNT_GetTokenID* function. It serves as a Token's handle.

**pbInData** points to the buffer that holds input data for the hash calculation.

**dwInLen** is the length of input data.

 

**pHashValue** points to the buffer where the hash value will be stored.
**pCrOpts** points to the structure that contains the information about the hash algorithm to be used. See the definition of *GNT_CRYPT_OPTIONS* and the *Table of required parameters for GNT_CRYPT_OPTONS*.

### Example

```
...
GNT_Hash(dwTokenId,);
...
```

### Return values

See the *Table of return values*.

### See also

*GNT_SignHash*.

## GNT_EncryptRSA

encrypts a small block with RSA algorithm.

### Syntax

```
DWORD GNT_EncryptRSA (
   DWORD          dwTokenId,     token identifier
   BYTE *         pbInData,      input
   DWORD          dwInLen,       input
   BYTE *         pbOutData,     output
   PGNT_CRYPT_KEY pCrKey         parameter
)
```

**dwTokenId** is the Token's identifier and can be obtained via *GNT_GetTokenID* function. It serves as a Token's handle.
**pbInData** points to the buffer that holds input data for RSA encryption.
**dwInLen** is the length of input data.
**pbOutData** points to the buffer where encrypted data will be stored.
**pCrKey** points to the structure that contains RSA key specification. See the definition of *GNT_CRYPT_KEY* and the *Table of required parameters for GNT_CRYPT_KEY*.

### Example

```
...
GNT_CRYPT_KEY CrKey;
PGNT_CRYPT_KEY pCrKey = &CrKey;
...
pCrKey->bPubKeySpecMethod = GNT_PUB_KEY_SPEC_RSASLOT;
pCrKey->bRSAslotEncrypt = 3;
// encrypt 16 bytes stored in pInput with public key
// stored in RSA slot 3 and the output save in pOutput
GNT_EncryptRSA(dwTokenId, pInput, 16, pOutput, pCrKey);
...
```

### Return values

See the *Table of return values*.

### See also

*GNT_DecryptRSA*.

## GNT_DecryptRSA

decrypts a small block with RSA algorithm.

### Syntax

```
DWORD GNT_DecryptRSA (
   DWORD                dwTokenId,      token identifier
   BYTE *               pbInData,       input
   BYTE *               pbOutData,      output
   BYTE *               pbOutLen,       output
   PGNT_CRYPT_KEY       pCrKey          parameter
)
```

**dwTokenId** is the Token's identifier and can be obtained via *GNT_GetTokenID* function. It serves as a Token's handle.
**pbInData** points to the buffer that holds input data for RSA decryption.
**pbOutData** points to the buffer where decrypted data will be stored.
**pbOutLen** points to the DWORD variable where the length of decrypted data will be stored.
**pCrKey** points to the structure that contains RSA key specification. See the definition of *GNT_CRYPT_KEY* and the *Table of required parameters for GNT_CRYPT_KEY*.

### Example

```
...
GNT_CRYPT_KEY CrKey;
PGNT_CRYPT_KEY pCrKey = &CrKey;
...
pCrKey->bRSAslotDecrypt = 2;
// decrypt block stored in pInput with private key stored
// in RSA slot 2 and result save in pOutput; length of
// decrypted block is stored in pOutLen
GNT_DecryptRSA(dwTokenId, pInput, pOutput, pOutLen,pCrKey);
...
```

### Return values

See the *Table of return values*.

### See also

*GNT_EncryptRSA*.

## GNT_Sign

generates a digital signature from input data.

### Syntax

```
DWORD GNT_Sign (
   DWORD                dwTokenId,      token identifier
   BYTE *               pbInData,       input
   DWORD                dwInLen,        input
   BYTE *               pSignature,     output
   PGNT_CRYPT_OPTIONS   pCrOpts,        parameter
   PGNT_CRYPT_KEY       pCrKey          parameter
)
```

 

**dwTokenId** is the Token's identifier and can be obtained via *GNT_GetTokenID* function. It serves as a Token's handle.
**pbInData** points to the buffer that holds input data to be signed.
**dwInLen** is the length of input data.
**pSignature** points to the buffer where the calculated signature will be stored.
**pCrOpts** points to the structure where the options for the sign operation are defined. See the definition of *GNT_CRYPT_OPTIONS* and the *Table of required parameters for GNT_CRYPT_OPTONS*.
**pCrKey** points to the structure that contains RSA key specification. See the definition of *GNT_CRYPT_KEY* and the *Table of required parameters for GNT_CRYPT_KEY*.

### Example

```
...
GNT_CRYPT_KEY CrKey;
PGNT_CRYPT_KEY pCrKey = &CrKey;
GNT_CRYPT_OPTIONS CrOpt;
PGNT_CRYPT_OPTIONS pCrOpts = &CrOpt;
...
pCrKey->bRSAslotGenSign = 0;
pCrOpt->wHashAlgorithm = GNT_CRYPT_HASH_ALG_SHA1;
// sign 1 kB long file stored in pInput with private key
// stored in RSA slot 0 and result save in pSignature;
//use RSA and SHA1 hash algorithm
GNT_Sign(dwTokenId, pInput, 1024, pSignature, pCrOpt,
     pCrKey);
...
```

### Return values

See the *Table of return values*.

### See also

*GNT_Verify*, *GNT_SignHash*, *GNT_EncryptSign*.

## GNT_Verify

verifies a digital signature of input data.

### Syntax

```
DWORD GNT_Verify (
   DWORD                dwTokenId,      token identifier
   BYTE *               pbInData,       input
   DWORD                dwInLen,        input
   BYTE *               pSignature,     input
   PGNT_CRYPT_OPTIONS   pCrOpts,        parameter
   PGNT_CRYPT_KEY       pCrKey          parameter
)
```

**dwTokenId** is the Token's identifier and can be obtained via *GNT_GetTokenID* function. It serves as a Token's handle.
**pbInData** points to the buffer that holds input data, whose signature has to be verified.
**dwInLen** is the length of input data.
**pSignature** points to the buffer that contains the signature to be verified.
**pCrOpts** points to the structure where the options for the verify operation are defined. See the definition of *GNT_CRYPT_OPTIONS* and the *Table of required parameters for*

 

*GNT_CRYPT_OPTONS*.
**pCrKey** points to the structure that contains RSA key specification. See the definition of *GNT_CRYPT_KEY* and the *Table of required parameters for GNT_CRYPT_KEY*.

### Example

```
...
GNT_Verify(dwTokenId, pInput, 1024, pSig, pCrOpt, pCrKey);
...
```

### Return values

See the *Table of return values*.

### See also

*GNT_Sign*, *GNT_DecryptVerify*.

## GNT_Encrypt

encrypts input data with symmetric algorithm.

