Guardant®

Anti-piracy Protection System

User's Manual

Supplement

Guardant Stealth II
Guardant Net II

Dongles

Revision 4.8

© 2005 Aktiv Company

Contents

Guardant Stealth II and Guardant Net II – New Dongles of Guardant Family

The new dongles employ a most up-to-date elemental base that made it possible to significantly expand their protection potential. Guardant Stealth II and Net II are fully compatible with the preceding products of the Guardant family, retaining the functionality of Guardant Stealth and Net dongles.

The difference between new dongles of Guardant family and the preceding Stealth and Net dongles lies primarily in the implementation of a new GSII64 data encoding hardware algorithm.

To protect the device against hardware cracking, the entire content of Guardant Stealth II EEPROM-memory is securely enciphered with a key that is unique for each dongle.

Guardant Stealth and Net II USB are introduced in a new embodiment, that is a single-unit cast high-strength plastic case and a custom designed long-life USB connector.

Important

This document is a supplement to the Guardant User's Manual and contains information specific for Guardant Stealth II and Guardant Net II dongles.

GSII64 Algorithm

Algorithm Description

GSII64 is the new algorithm developed for Guardant dongles. GSII64 is a block algorithm resistant to cryptanalysis and specially devised for Guardant Stealth II. Key length is 16 or 32 bytes (128 or 256 bits). GSII64 encodes data in 8-byte (64-bit) blocks. It also supports encoding of data sequences of length divisible by 8 and random length. The algorithm is symmetric, hence it can be used for both data encoding and decoding with the same key.

GSII64 algorithm supports the following functions:

- block data encoding (64-bit blocks)
- stream data encoding (data of random length)
- pseudorandom numbers generation
- hash calculation

GSII64 Algorithm Descriptor

An individual descriptor specifies the properties and representation of each hardware algorithm. Each descriptor occupies a certain space in the memory. Therefore, the number of algorithms in the dongle is limited.

The descriptor of GSII64 hardware algorithm comprises the following components:

Offset from the Beginning of the Descriptor	**Field Length in Bytes**	**Field Name**	**Field Description**
0	1	km_ad_flags	Algorithm flags
1	1	km_ad_algo	Algorithm type code (must be 5 for GSII64)
2	4	km_ad_GP	Algorithm counter
6	1	km_ad_klen	Determinant (key) size in bytes (16 or 32 bytes)
7	1	km_ad_blen	Query length in bytes (must be 8 bytes)
8	equals km_ad_klen	km_ao	Determinant

km_ad_flags field. This contains flags*, which specify the properties of the hardware algorithm. The following flags can be set (names of flags provided below are used in Guardant API):

- nsaf_ID: the algorithm depends on the dongle's ID
- nsaf_GP: the algorithm depends on its km_ad_GP field
- nsaf_GP_dec: km_ad_GP field must be decremented each time before the algorithm is executed.

km_ad_algo field. This contains algorithm type code. The value must equal 5 for GSII64 algorithm.

km_ad_GP field. This contains the algorithm's counter. If nsaf_GP_dec is indicated, the field specifies the number of times the algorithms can be executed. When the counter reaches 0 value the algorithm ceases to convert data. If an area of the algorithm has a hardware-based read and write lock, this field can be incremented only if the entire algorithm is written anew.

km_ad_klen field. This contains the size of the algorithm determinant in bytes. For GSII64 type algorithm the value may equal only 16 or 32 bytes. This value must be correct. If a wrong value has been entered, the determinant length is set at 16 bytes by default.

km_ad_blen field. This contains the size of the algorithm query in bytes. For GSII64 type algorithm the value must equal 8 bytes.

km_ao field. This contains the algorithm's determinant which is the most important part of the descriptor. The determinant plays a major part in specifying a particular method for data conversion (i.e., in specifying the exact type of hardware algorithm). The value of the determinant must correspond to the value of **km_ad_klen** field and equal 16 or 32 bytes.

A hardware-based read and write lock can be implemented for the memory area occupied by descriptors; this makes study, replication or modification of hardware algorithms impossible.

GSII64 Algorithm Modes

ECB Mode

Electronic Code Book mode. This is the simplest of GSII64 algorithm modes. Under ECB, each 8-byte block supplied to the algorithm encodes into 8-byte block of encoded data with the same determinant. Thus a data block will transform to an identical encoded data block.