### Syntax

```
DWORD GNT_Encrypt (
   DWORD                 dwTokenId,      token identifier
   BYTE *                pbInData,       input
   DWORD                 dwInLen,        input
   BYTE *                pbOutData,      output
   DWORD *               pdwOutLen,      output
   PGNT_CRYPT_OPTIONS    pCrOpts,        parameter
   PGNT_CRYPT_KEY        pCrKey          parameter
)
```

**dwTokenId** is the Token's identifier and can be obtained via *GNT_GetTokenID* function. It serves as a Token's handle.
**pbInData** points to the buffer that holds input data for the symmetric encryption.
**dwInLen** is the length of input data.
**pbOutData** points to the buffer where encrypted data will be stored.
**pdwOutLen** points to the DWORD variable where the length of encrypted data will be stored.
**pCrOpts** points to the structure where the options for the sign operation are defined. See the definition of *GNT_CRYPT_OPTIONS* and the *Table of required parameters for GNT_CRYPT_OPTONS*.
**pCrKey** points to the structure that contains RSA key specification. See the definition of *GNT_CRYPT_KEY* and the *Table of required parameters for GNT_CRYPT_KEY*.

### Example

```
...
GNT_CRYPT_KEY CrKey;
PGNT_CRYPT_KEY pCrKey = &CrKey;
GNT_CRYPT_OPTIONS CrOpt;
PGNT_CRYPT_OPTIONS pCrOpts = &CrOpt;
...
// options
pCrOpt->wCrAlgorithm = GNT_CRYPT_ALG_AES;
pCrOpt->wCrMode = GNT_CRYPT_MODE_CBC;
// sym. key
pCrKey->bKeySpecMethod = GNT_KEY_SPEC_PTRMEM;
```

 

```
pCrKey->PtrToSymKey.bMEMIndex = 2;
pCrKey->PtrToSymKey.wMEMAddress = 0x120;
// encrypt 258 bytes stored in pInput with AES algorithm
// in CBC mode with the symmetric key stored in MEM2 on
// address 0x120; save result to pOutput; the length of
// result will be saved to dwOutLen
GNT_Encrypt(dwTokenId, pInput, 258, pOutput, &dwOutLen,
     pCrOpt, pCrKey);
...
```

### Return values

See the *Table of return values*.

### See also

*GNT_Decrypt*, *GNT_EncryptSign*.

## GNT_Decrypt

decrypts input data with symmetric algorithm.

### Syntax

```
DWORD GNT_Decrypt (
   DWORD                dwTokenId,     token identifier
   BYTE *               pbInData,      input
   DWORD                dwInLen,       input
   BYTE *               pbOutData,     output
   DWORD *              pdwOutLen,     output
   PGNT_CRYPT_OPTIONS   pCrOpts,       parameter
   PGNT_CRYPT_KEY       pCrKey         parameter
)
```

**dwTokenId** is the Token's identifier and can be obtained via *GNT_GetTokenID* function. It serves as a Token's handle.
**pbInData** points to the buffer that holds input data for the symmetric decryption.
**dwInLen** is the length of input data.
**pbOutData** points to the buffer where decrypted data will be stored.
**pdwOutLen** points to the DWORD variable where the length of decrypted data will be stored.
**pCrOpts** points to the structure where the options for the sign operation are defined. See the definition of *GNT_CRYPT_OPTIONS* and the *Table of required parameters for GNT_CRYPT_OPTONS*.
**pCrKey** points to the structure that contains RSA key specification. See the definition of *GNT_CRYPT_KEY* and the *Table of required parameters for GNT_CRYPT_KEY*.

### Example

```
...
GNT_CRYPT_KEY CrKey;
PGNT_CRYPT_KEY pCrKey = &CrKey;
GNT_CRYPT_OPTIONS CrOpt;
PGNT_CRYPT_OPTIONS pCrOpts = &CrOpt;
...
// options
pCrOpt->wCrAlgorithm = GNT_CRYPT_ALG_3DES;
pCrOpt->wCrMode = GNT_CRYPT_MODE_CTR;
```

 

```
// sym. key
pCrKey->bKeySpecMethod = GNT_KEY_SPEC_EXPLICIT;
pCrKey->pbIV = pIV; // init IV
pCrKey->pbKey = pKey; // init Key
// decrypt 33 bytes stored in pInput with 3DES algorithm
// in CTR mode with the symmetric key stored in pKey and
// initialization vector stored in pIV; save result to
// pOutput; the length of result will be saved to
// dwOutLen
GNT_Decrypt(dwTokenId, pInput, 33, pOutput, &dwOutLen,
     pCrOpt, pCrKey);
...
```

### Return values

See the *Table of return values*.

### See also

*GNT_Encrypt*, *GNT_DecryptVerify*.

## GNT_EncryptSign

encrypts and sign input data at once.

### Syntax

```
DWORD GNT_EncryptSign (
   DWORD                  dwTokenId,      token identifier
   BYTE *                 pbInData,       input
   DWORD                  dwInLen,        input
   BYTE *                 pbOutData,      output
   DWORD *                pdwOutLen,      output
   BYTE *                 pSignature,     output
   PGNT_CRYPT_OPTIONS     pCrOpts,        parameter
   PGNT_CRYPT_KEY         pCrKey          parameter
)
```

**dwTokenId** is the Token's identifier and can be obtained via *GNT_GetTokenID* function. It serves as a Token's handle.
**pbInData** points to the buffer that holds input data to be signed and encrypted by symmetric encryption.
**dwInLen** is the length of input data.
**pbOutData** points to the buffer where encrypted data will be stored.
**pdwOutLen** points to the DWORD variable where the length of encrypted data will be stored.
**pSignature** points to the buffer where the calculated signature will be stored.
**pCrOpts** points to the structure where the options for the sign operation are defined. See the definition of *GNT_CRYPT_OPTIONS* and the *Table of required parameters for GNT_CRYPT_OPTONS*.
**pCrKey** points to the structure that contains RSA key specification. See the definition of *GNT_CRYPT_KEY* and the *Table of required parameters for GNT_CRYPT_KEY*.

### Example

```
...
GNT_CRYPT_KEY CrKey;
PGNT_CRYPT_KEY pCrKey = &CrKey;
```

 

```
GNT_CRYPT_OPTIONS CrOpt;
PGNT_CRYPT_OPTIONS pCrOpts = &CrOpt;
...
// options
pCrOpt->wCrAlgorithm = GNT_CRYPT_ALG_DES;
pCrOpt->wCrMode = GNT_CRYPT_MODE_OFB;
pCrOpt->wHashAlgorithm = GNT_CRYPT_HASH_ALG_SHA1;
// sym. key (to encrypt input data)
pCrKey->bKeySpecMethod = GNT_KEY_SPEC_GENERATE;
pCrKey->pbWrappedKeyOut = pWrappedKey;
// private key (to sign input data)
pCrKey->bRSAslotGenSign = 3;
// public key (to wrap generated sym. key)
pCrKey->bPubKeySpecMethod = GNT_PUB_KEY_SPEC_PTRMEM;
pCrKey->PtrToPubKey.bMEMIndex = 1;
pCrKey->PtrToPubKey.wMEMAddress = 0;
// encrypt and sign input data stored in pInput; generate
// symmetric key in the Token as a random number, wrap it
// with public key stored in MEM1 on address 0 and
// wrapped key save to pWrappedKey; input data (128
// bytes) are encrypted with DES algorithm in OFB mode
// and result is saved to pOutput; the length of output
// is saved to dwOutLen; input data are signed with
// RSA+SHA1 algorithm with the private key stored in
// RSA slot 3 and resulting signature is saved to pSig.
GNT_EncryptSign(dwTokenId,pInput,128,pOutput,&dwOutLen,
pSig,pCrOpt,pCrKey);
...
```

### Return values

See the *Table of return values*.