GSII64 is designed to process 8-byte blocks. If the block length exceeds 8 bytes, then the data must be sent to the algorithm by 8-byte blocks. If the final block is less than 8 bytes, it must be padded to 8 bytes. It is highly recommended to keep these additional bytes random. Random numbers can be used for the additional bytes. In the latter case the last encoded 8-byte block must be stored completely, together with the encoded additional bytes (rather than discarding these bytes). This will ensure correct decoding of the useful data in this block.

The ECB mode is intended to encode small volumes of data, like initialization vectors or encryption keys for other algorithm modes and algorithms.

The ECB mode is recommended for use instead of the previous Stealth algorithm, due to GSII64's higher resistibility.

CBC Mode

Code Block Chaining mode. Under CBC mode, as in the case of ECB, every 8-byte block transforms to an 8-byte block. Algorithm encodes 8-byte blocks with the same determinant. The CBC mode is better suited for conversion of data blocks that exceed 8 bytes.

Unlike ECB, however, conversion of two identical 8-byte blocks located in different positions of the original data array will not yield an identical result. This is accomplished due to the fact that, rather than encoding the block itself, the sum of the block by module 2 is encoded at every successive step. To produce the first encoded block the sum by module 2 of the first encoded block and some initialization vector IV is used. Value IV should be retained for correct reverse transformation (decoding), yet it is advisable to protect it (e.g. to encode it in ECB mode).

Thus conversion will be position-dependent, since encoding result depends not only on the block itself, but also on the preceding block.

Reverse transformation will also be done on a block-by-block basis.

The total length of the original data block must be divisible by 8 bytes. Otherwise, the last block has to be padded to 8 bytes, as in the case of ECB mode.

The CBC mode can be employed for calculation of reliable checksums, and data authentication and verification. The last encoded 8-byte block is used as checksum. This block depends both on all preceding encoded blocks and on initialization vector, and is calculated on the basis of the algorithm determinant. The block does not provide information on original data, but identifies those unambiguously. It is as difficult to replicate this block as to fit the algorithm determinant.

If the length of the encoded data array is not divisible by 8, then the last block must be padded to 8 bytes. It is highly recommended to keep these additional bytes random. Random numbers can be used for the additional bytes. In the latter case the last encoded 8-byte block must be stored completely, together with the encoded additional bytes (rather than giving up these bytes). This will ensure correct decoding of the useful data in this block.

CFB Mode

Coded FeedBack mode. The CFB mode allows transforming data blocks of arbitrary length, not necessarily divisible by 8 bytes. This spares the effort to pad the original data to an integer number of 8-byte blocks. Under this mode the length of the encoded and the original sequences will be equal

Under the CFB mode, as in the case of CBC, original data blocks are chained; hence every encoded block will depend on all preceding blocks of original data, since an encoded preceding block is used for encoding of every following block of original data.

Under this mode initialization vector IV is used for data transformation (see CBC mode).

Important

If, at decoding, a wrong initialization vector is indicated, all data, except for the first 8 bytes, will be decoded correctly. Should this be critical for the application, the OFB mode is to be preferred.

OFB Mode

Output FeedBack mode. This has much in common with the CFB mode.

The main difference is that in OFB, to encode the following block, the result of initialization vector IV's transformation is used instead of the encoded preceding block. The advantage of this mode is that at transmission of the encoded data the dependence on distortions in the preceding blocks is diminished. Yet, the mode has its negative side: the OFB provides lower protection against malicious alteration of data, since alteration in one bit of encoded data will result in alteration of the same bit in decoded data. Here, a reliable checksum need to be used to authenticate data.

As in the case of two previous modes, the OFB uses initialization vector IV to transform data (See CBC and CFB modes).

Recommendations for Working with Initialization Vector IV

To correctly transform data using GSII64 algorithm it is required to take into account the following:

- initialization vector IV must be equally initialized before encoding and decoding;
- to preserve the value of initialization vector IV in the intervals between addressing TransformEx at continuous encoding/decoding of large blocks (over 248 bytes for ECB and CBC and 255 bytes for CFB and OFB);
- at some encoding operations, like encoding of various database records or disk sectors, to initialize IV with that number of the record/sector. This is done to ensure that each of those records/sectors is at all times encoded identically, while different records with same values are encoded differently.

Modes, Administered by Algorithm Property Flags

Field value km_ad_flags	Mode Meaning	Mode Description
0	Default mode	Flags are not set; algorithm does not depend on flags.
nsaf_GP_dec	Limitation on the number of algorithm executions	A 4-byte initial value of the counter is entered into the km_ad_GP field. The counter is decremented at every address to TransformEx, and the algorithm stops at counter reaching 0.
nsaf_GP	Dependence of algorithm on counter	A 4-byte value of the counter is entered into the km_ad_GP field. The type of transformation depends on counter value. If determinants are identical, algorithms with different counter values will encode data differently.
nsaf_GP +nsaf_GP_dec	Pseudorandom number generator	A high 4-byte initial value of the counter is entered into the km_ad_GP field. The counter is decremented at every address to TransformEx, and the algorithm stops at counter reaching 0. With every counter decrement the transformation is done differently.
nsaf_ID	Algorithm unicity by ID	Encoding depends on the dongle's ID. If determinants are identical, algorithms with different IDs will encode data differently.
nsaf_ID + nsaf_GP_dec	Limitation on the number of algorithm executions + unicity by ID	A 4-byte initial value of the counter is entered into the km_ad_GP field. The counter is decremented at every address to TransformEx, and the algorithm stops at counter reaching 0.
nsaf_ID + nsaf_GP +nsaf_GP_dec	Pseudorandom number generator	A high 4-byte initial value of the counter is entered into the km_ad_GP field. The counter is decremented at every address to TransformEx, and the algorithm stops at counter reaching 0. With every counter decrement the transformation is done differently.

Guardant API Functions

New Constants

New constants for C/C++ are described in the last version of NVSKEY32.H file. Constants for other languages are described in corresponding header files or within source code samples.

The Codes of Dongle Models

To assure identification of Guardant Stealth II dongles new codes of dongle models have been added. The model code is stored in the memory at sam_bKeyModel address (see the dongle memory map). Following codes are assigned to dongle models (see NVSKEY32. H):

Constant Name	Value	Dongle Model
nskm_GS1L	0	Guardant Stealth LPT
nskm_GS1U	1	Guardant Stealth USB
nskm_GF1L	2	Guardant Fidus LPT
nskm_GS2L	3	Guardant StealthII LPT
nskm_GS2U	4	Guardant StealthII USB

Previous constants have been kept for compatibility purposes.

Different Dongle Type Codes

To specify a dongle type, a new nskt_GSII64 flag has been added on the sam_bwType address implying that the dongle supports the GSII64 algorithm. This flag may be used for dongle search.

Thus following flags of dongle types are available (See NVSKEY32.H):

Constant Name	Value	Dongle Type
nskt_DOS	0	The dongle supports protection of DOS applications
nskt_Win	0	The dongle supports protection of Windows applications
nskt_LAN	1	The dongle supports protection of LAN applications. (Net or Net II dongle)
nskt_Time	2	The dongle is capable of limiting the protected application’s license term
nskt_GSII64	8	The dongle supports GSII64 algorithm

New Error Code

In addition to the above-mentioned changes, a new error code was added:

Constant Name	Value	Short Description
nse_InvalidArg	46	Inadmissible value of one of function's arguments is set

TransformEx Operation. Transforming Information Using the Dongle's GSII64 Hardware Algorithm

For Guardant Stealth II:

nRet = nskTransformEx(dwPrivateRD, dwAlgoNum, dwLng, pData, dwMethod, pIV)

For Guardant Net II:

nRet = nnkTransformEx(dwPrivateRD, dwAlgoNum, dwLng, pData, dwMethod, pIV)

Operation type:

Main built-in

Input parameters:

dwPrivateRD	Private Read code in numerical form.
dwAlgoNum	Hardware algorithm number
dwLng	Length in bytes of the data block to be converted.
pData	Address of the data block to be converted.
dwMethod	Conversion mode
pIV	The address of 8-byte IV initialization vector

Output parameters:

Error code

Description:

The nXkTransformEx() functions allow to transform information with the GSII64 hardware algorithm. These functions are available only for Win32 applications.