### See also

*GNT_DecryptVerify*, *GNT_Encrypt*.

## GNT_DecryptVerify

decrypts and verifies a digital signature of input data at once.

### Syntax

```
DWORD GNT_DecryptVerify (
   DWORD                dwTokenId,      token identifier
   BYTE *               pbInData,       input
   DWORD                dwInLen,        input
   BYTE *               pSignature,     input
   BYTE *               pbOutData,      output
   DWORD *              pdwOutLen,      output
   PGNT_CRYPT_OPTIONS   pCrOpts,        parameter
   PGNT_CRYPT_KEY       pCrKey          parameter
)
```

**dwTokenId** is the Token's identifier and can be obtained via *GNT_GetTokenID* function. It serves as a Token's handle.

 

**pbInData** points to the buffer that holds input data to be decrypted and whose signature has to be verified.
**dwInLen** is the length of input data.
**pSignature** points to the buffer that contains the signature to be verified.
**pbOutData** points to the buffer where decrypted data will be stored.
**pdwOutLen** points to the DWORD variable where the length of decrypted data will be stored.
**pCrOpts** points to the structure where the options for the sign operation are defined. See the definition of *GNT_CRYPT_OPTIONS* and the *Table of required parameters for GNT_CRYPT_OPTONS*.
**pCrKey** points to the structure that contains RSA key specification. See the definition of *GNT_CRYPT_KEY* and the *Table of required parameters for GNT_CRYPT_KEY*.

### Example

```
...
// options
pCrOpt->wCrAlgorithm = GNT_CRYPT_ALG_DES;
pCrOpt->wCrMode = GNT_CRYPT_MODE_OFB;
pCrOpt->wHashAlgorithm = GNT_CRYPT_HASH_ALG_SHA1;
// sym. key (to decrypt input data)
pCrKey->bKeySpecMethod = GNT_KEY_SPEC_UNWRAP;
pCrKey->pbWrappedKeyIn = pWrappedKey;
// private key (to unwrap generated sym. key)
pCrKey->bRSAslotUnwrap = 2;
// public key (to verify signature of input data)
pCrKey->bPubKeySpecMethod = GNT_PUB_KEY_SPEC_RSASLOT;
pCrKey->bRSAslotVerSign = 0;
// decrypt and verify signature of input data stored in
// pInput; obtain symmetric key by unwrapping supplied
// wrapped key stored in pWrappedKey with private key
// stored in RSA slot 2; input data (128 bytes) are
// decrypted using DES algorithm in OFB mode and result
// is saved to pOutput; the length of output is saved
// to dwOutLen; the signature stored in pSig is verified
// with RSA+SHA1 algorithm with the public key stored in
// RSA slot 0.
dwRes = GNT_DecryptVerify(dwTokenId,pInput,128,pSig,
     pOutput,&dwOutLen,pCrOpt,pCrKey);
if (dwRes == GNT_STATUS_SUCCESS)
     printf(”Input successfully decrypted and verified.\n”);
else
     printf(”Decryption or signature verification
failed.\n”);
...
```

### Return values

See the *Table of return values*.

### See also

*GNT_EncryptSign*, *GNT_Decrypt*.

## GNT_SignHash

generates a digital signature from supplied hash value. The difference between GNT_SignHash and *GNT_Sign* is that GNT_Sign works with plain data and it calculates hash value from input data itself. GNT_SignHash works with hash value already calculated outside this function.

## Syntax

```
DWORD GNT_SignHash (
   DWORD                  dwTokenId,      token identifier
   BYTE *                 pHashValue,     input
   BYTE                   bHashLen,       input
   BYTE *                 pSignature,     output
   PGNT_CRYPT_OPTIONS     pCrOpts,        parameter
   PGNT_CRYPT_KEY         pCrKey          parameter
)
```

**dwTokenId** is the Token's identifier and can be obtained via *GNT_GetTokenID* function. It serves as a Token's handle.
**pHashValue** points to the buffer that holds hash value to be signed.
**bHashLen** is the length of hash value to be signed.
**pSignature** points to the buffer where the calculated signature will be stored.
**pCrOpts** points to the structure where the options for the sign operation are defined. See the definition of *GNT_CRYPT_OPTIONS* and the *Table of required parameters for GNT_CRYPT_OPTONS*.
**pCrKey** points to the structure that contains RSA key specification. See the definition of *GNT_CRYPT_KEY* and the *Table of required parameters for GNT_CRYPT_KEY*.

## Example

```
...
pCrOpt->wHashAlgorithm = GNT_CRYPT_HASH_ALG_SHA1;
pCrKey->bRSAslotGenSign = 4;
// generate digital signature from supplied hash value;
// hash was generated outside this example with SHA1
// algorithm; sign with RSA slot 4
GNT_SignHash(dwTokenId, pHash, 20, pSig, pCrOpt, pCrKey);
...
```

## Return values

See the *Table of return values*.

## See also

*GNT_Sign*, *GNT_Hash*.

 

# Memory functions

## GNT_ReadMEM

reads a block of bytes from the Token's non-volatile memory.

### Syntax

```
DWORD GNT_ReadMEM (
   DWORD          dwTokenId,      token identifier
   PVOID          pBuffer,        output
   BYTE           bMEMIndex,      parameter
   WORD           wAddress,       parameter
   WORD           wLen            parameter
)
```

**dwTokenId** is the Token's identifier and can be obtained via *GNT_GetTokenID* function. It serves as a Token's handle.
**pBuffer** points to the buffer in PC's memory where data from a Token are copied to.
**bMEMIndex** is the index of MEM area. Valid values are 1,2,3.
**wAddress** is the address in MEM1/2/3 where data are read from.
**wLen** is the length of desired block of data to be read.

### Example

```
...
// reads 16 bytes from MEM3 from address 0x0100 into 'buff'
GNT_ReadMEM(dwTokenId,3,0x0100,16,buff);
...
```

### Return values

See the *Table of return values*.

### See also

*GNT_WriteMEM*.

## GNT_WriteMEM

writes a block of bytes to the Token's non-volatile memory.

### Syntax

```
DWORD GNT_WriteMEM (
   DWORD          dwTokenId,      token identifier
   PVOID          pBuffer,        input
   BYTE           bMEMIndex,      parameter
   WORD           wAddress,       parameter
   WORD           wLen            parameter
)
```

**dwTokenId** is the Token's identifier and can be obtained via *GNT_GetTokenID* function. It serves as a Token's handle.
**pBuffer** points to the buffer, where data to the Token are copied from.
**bMEMIndex** is the index of MEM area. Valid values are 1,2,3.
**wAddress** is the address in MEM1/2/3 where data are copied to.
**wLen** is the length of desired block of data to be written.

Example

```
...
// write 1 byte to MEM2 to the address 0x001F from 'buff'
GNT_WriteMEM(dwTokenId,2,0x001F,1,buff);
...
```

### Return values

See the *Table of return values*.

### See also

*GNT_ReadMEM*.

 

# Lifecycle functions

## GNT_Issue

issues a Token with specified data.

### Syntax

```
DWORD GNT_Issue (
   DWORD                dwTokenId,      token identifier
   PGNT_TOKEN_DATA      pIssueData      input
)
```

**dwTokenId** is the Token's identifier and can be obtained via *GNT_GetTokenID* function. It serves as a Token's handle.
**pIssueData** points to the structure that contains data related to the process of issuing. See the definition of *GNT_TOKEN_DATA*.

### Example

```
...
BYTE PIN1[0x10] = "pin1\x0\x0\x0\x0\x0\x0\x0\x0\x0\x0\x0\x0";
BYTE PIN2[0x10] = "pin2\x0\x0\x0\x0\x0\x0\x0\x0\x0\x0\x0\x0";
pIssueData->bMaxWrongLogins = 0;
pIssueData->bRSAkeys = 3;
strncpy(pIssueData->PIN1, PIN1, 16);
strncpy(pIssueData->PIN2, PIN2, 16);
pIssueData->wMEM1AccRights = GNT_MEM_ACCR_WRITE_PIN1 |
     GNT_MEM_ACCR_WRITE_PIN2 | GNT_MEM_ACCR_READ_FREE;
pIssueData->wMEM2AccRights = GNT_MEM_ACCR_WRITE_PIN2 |
     GNT_MEM_ACCR_READ_PIN2;
pIssueData->wMEM3AccRights = GNT_MEM_ACCR_WRITE_FREE |
     GNT_MEM_ACCR_READ_FREE;
pIssueData->wRSAslotAttributes[1] =
     GNT_RSA_ATTR_ALLOWED_SIGNATURE |
     GNT_RSA_ATTR_ALLOWED_WRAPPING |
     GNT_RSA_ATTR_ALLOWED_RSA_CRYPT |
     GNT_RSA_ATTR_PIN1_ACCESSIBLE |
     GNT_RSA_ATTR_PIN2_ACCESSIBLE |
     GNT_RSA_ATTR_EXPORTABLE_BY_GENER_ONLY |
     GNT_RSA_ATTR_GENERATABLE;
pIssueData->wRSAslotAttributes[2] =
     GNT_RSA_ATTR_ALLOWED_WRAPPING |
     GNT_RSA_ATTR_ALLOWED_RSA_CRYPT |
     GNT_RSA_ATTR_ERASABLE |
     GNT_RSA_ATTR_PIN2_ACCESSIBLE |
     GNT_RSA_ATTR_IMPORTABLE;
pIssueData->wRSAslotAttributes[3] =
     GNT_RSA_ATTR_ALLOWED_SIGNATURE |
     GNT_RSA_ATTR_ALLOWED_WRAPPING |
     GNT_RSA_ATTR_ERASABLE |
     GNT_RSA_ATTR_PIN2_ACCESSIBLE |
     GNT_RSA_ATTR_PIN1_ACCESSIBLE |
```

 

```
        GNT_RSA_ATTR_GENERATABLE |
        GNT_RSA_ATTR_EXPORTABLE;
... other pIssueData definitions ...
GNT_Issue(dwTokenId,pIssueData);
...
```

**Return values**

See the *Table of return values*.

**See also**

*GNT_Unissue*.

## GNT_Unissue

un-issues a Token, i.e. resets the Token to the uninitialized state.

**Syntax**

```
DWORD GNT_Unissue (
   DWORD                dwTokenId          token identifier
)
```

**dwTokenId** is the Token's identifier and can be obtained via *GNT_GetTokenID* function. It serves as a Token's handle.

**Example**

```
...
// the Token is in the 'issued' state
GNT_Unissue(dwTokenId);
// the Token is in the 'unissue' state
...
```

**Return values**

See the *Table of return values*.

**See also**

*GNT_Issue*.

 

# Data structures

## GNT_TOKEN_DATA

is a structure that contains information about internal Token's resources, such as RSA slots, MEM areas, etc.

**Definition**

```
typedef struct _GNT_TOKEN_DATA
{
   BYTE          PIN1[16];
   BYTE          PIN2[16];
   WORD          wMEM1End;
   WORD          wMEM2End;
   WORD          wMEM3End;
   WORD          wMEM1AccRights;
   WORD          wMEM2AccRights;
   WORD          wMEM3AccRights;
   BYTE          bRSAKeys;
   WORD          wRSAslotAttributes[GNT_MAX_RSA_SLOTS+1];
   DWORD         dwCurrWrongLogins;
   BYTE          bMaxWrongLogins;
} GNT_TOKEN_DATA, *PGNT_TOKEN_DATA
```

**PIN1** is the 16 bytes long 1st level user's access password. The password input is not limited to characters, the whole range of byte values 0-255 is supported.
**PIN2** is the 16 bytes long 2nd level user's access password. The password input is not limited to characters, the whole range of byte values 0-255 is supported.
**wMEM1End** is the address of the end of MEM1 area. The user memory area MEM1 is preceeded by RSA slots (value bRSAKeys), so this value must be taken into account while calculating the MEM1 length from wMEM1End (or vice versa in *GNT_Issue*). The following relation is valid:

*MEM1Len = wMEM1End - bRSAKeys*GNT_RSA_SLOT_LENGTH.*

**wMEM2End** is the address of the end of MEM2 area. The user memory area MEM2 is preceeded by MEM1. The following relation is valid:

*MEM2Len = wMEM2End - wMEM1End.*

**wMEM3End** is the address of the end of MEM3 area. The user memory area MEM3 is preceeded by MEM2, so the length of the MEM2. The following relation is valid:

*MEM3Len = wMEM3End - wMEM2End.*

**wMEM1AccRights** is the configuration word that defines access rights for MEM1 area. The following attributes can be used:

| Attribute | Description |
|---|---|
| GNT_MEM_ACCR_READ_FREE | enables free read access |
| GNT_MEM_ACCR_READ_PIN1 | enables read access with PIN1 |
| GNT_MEM_ACCR_READ_PIN2 | enables read access with PIN2 |
| GNT_MEM_ACCR_WRITE_FREE | enables free write access |