Conversion is carried out by the algorithm whose number is specified by bAlgoNum parameter. This algorithm must be created in advance. The length in bytes of data array to be converted is specified by bLng parameter and depends on the conversion mode specified by dwMethod. For the ECB and CBC modes the data length should be divisible by nsars_GSII64 (8 bytes), 248 bytes maximum. The function returns the nse_InvalidArg error code if data length is not divisible by 8 bytes. For the CFB and OFB modes any length not exceeding 255 bytes may be set. The dwMethod parameter is a bitmapped value(See NVSKEY32.H):

Flag	**Value**	**Description**
0-5 bits – algorithm's work mode		
nsam_ECB	0	Electronic code book mode
nsam_CBC	1	Code block chaining mode
nsam_CFB	2	Coded feedback mode
nsam_OFB	3	Output feedback mode
Bit 6 - reserved		
Bit 7 – operation type		
nsam_Encode	0	Encode block
nsam_Decode	128	Decode block

The data block to be transformed should be placed at address specified by pData parameter. If the function is executed successfully, then the same length sequence of the transformed data will be placed at that address. In this case the function returns nse_Ok.

The encoding / decoding speed directly depends on the dwLng length of the pData data block. Thus the speed is maximal at the maximal block length. If the data block size significantly exceeds the maximal dwLng value, then it needs to be broken into pieces of the maximum allowed length. However under this approach the dongle (especially an LPT dongle) may be busy for a longer time at every operation of this kind. Therefore in applications with critical timing parameters (for example, in applications with multilple independent parallel inquiries to the dongle), it is better to use smaller blocks.

For the modes using chaining of blocks, it is necessary to set an 8-byte pIV initialization vector. The same initialization vector value must be specified for both encoding and decoding. If encoding / decoding is performed in multiple steps, then a value set as an initialization vector is returned to pIV after the step has been completed to serve as an initialization vector for the subsequent step.

If the nsaf_GP_dec flag is set in a descriptor of algorithm, then the decrementation of GP counter occurs at each TransformEx call.

Example:

To encode and decode a test string by available GSII64 algorithm in OFB mode.

```
/* A string to be encoded */
char sData[]            = "Test 32 bytes for Encode/Decode.\0";
/* Initialization Vector */
char sInitVector[nsars_GSII64];
/* Set the initial value of the initialization vector*/
strcpy( sInitVector, "__IV___" ); /* Init Vector for GSII64 algo    */
/* Encode data */
nRet = nskTransformEx (
          // Regular Transform parameters
          dwPrivateRD, nsan_GSII64, DataLen, sData,
          // New TransformEx parameters
          nsam_OFB + nsam_Encode,     /* Encode data in OFB mode */
          sInitVector );
/* Check error code (nRet) and decide on further running of the application */
.........
/* Restore initialization vector*/
strcpy( sInitVector, "__IV___" );
/* Decode data */
      nRet = nskTransformEx (
          // Regular Transform parameters
          dwPrivRD + Crypt, nsan_GSII64, DataLen, sData,
          // New TransformEx parameters
          nsam_OFB + nsam_Decode,      /* Decode data in OFB mode */
          sInitVector );
/* Check error code (nRet) and decide on further running of the application */
.........
```

The nskCommand() Function

nsc_Transform

This document describes only changes in Guardant Stealth II (see nskCommand() function description in User's Manual).

The nsc_Transform command parameters of nskCommand() function (these are stored in the ns_Args structure type).

The na_CRC variable sets an algorithm mode (see the ns_Args structure), defined as a bitmapped value (see NVSKEY32.H):

Flag	Value	Description
Bits 0-5 – algorithm work mode		
nsam_ECB	0	Electronic code book mode
nsam_CBC	1	Code block chaining mode
nsam_CFB	2	Coded feedback mode
nsam_OFB	3	Output feedback mode
Bit 6 – reserved		
Bit 7 - operation type		
nsam_Ecode	0	Encode block
nsam_Decode	128	Decode block

The na_bLen variable: sets the size of data block to be transformed. For the ECB and CBC modes this value must be divisible by 8 and not exceed 248 bytes. For other modes any value may be set within the limits of 1 – 255 bytes. If the value is specified incorrectly in na_bLen, nse_InvalidArg error code is returned.

Starting with the na_dwLen32 variable's address (bytes 56-64 in the ns_Args structure), 8 bytes are occupied by the initialization vector IV. After the nskCommand function has been called, the new value of initialization vector is written in it, which may be used for subsequent nsc_Transform command callings for GSII64 algorithm. Sequential Transform call can be applied for stream encoding of data with size exceeding 255 bytes. Here, however, for all calls but the last one, the data length should necessarily be divisible by 8, i.e., including CFB and OFB methods.

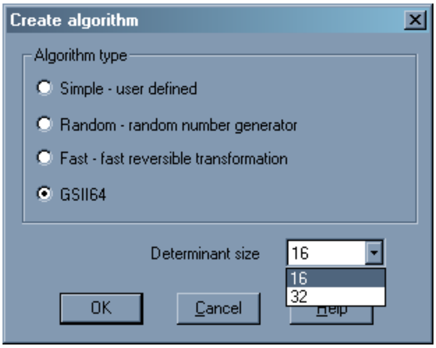
Data decoding may be performed by blocks of optional size (not necessarily by those used for the same data encoding), but with size divisible by 8 (except for the last portion of data). Other parameters remain unchanged.

Working with GSII64 Algorithm from the NSKUTIL Dongle Programming Utility

New GSII64 Algorithm Creation

Toolbar	
Menu	**Edit | Add**
Hotkey	**<Ins>**

To create a new hardware algorithm it is required to write a descriptor into the dongle. To create a new algorithm you should add a new field of 'Algorithm' type. It is possible to add this field by clicking any of above-mentioned controls. A window then appears where the 'GSII64' algorithm type needs to be selected:


Create algorithm

Algorithm type

( ) Simple - user defined

( ) Random - random number generator

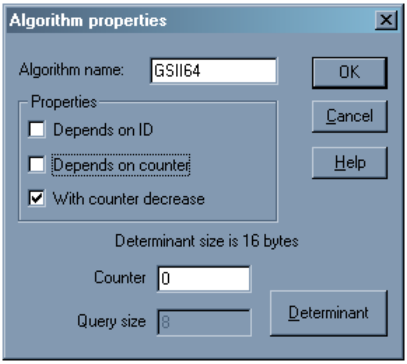
( ) Fast - fast reversible transformation

(●) GSII64

Determinant size: 16 (list: 16, 32)

The GSII64 type algorithm determinant size may be 16 or 32 bytes; the value is to be selected from the list. The values in the list are displayed in a number system chosen in the utility main window.

After choosing the type of algorithm, the following window will appear:



Algorithm name: GSII64

Properties

- [ ] Depends on ID
- [ ] Depends on counter
- [✓] With counter decrease

Determinant size is 16 bytes

Counter 0

Query size 8

In this window the algorithm name and properties should be set. The query size for this algorithm is a fixed value. To set the value of a determinant, click **[Determinant]** button and enter or load the data from a file. By default random numbers are entered in the determinant.

To assure writing and reading protection for new algorithm descriptor the NSKUTIL will automatically correct the length of protected memory area at the stage of new algorithm creation. The new hardware algorithm will become available only after writing data into the dongle's memory.

Warning

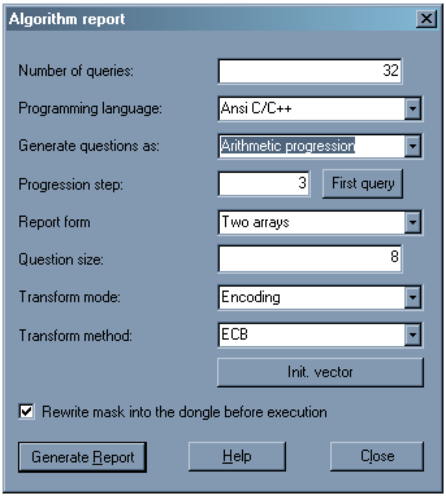
Since an algorithm is started by its sequential number, it is not recommended to insert new algorithms between the existing ones; this will cause changes in numbers of algorithms which follow the new algorithm in memory.

Obtaining Responses from GSII64 Hardware Algorithm

Toolbar	
Menu	**Dongle/Generate algorithm report**
Hotkey	**<Ctrl-Q>**

To use hardware algorithms of the dongle, it is necessary to know the sequence that the required algorithm will return in reply to a query. This response can then be used to make the protection logic more sophisticated.