 

| Attribute | Description |
|---|---|
| GNT_MEM_ACCR_WRITE_PIN1 | enables write access with PIN1 |
| GNT_MEM_ACCR_WRITE_PIN2 | enables write access with PIN2 |

**wMEM2AccRights** is the configuration word that defines access rights for MEM2 area. See the description of the *wMEM1AccRights*.
**wMEM3AccRights** is the configuration word that defines access rights for MEM3 area. See the description of the *wMEM1AccRights*.
**bRSAkeys** is the number of configurable RSA slots. E.g., if this parameter is set to 2, a Token will contain 2 configurable slots and the RSA slot 0 that is always present.
**wRSAslotAttributes[ ]** is an array of configuration words that define attributes for RSA slots. While issuing a Token, a user cannot set *wRSAslotAccRights* for RSA slot 0 (i.e. *wRSAslotAccRights[0]*), as it has fixed attributes. However, after calling the GNT_GetTokenInfo() functions, *wRSAslotAccRights[0]* contains attributes of RSA slot 0. The following attributes can be used:

| Attribute | Description |
|---|---|
| GNT_RSA_ATTR_EXPORTABLE | complete RSA key can be exported from the RSA slot |
| GNT_RSA_ATTR_IMPORTABLE | complete RSA key can be imported into the RSA slot |
| GNT_RSA_ATTR_GENERATABLE | RSA key can be generated onboard in the RSA slot |
| GNT_RSA_ATTR_EXPORTABLE_BY_GENER_ONLY | complete RSA key can be exported only once - after successful onboard generation |
| GNT_RSA_ATTR_PIN1_ACCESSIBLE | RSA key is accessible if user is logged in with PIN1 |
| GNT_RSA_ATTR_PIN2_ACCESSIBLE | RSA key is accessible if user is logged in with PIN2 |
| GNT_RSA_ATTR_ALLOWED_RSA_CRYPT | RSA key can be used for RSA encryption/decryption |
| GNT_RSA_ATTR_ALLOWED_WRAPPING | RSA key can be used for key wrapping/unwrapping |
| GNT_RSA_ATTR_ALLOWED_SIGNATURE | RSA key can be used for generating/verifying digital signature |

**dwCurrWrongLogins** is the number of incorrect logins since last successful login. It's read only value.
**bMaxWrongLogins** is the maximum number of incorrect logins allowed. After the incorrect logins counter overflows this value, a Token is unissued and all internal data are erased. If this parameter is set to zero, there is no limit for incorrect logins.

## GNT_TOKEN_STATUS

is a structure that contains information about Token's login state and production state and

 

version number.

**Definition**

```
typedef struct _GNT_TOKEN_STATUS
{
   BYTE          LoVersion;
   BYTE          HiVersion;
   BYTE          LoginState;
   BYTE          ProdState;
} GNT_TOKEN_STATUS, *PGNT_TOKEN_STATUS
```

**LoVersion** is the minor part of firmware version number.
**HiVersion** is the major part of firmware version number.
**LoginState** is the login status information. It contains flags showing login state for PIN1, PIN2 and Admin. LoginState value can be the combination of any of these flags:

| Flag | Description |
|---|---|
| GNT_PIN1_LOGIN | user with PIN1 logged in |
| GNT_PIN2_LOGIN | user with PIN2 logged in |
| GNT_ADMIN_LOGIN | admin logged in |

**ProdState** is the information about Token's current production state. It can hold one of the following values:

| Value | Description |
|---|---|
| GNT_PS_ISSUED | Token is in the issued state. It can be used for cryptographic purposes. |
| GNT_PS_UNISSUED | Token is in the unissued state. It should be issued/configured by an Admin. |
| GNT_PS_UNPRODUCED | Token is in the unproduced state. It should be produced by the producer. |

## GNT_RSA_KEY

is a structure that contains RSA key in the form of modulus, public exponent and prime factors.

**Definition**

```
typedef struct _GNT_RSA_KEY
{
   unsigned char          modulus[GNT_RSA_MODULUS_LEN];
   unsigned char          publicExponent[GNT_RSA_MODULUS_LEN];
   unsigned char          prime[2][GNT_RSA_PRIME_LEN];
} GNT_RSA_KEY, *PGNT_RSA_KEY
```

**modulus** is the modulus of RSA key; $N=pq$.
**publicExponent** is the public exponent of RSA key, $e$.
**prime[ ]** is the array of two primes $p,q$, that are used by CRT private exponentiation;

 

private exponent *d* can be computed from them in the following way:
*d e ≡ 1 (mod (p-1)(q-1))*.

## GNT_CRYPT_KEY

is a structure that contains the specification of symmetric, public or private key required for crypto operations, where these keys are used, such as encryption/decryption, digital signature generation/verification, etc.

### Definition

```
typedef struct _GNT_CRYPT_KEY
{
   // symmetric key
   BYTE                    bKeySpecMethod;
   BYTE                    *pbKey;
   BYTE                    *pbIV;
   BYTE                    *pbCTR;
   BYTE                    *pbWrappedKeyIn;
   BYTE                    *pbWrappedKeyOut;
   GNT_POINTER_TO_MEM      PtrToSymKey;
   // public key
   BYTE                    bPubKeySpecMethod;
   GNT_POINTER_TO_MEM      PtrToPubKey;
   BYTE                    bRSAslotVerSign;
   BYTE                    bRSAslotEncrypt;
   BYTE                    bRSAslotWrap;
   // private key
   BYTE                    bRSAslotGenSign;
   BYTE                    bRSAslotDecrypt;
   BYTE                    bRSAslotUnwrap;
} GNT_CRYPT_KEY, *PGNT_CRYPT_KEY
```

**bKeySpecMethod** is the method for symmetric crypto key specification. Following methods are allowed:

| Value | Description |
|---|---|
| GNT_KEY_SPEC_GENERATE | onboard random symmetric key generation |
| GNT_KEY_SPEC_EXPLICIT | explicit definition of symmetric key defined by the user |
| GNT_KEY_SPEC_UNWRAP | symmetric key obtained by unwrapping wrapped key provided from outside of the token |
| GNT_KEY_SPEC_PTRMEM | symmetric key obtained from MEM area; the location of key in MEM is defined by specific pointer (see *GNT_POINTER_TO_MEM*) |
| GNT_KEY_SPEC_EXPLICIT_WRAP | explicit definition of symmetric key defined by the user; at the same time this key is wrapped and sent outside the token for possible later use |
| GNT_KEY_SPEC_PTRMEM_WRAP | symmetric key obtained from MEM area; the location of key in MEM is defined by specific |

 

| Value | Description |
|---|---|
| | pointer (see *GNT_POINTER_TO_MEM*); at the same time this key is wrapped and sent outside the token for possible later use |