To obtain responses from a particular algorithm you should select it from the list and click on any of the above controls. The following window will appear:



Number of queries:	32
Programming language:	Ansi C/C++
Generate questions as:	Arithmetic progression
Progression step:	3
Report form	Two arrays
Question size:	8
Transform mode:	Encoding
Transform method:	ECB

☑ Rewrite mask into the dongle before execution

In the ‘Number of queries’ field you should specify the number of instances of the TransformEx operation required to obtain response. At each TransformEx call a reply sequence is generated (reply), with length coinciding with the question’s length, which is set in a ‘Question size’ field. For example, if in the ’Number of queries’ you specified number 4, and in the ‘Question size’ you specified 32, then 4 32 byte-long answers will be returned. For the ECB and CBC modes the question length should be divisible by 8 bytes and not exceed 248 bytes. For the ECB and CBC modes any question length not exceeding 255 bytes may be set.

From the ’Generate questions as’ list the sequence generation method of an algorithm question is selected. There are 2 ways for doing that: the question may be generated as a random numbers sequence or as an arithmetical progression. The second option requires selection of the progression step (in ’Progression step‘ field) and its first element should be set in a hexadecimal editor (the **[First query]** button).

For convenience the algorithm’s questions and corresponding answers are saved in a log file. This is a text file following syntax of one of the programming languages: C/C++, Pascal/Delphi or assembler. A language may be selected from the ‘Programming language’ list.

Questions and answers are written into a log file in the form of one or two arrays. The ‘Report form’ list is available for selection of the report form. In the first case both question and answer are collected in the same array. Elements of such an array will represent an alternating sequence of questions and answers. The number of array elements will be equal to the double number of questions, and each element will have the length equal to the length of the question.

In the second case there will be 2 arrays created in a log file. One will contain algorithm's questions and the other - corresponding answers. The number of elements for each array will be equal to the number of questions, and each element of array will have the length equal to the length of the question.

Further, the GSII64 algorithm's parameters are adjusted. From the 'Transform method' list it is necessary to choose a method, from the 'Transform mode' list to establish the direction of algorithm (encoding/decoding) and, if necessary, to set an initialization vector IV in the hexadecimal editor, which is called by clicking the **[Init. vector]** button. It is important in CBC, CFB and OFB modes to save the initialization vector value for correct back transformation.

By default the mask is rewritten into a dongle's memory before transformation. It is possible to disable this mode by removing a checkmark on 'Rewrite mask into the dongle before execution'.

To generate a log file, click on the **[Generate report]** button. In the window specify the report file name (a file with .REP extension; TRANSFOR.REP is set by default). By means of Transform operation the NSKUTIL will then call the selected algorithm, obtain answers from this algorithm and generate a log file. Transformation of algorithm's questions into answers is reversible. To obtain answers it is possible to use any of available hardware algorithms.

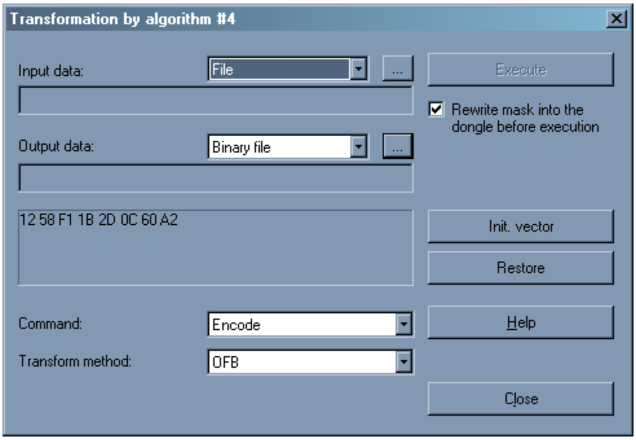
Data Encoding and Decoding with GSII64 Algorithm

Toolbar	
Menu	**Dongle / Transform**
Hotkey	**<Ctrl-T>**

The GSII64 algorithm is symmetric, which allows to encode and decode data directly by the dongle using the same key. This hardware implementation alongside with a higher strength of the GSII64 algorithm provide for a difference from the Encode/Decode operations that exist in Guardant API and perform encoding/decoding at the PC's CPU.