**pbKey** points to the buffer in computer's memory, where the desired symmetric key is retrieved from. This parameter is needed when using GNT_KEY_SPEC_EXPLICIT key specification method.
**pbIV** points to the buffer in computer's memory, where the desired initialization vector is retrieved from. This parameter is needed when using GNT_KEY_SPEC_EXPLICIT key specification method and CBC or OFB mode is used.
**pbCTR** points to the buffer in computer's memory, where the desired counter is retrieved from. This parameter is needed when using GNT_KEY_SPEC_EXPLICIT key specification method and CTR mode is used.
**pbWrappedKeyIn** points to the buffer in computer's memory, where the desired wrapped symmetric key is retrieved from. This parameter is needed when using GNT_KEY_SPEC_UNWRAP key specification method.
**pbWrappedKeyOut** points to the buffer in computer's memory, where the wrapped symmetric key is stored into. This parameter is needed when using GNT_KEY_SPEC_GENERATE, GNT_KEY_SPEC_EXPLICIT_WRAP or GNT_KEY_SPEC_PTRMEM_WRAP key specification method.
**PtrToSymKey** is the structure that contains the information about symmetric key's position in MEM area. See the definition of *GNT_POINTER_TO_MEM*.
**bPubKeySpecMethod** is the method for public RSA key specification. Following methods are allowed:

| Value | Description |
|---|---|
| GNT_PUB_KEY_SPEC_PTRMEM | public RSA key is stored in MEM area |
| GNT_PUB_KEY_SPEC_RSASLOT | public RSA key is stored in RSA slot |

**PtrToPubKey** is the structure that contains the information about RSA public key's position in MEM area. See the definition of *GNT_POINTER_TO_MEM*. This parameter is needed when using GNT_PUB_KEY_SPEC_PTRMEM key specification method.
**bRSAslotVerSign** is the index of RSA slot where the public RSA key for signature verifying is stored.
**bRSAslotEncrypt** is the index of RSA slot where the public RSA key for encrypting of input block is stored.
**bRSAslotWrap** is the index of RSA slot where the public RSA key for symmetric key wrapping is stored.
**bRSAslotGenSign** is the index of RSA slot where the public RSA key for signature generating is stored.
**bRSAslotDecrypt** is the index of RSA slot where the public RSA key for decrypting of input block is stored.
**bRSAslotUnwrap** is the index of RSA slot where the public RSA key for symmetric key unwrapping is stored.

## GNT_CRYPT_OPTIONS

is a structure that contains information about used crypto algorithm, crypto mode or hash

 

algorithm.

**Definition**

```
typedef struct _GNT_CRYPT_OPTIONS
{
   WORD          wCrAlgorithm;
   WORD          wCrMode;
   WORD          wHashAlgorithm;
} GNT_CRYPT_OPTIONS, *PGNT_CRYPT_OPTIONS
```

**wCrAlgorithm** is the algorithm for symmetric cipher: GNT_CRYPT_ALG_AES, GNT_CRYPT_ALG_DES or, GNT_CRYPT_ALG_3DES.
**wCrMode** is the mode for symmetric cipher: GNT_CRYPT_MODE_ECB, GNT_CRYPT_MODE_CBC, GNT_CRYPT_MODE_CTR or GNT_CRYPT_MODE_OFB.
**wHashAlgorithm** is the hash algorithm: GNT_CRYPT_HASH_ALG_SHA1 or GNT_CRYPT_HASH_ALG_SHA256.

## GNT_SERIAL_NUM

is a structure that contains information about Token's serial number. The serial number is 16 bytes long, first 4 bytes (=DWORD) are used as a Token identifier *dwTokenId* and remaining 12 bytes are reserved for vendor specific informations.

**Definition**

```
typedef struct _GNT_SERIAL_NUM
{
   DWORD         dwTokenId;
   BYTE          bReserved[12];
} GNT_SERIAL_NUM, *PGNT_SERIAL_NUM
```

**dwTokenId** is the Token's identifier. It the DWORD value used to address specific Token. This value is equal to the one returned by the *GNT_GetTokenID* function.
**bReserved** field is reserved for vendor purposes.

## GNT_POINTER_TO_MEM

is a structure that contains a pointer to MEM area, specified by MEM index and MEM address.

**Definition**

```
typedef struct _GNT_POINTER_TO_MEM
{
   WORD          wMEMAddress;
   BYTE          bMEMIndex;
} GNT_POINTER_TO_MEM
```

**wMEMAddress** is the address inside MEM1/2/3; the address is relative to the beginning of MEMx (i.e. first byte of MEM3 has address *wMEMAddress = 0*)
**bMEMIndex** is the index of MEM area. It is equal to 1 for MEM1 area

 

# Appendix A

| Table of required parameters for GNT_CRYPT_KEY | | | |
|---|---|---|---|
| **Algorithm** | **Symmetric key specification (bKeySpecMethod)** | **Public key specification (bPubKeySpecMethod)** | **Required parameters** |
| **GNT_EncryptRSA** | - | GNT_PUB_KEY_SPEC_PTRMEM *(public key for RSA encryption is defined via pointer to MEM)* | bPubKeySpecMethod PtrToPubKey |
| | | GNT_PUB_KEY_SPEC_RSASLOT *(public key for RSA encryption is defined as an index of RSA slot)* | bPubKeySpecMethod bRSAslotEncrypt |
| **GNT_DecryptRSA** | - | - | bRSAslotDecrypt |
| **GNT_Sign** | - | - | bRSAslotGenSign |
| **GNT_Verify** | - | GNT_PUB_KEY_SPEC_PTRMEM *(public key for signature verification is defined via pointer to MEM)* | bPubKeySpecMethod PtrToPubKey |
| | | GNT_PUB_KEY_SPEC_RSASLOT *(public key for signature verification is defined as an index of RSA slot)* | bPubKeySpecMethod bRSAslotVerSign |
| **GNT_SignHash** | - | - | bRSAslotGenSign |
| **GNT_Encrypt** | GNT_KEY_SPEC_GENERATE *(symmetric key is generated in the token as a sequence of random numbers and exported outside token in wrapped form)* | GNT_PUB_KEY_SPEC_PTRMEM *(public key for wrapping a symmetric key is defined via pointer to MEM)* | bKeySpecMethod pbWrappedKeyOut bPubKeySpecMethod PtrToPubKey |
| | | GNT_PUB_KEY_SPEC_RSASLOT *(public key for wrapping a symmetric key is defined as an index of RSA slot)* | bKeySpecMethod pbWrappedKeyOut bPubKeySpecMethod bRSAslotWrap |
| | GNT_KEY_SPEC_EXPLICIT *(symmetric key is defined explicitly by the user as a sequence of bytes)* | - | bKeySpecMethod pbKey (pbIV, pbCTR) |
| | GNT_KEY_SPEC_PTRMEM *(symmetric key is defined as a sequence of bytes in MEM that are pointed to by a pointer to MEM)* | - | bKeySpecMethod PtrToSymKey |
| | GNT_KEY_SPEC_EXPLICIT_WRAP *(symmetric key is defined explicitly by the user as a sequence of bytes and exported outside token in wrapped form)* | GNT_PUB_KEY_SPEC_PTRMEM *(public key for wrapping a symmetric key is defined via pointer to MEM)* | bKeySpecMethod pbKey (pbIV, pbCTR) pbWrappedKeyOut bPubKeySpecMethod PtrToPubKey |
| | | GNT_PUB_KEY_SPEC_RSASLOT *(public key for wrapping a symmetric key is defined as an index of RSA slot)* | bKeySpecMethod pbWrappedKeyOut bPubKeySpecMethod bRSAslotWrap |
| | GNT_KEY_SPEC_PTRMEM_WRAP *(symmetric key is defined as a sequence of bytes in MEM that are pointed to by a pointer to MEM and exported outside token in wrapped form)* | GNT_PUB_KEY_SPEC_PTRMEM *(public key for wrapping a symmetric key is defined via pointer to MEM)* | bKeySpecMethod PtrToSymKey pbWrappedKeyOut bPubKeySpecMethod PtrToPubKey |
| | | GNT_PUB_KEY_SPEC_RSASLOT *(public key for wrapping a symmetric key is defined as an index of RSA slot)* | bKeySpecMethod PtrToSymKey pbWrappedKeyOut bPubKeySpecMethod bRSAslotWrap |

| Algorithm | Symmetric key specification (bKeySpecMethod) | Public key specification (bPubKeySpecMethod) | Required parameters |
|---|---|---|---|
| | GNT_KEY_SPEC_UNWRAP<br>*(symmetric key is imported into token in wrapped form and unwrapped by the token via RSA algorithm)* | - | bKeySpecMethod<br>pbWrappedKeyIn<br>bRSAslotUnwrap |
| **GNT_Decrypt** | GNT_KEY_SPEC_EXPLICIT<br>*(symmetric key is defined explicitly by the user as a sequence of bytes)* | - | bKeySpecMethod<br>pbKey (pbIV, pbCTR) |
| | GNT_KEY_SPEC_UNWRAP<br>*(symmetric key is imported into token in wrapped form and unwrapped by the token via RSA algorithm)* | - | bKeySpecMethod<br>pbWrappedKeyIn<br>bRSAslotUnwrap |
| | GNT_KEY_SPEC_PTRMEM<br>*(symmetric key is defined as a sequence of bytes in MEM that are pointed to by a pointer to MEM)* | - | bKeySpecMethod<br>PtrToSymKey |
| **GNT_EncryptSign** | GNT_KEY_SPEC_GENERATE<br>*(symmetric key is generated in the token as a sequence of random numbers and exported outside token in wrapped form)* | GNT_PUB_KEY_SPEC_PTRMEM<br>*(public key for wrapping a symmetric key is defined via pointer to MEM)* | bKeySpecMethod<br>pbWrappedKeyOut<br>bPubKeySpecMethod<br>PtrToPubKey<br>bRSAslotGenSign |
| | | GNT_PUB_KEY_SPEC_RSASLOT<br>*(public key for wrapping a symmetric key is defined as an index of RSA slot)* | bKeySpecMethod<br>pbWrappedKeyOut<br>bPubKeySpecMethod<br>bRSAslotWrap<br>bRSAslotGenSign |
| | GNT_KEY_SPEC_EXPLICIT<br>*(symmetric key is defined explicitly by the user as a sequence of bytes)* | - | bKeySpecMethod<br>pbKey (pbIV, pbCTR)<br>bRSAslotGenSign |
| | GNT_KEY_SPEC_PTRMEM<br>*(symmetric key is defined as a sequence of bytes in MEM that are pointed to by a pointer to MEM)* | - | bKeySpecMethod<br>PtrToSymKey<br>bRSAslotGenSign |
| | GNT_KEY_SPEC_EXPLICIT_WRAP<br>*(symmetric key is defined explicitly by the user as a sequence of bytes and exported outside token in wrapped form)* | GNT_PUB_KEY_SPEC_PTRMEM<br>*(public key for wrapping a symmetric key is defined via pointer to MEM)* | bKeySpecMethod<br>pbKey (pbIV, pbCTR)<br>pbWrappedKeyOut<br>bPubKeySpecMethod<br>PtrToPubKey<br>bRSAslotGenSign |
| | | GNT_PUB_KEY_SPEC_RSASLOT<br>*(public key for wrapping a symmetric key is defined as an index of RSA slot)* | bKeySpecMethod<br>pbKey (pbIV, pbCTR)<br>pbWrappedKeyOut<br>bPubKeySpecMethod<br>bRSAslotWrap<br>bRSAslotGenSign |
| | GNT_KEY_SPEC_PTRMEM_WRAP<br>*(symmetric key is defined as a sequence of bytes in MEM that are pointed to by a pointer to MEM and exported outside token in wrapped form)* | GNT_PUB_KEY_SPEC_PTRMEM<br>*(public key for wrapping a symmetric key is defined via pointer to MEM)* | bKeySpecMethod<br>PtrToSymKey<br>pbWrappedKeyOut<br>bPubKeySpecMethod<br>PtrToPubKey<br>bRSAslotGenSign |
| | | GNT_PUB_KEY_SPEC_RSASLOT<br>*(public key for wrapping a symmetric key is defined as an index of RSA slot)* | bKeySpecMethod<br>PtrToSymKey<br>pbWrappedKeyOut<br>bPubKeySpecMethod<br>bRSAslotWrap<br>bRSAslotGenSign |
| | GNT_KEY_SPEC_UNWRAP | - | |