Preliminary encoded data can be stored in the application or in separate data files and be decoded directly before they are used. The NSKUTIL utility allows for preliminary data preparation.

It is necessary to select ‘GSII64’ type algorithm in the main window and to use any of controls mentioned above (they are accessible only if reversible algorithm is selected). The following window will appear:



Input data: File

[✓] Rewrite mask into the dongle before execution

Output data: Binary file

12 58 F1 1B 2D 0C 60 A2

Command: Encode

Transform method: OFB

The number of algorithm used for data encoding is indicated in the title of this window. In ‘Input data’ section you can specify the data to be transformed: the text or file of any format. Making a choice, you should enter the text in the editor or specify a file name. Text or file name are entered upon clicking button **[...]**, which is located to the right from the input data selector. After that the text string or a file name will be displayed in the field.

The ‘Output data’ section serves for assigning the format of encoded data. This can be text or binary file. In the first case, the encoded data presented as an array of numbers will be written to the text file created by syntax rules of one of three basic programming languages: C/C++, Pascal/Delphi or the assembler. This form of data presentation is convenient when, for example, you want to encode a text string. In the second case a binary file containing encoded sequence of bytes will be created. This form of data presentation is convenient when, for example, you want to encode a configuration file. You can enter a file name by clicking the button **[...]** located to the right from the target file type selector. File OUTPUT.REP is used by default. After that the file name will be displayed in the ‘Output data’ dialogue.

If the output data are represented as text it is necessary to choose from the ‘Programming language’ list the programming language syntax for array creation.

From the ‘Command’ list choose whether data will be encoded or decoded.

Transformation is performed in one of four GSII64 algorithm operating modes: ECB, CBC, CFB or OFB. The operating mode of algorithm is set from the 'Transform method' list. It is necessary to remember special features of GSII64 algorithm modes.

For data transformation in CBC, CFB and OFB modes it is required to set an 8-byte initialization vector IV. The initialization vector is set in the hexadecimal editor, which is called by the **[Init. vector]** button. In the editor, data can be entered manually or loaded from a file. By default NSKUTIL creates an initialization vector as a sequence of random numbers. The initialization vector is displayed in dialogue that enables to follow changes in its value in the encoding/decoding process. The **[Restore]** button serves for restoration of an initialization vector initial value.

By default the mask is rewritten into a dongle's memory before transformation. It is possible to disable this mode by removing a checkmark on 'Rewrite mask into the dongle before execution'.

After preparatory actions data transformation may start. Click **[Execute]** button at the top of the window to start the process. For speed considerations related to reducing the number of TransformEx operations, transformation will run with the longest possible blocks. Reverse transformation can be set to a different block length, depending on the task.

The transformed data will be placed into the specified target file or as an array of numbers, or as a byte sequence.

Now, in order to get access to the encoded data from the protected application, the data need to be decoded. For correct decoding in CBC, CFB and OFB modes it is necessary to set the same initialization vector as that used at encoding.

Important

The nskUtil utility version 3.3.0.1 and earlier is unable to work with Stealth II since it will not process information about the dongle and will not write / read Stealth II.

Old (current) utilities chknsk, chknsk32, chknskw and the diagnostics utility will detect new dongles as:

Stealth II LPT: Stealth II LPT

Stealth II USB: Stealth III LPT

All other utilities will work correctly. At the time of replacing Stealth I with Stealth II dongles it is OBLIGATORY to load new nskUtil and diagnostic utilities, since the old versions will not work and old diagnostic utilities will return incorrect information for Stealth II USB.

Additional Sources of Information

Should any questions arise that remain unanswered after reading this Manual, please refer to the following additional sources of information:

• README File. Can be located on the distributive media or on your computer once Guardant software has been installed on it. It contains the latest information on advanced features and updates of Guardant hardware and software.

• WWW: http://www.guardant.com – developer's web site with huge amount of reference information on Guardant protection, namely: FAQ, troubleshooting, recommendations, etc.

• Technical Support Service: e-mail: hotline@guardant.com, Telephone: +7 (095) 105-77-90. We will do our best to supply a most prompt and comprehensive reply to your query.