| Algorithm | Symmetric key specification (bKeySpecMethod) | Public key specification (bPubKeySpecMethod) | Required parameters |
|---|---|---|---|
| | *(symmetric key is imported into token in wrapped form and unwrapped by the token via RSA algorithm)* | | bKeySpecMethod<br>pbWrappedKeyIn<br>bRSAslotUnwrap |
| **GNT_DecryptVerify** | GNT_KEY_SPEC_EXPLICIT<br>*(symmetric key is defined explicitly by the user as a sequence of bytes)* | GNT_PUB_KEY_SPEC_PTRMEM<br>*(public key for signature verification is defined as a pointer to MEM)* | bKeySpecMethod<br>pbKey (pbIV, pbCTR)<br>bPubKeySpecMethod<br>PtrToPubKey |
| | | GNT_PUB_KEY_SPEC_RSASLOT<br>*(public key for signature verification is defined as an index of RSA slot)* | bKeySpecMethod<br>pbKey (pbIV, pbCTR)<br>bPubKeySpecMethod<br>bRSAslotVerSign |
| | GNT_KEY_SPEC_UNWRAP<br>*(symmetric key is imported into token in wrapped form and unwrapped by the token via RSA algorithm)* | GNT_PUB_KEY_SPEC_PTRMEM<br>*(public key for signature verification is defined as a pointer to MEM)* | bKeySpecMethod<br>pbWrappedKeyIn<br>bRSAslotUnwrap<br>bPubKeySpecMethod<br>PtrToPubKey |
| | | GNT_PUB_KEY_SPEC_RSASLOT<br>*(public key for signature verification is defined as an index of RSA slot)* | bKeySpecMethod<br>pbWrappedKeyIn<br>bRSAslotUnwrap<br>bPubKeySpecMethod<br>bRSAslotVerSign |
| | GNT_KEY_SPEC_PTRMEM<br>*(symmetric key is defined as a sequence of bytes in MEM that are pointed to by a pointer to MEM)* | GNT_PUB_KEY_SPEC_PTRMEM<br>*(public key for signature verification is defined as a pointer to MEM)* | bKeySpecMethod<br>PtrToSymKey<br>bPubKeySpecMethod<br>PtrToPubKey |
| | | GNT_PUB_KEY_SPEC_RSASLOT<br>*(public key for signature verification is defined as an index of RSA slot)* | bKeySpecMethod<br>PtrToSymKey<br>bPubKeySpecMethod<br>bRSAslotVerSign |

*Note:* pbIV and pbCTR are conditional parameters; they depend on chosen symmetric encryption mode (ECB, CBC, OFB, CTR). If ECB mode is set, only pbKey is required, if CBC mode is set, pbKey and pbIV are required, if OFB mode is set, pbKey and pbIV are required and finally if CTR mode is set, pbKey and pbCTR are required.

**Table of required parameters for GNT_CRYPT_OPTONS**

| Algorithm | Required parameter | Possible values |
|---|---|---|
| **GNT_Hash** | wHashAlgorithm | GNT_CRYPT_HASH_ALG_SHA1, GNT_CRYPT_HASH_ALG_SHA256 |
| **GNT_EncryptRSA** | - | - |
| **GNT_DecryptRSA** | - | - |
| **GNT_Sign** | wHashAlgorithm | GNT_CRYPT_HASH_ALG_SHA1, GNT_CRYPT_HASH_ALG_SHA256 |
| **GNT_Verify** | wHashAlgorithm | GNT_CRYPT_HASH_ALG_SHA1, GNT_CRYPT_HASH_ALG_SHA256 |
| **GNT_Encrypt** | wCrAlgorithm | GNT_CRYPT_ALG_AES, GNT_CRYPT_ALG_DES, GNT_CRYPT_ALG_3DES |
| | wCrMode | GNT_CRYPT_MODE_ECB GNT_CRYPT_MODE_CBC, GNT_CRYPT_MODE_OFB, GNT_CRYPT_MODE_CTR |
| **GNT_Decrypt** | wCrAlgorithm | GNT_CRYPT_ALG_AES, GNT_CRYPT_ALG_DES, GNT_CRYPT_ALG_3DES |
| | wCrMode | GNT_CRYPT_MODE_ECB GNT_CRYPT_MODE_CBC, GNT_CRYPT_MODE_OFB, GNT_CRYPT_MODE_CTR |
| **GNT_EncryptSign** | wCrAlgorithm | GNT_CRYPT_ALG_AES, GNT_CRYPT_ALG_DES, GNT_CRYPT_ALG_3DES |
| | wCrMode | GNT_CRYPT_MODE_ECB GNT_CRYPT_MODE_CBC, GNT_CRYPT_MODE_OFB, GNT_CRYPT_MODE_CTR |
| | wHashAlgorithm | GNT_CRYPT_HASH_ALG_SHA1 |
| **GNT_DecryptVerify** | wCrAlgorithm | GNT_CRYPT_ALG_AES, GNT_CRYPT_ALG_DES, GNT_CRYPT_ALG_3DES |
| | wCrMode | GNT_CRYPT_MODE_ECB GNT_CRYPT_MODE_CBC, GNT_CRYPT_MODE_OFB, GNT_CRYPT_MODE_CTR |
| | wHashAlgorithm | GNT_CRYPT_HASH_ALG_SHA1 |
| **GNT_SignHash** | wHashAlgorithm | GNT_CRYPT_HASH_ALG_SHA1, GNT_CRYPT_HASH_ALG_SHA256 |

## Table of return values

| Return status | Value | Description |
|---|---|---|
| GNT_STATUS_SUCCESS | 0x00 | Operation was successfully completed. |
| GNT_STATUS_WRONG_PARAM | 0x01 | The some of the input parameters of the function is wrong or out of range. |
| GNT_STATUS_NOT_LOGGED | 0x02 | The login procedure failed (incorrect access password) or user is does not satisfy the access level (PIN1 or PIN2) required to perform specified operation. |
| GNT_STATUS_WRONG_DATA | 0x03 | Wrong data received, probably due to a USB error. |
| GNT_STATUS_INVALID_SIGNATURE | 0x04 | Digital signature verification failed, signature is untrusted. |
| GNT_STATUS_INSUFF_PRIVILEGES | 0x05 | Insufficient privileges to perform requested operation, login (PIN1 or PIN2) is needed. |
| GNT_STATUS_NO_MORE_DATA | 0x07 | No more decrypted data available. Applies only to the low level API functions. |
| GNT_STATUS_ALREADY_PRODUCED | 0x08 | User tried to produce a token that is already produced. Once a token is produced, it cannot be reproduced. |
| GNT_STATUS_UNISSUED | 0x09 | User tries to use a token that hasn't been issued yet. Prior to using a token for cryptographic purposes one has to issue it first (via *GNT_Issue()* function). |
| GNT_STATUS_ALREADY_ISSUED | 0x0A | User tried to issue a token that is already issued. If there's need to reissue a token, user has to call *GNT_Unissue()* first. |
| GNT_STATUS_UNSUPPORTED_OPERATION | 0x0B | Requested operation is not permitted for specified RSA slot. |
| GNT_STATUS_NOT_PRODUCED | 0x0C | User tries to access a token that hasn't been produced yet. Prior to issuing or using a token it has to be produced first by the production. |
| GNT_STATUS_RSA_SLOT_EMPTY | 0x0D | The specified RSA slot doesn't contain an RSA key so it cannot be used for cryptographic operations. |
| GNT_STATUS_NOT_FOUND | 0x16 | Specified token was not found, it is probably unplugged. |
| GNT_STATUS_NOT_ENOUGH_MEMORY | 0x17 | Not enough PC's memory for requested operation. |
| GNT_STATUS_CANCELLED | 0x20 | The operation has been cancelled. |
| GNT_STATUS_UNKNOWN_ERROR | 0xFF | Unspecified error. |

SoftIdea s.r.o.
Sliačska 10, 831 02 Bratislava
tel.: +421 2 444 60 444
fax.: +421 2 446 40 441
http://www.softidea.sk
info@softidea.sk