STTKDMA Driver

Change History

Date	Version	Reason and Change
05 July 2011	3.7.0	Added support for a secondary virtual key ladder. This allows a two stage ladder to be used on DRM versions of the driver that have the normal FK->CPCW blocked. Modified the driver to allow the CPCW keys to be managed from within the driver. This allows for a better multi process environment.
21 April 2011	3.6.0	Added decrypt container from virtual memory STTKDMA_DecryptVirtContainer
4 April 2011	3.5.5	Added thread safe STTKDMA_ProcessDMA
2 February 2011	3.5.4	Added support for M3 Secure Container.
17 January 2011	3.5.3	Added support for virtual keys that are used by DRM processes. These keys are loaded using STTKDMA_LOAD_PROTECTED_CW and STTKDMA_LOAD_PROTECTED_CPCW. Prefered use is via the STTKDMA_DecryptLadder() function. Fixed Bug2698 - Task lock error under Linux. Fixed Bug2792 - Bare (no OS) DMA time-out bug.
23 September 2010	3.5.1	Changes made for Bug2551 to be compatible with a CA vendors customer requirements for DMA's. This includes a firmware update to v3.22.

Date	Version	Reason and Change
10 September 2010	3.5.0	Add new API command STTKDMA_DecryptLadder() to make it much simpler to perform key decryption and also make decryptions thread safe. Bug fixes: Bug1469 Non-Blocking DMA Issue Fixed. Bug1569 Public ID tests added to test bench. Bug1574 Readme file added to examples Bug1575 TDES example code variable names updated for clarity Bug1576 Example code covers and comments basic API calls Bug1577 Examples updated. Bug1928, Bug2381 Improvements to the test bench. Bug1971 Documentation updated Bug2046 TDES IV values included in examples. Bug2202 Non-blocking DMA code refactored Bug2209 Better examples distributed with all customer modes. Bug2378 Firmware updated. Bug2387 Plain mode under Linux fixed. Bug2402 Byte swapping issues resolved with new API function. Bug2404 State error in STTKDMA_CurrentConfigState fixed Bug2449 Vendor-specific header file definitions updated Bug2456 Wrong error code returned for illegal key decryption. Bug2474, Bug2490 Add support and unit tests for low power mode
7 July 2010	3.4.0	Bugfix release: BriSec2380 - top level bug for release. See release notes for more details.
3 June 2010	3.3.2	Bugfix release: BriSec1557 - STTKDMA_open error case increases use count. BriSec1565 - STTKDMA_Reset API doc bug BriSec1582 - Remove dependency on firmware_defs.h BriSec1584 - Remove internal stload.h dependency BriSec1763 - Removal of DES DMA functionality from code BriSec1854 - Minor API doc clarification BriSec2025 - Full version string incorrect under Linux BriSec2121 - Remove all stload dependencies BriSec2132 - Remove obsolete code from TKDMA HAL BriSec2245 - Version string incorrect BriSec2350 - SigDMA files incorrect for some modes BriSec2360 - Increase maintainability index >= 77 BriSec2363 - Add STAPI header to Linux files BriSec2374 - 5206 Linux commands time-out BriSec2375 - Version string reports wrong mode
13 May 2010	3.3.1	Added support for 5289/5206 Linux. This has not been tested.

Date	Version	Reason and Change
16 April 2010	3.3.0	Added full support for 5206/5289 and support new mode 1 CA. ELF HASH support added and the ability for the user to build the driver against their own version of the Linux Kernel and hence no longer reliant on ST to provide a build for a particular kernel.
17 February 2010	3.2.0	Added STTKDMA_BUFFER_NO_PTE_CHECK support as a type of user space buffer, this should be used with caution. (WARNING can cause errors in buffers if not used correctly as no cache management is done). Added new error codes STTKDMA_ERROR_INVALID_TKD_KEY and STTKDMA_ERROR_INVALID_DMA_KEY. Fixed bug BriSec 1897
11 November 2009	3.1.7	Fixed BRISec 1773 - Firmware update Added uclibc support for 7105, 7111, 7141
24 June 2009	3.1.6	Fixed BRISec 1530 - STTKDMA_GetCounter millisecond value incorrect under Linux.
13 February 2009	3.1.1	Fixed BRISec 1313 - Typographical error in header file Fixed BRISec 1349 - Documentation update for clarification
22 January 2009	3.1.0	Small changed to STTKDMA_Init but no usage or api change.
1 December 2008	3.0.0	Added new error code. Change STTKDMA_Init.
24 September 2008	2.2.3	Update the error code list to include the standard STAPI error codes we use.
12 September 2008	2.2.2	Update for the linux version of STTKDMA_Init
11 August 2008	2.2.1	Added information for the DecryptContainer function.
15 July 2008	2.2.0	Added STTKDMA_TKCFG_CPCW_ALT_FORMAT
10 June 2008	2.1.0	Added STTKDMA_CurrentConfigState Fixed STTKDMA_COMMAND_DECRYPT_SW
23 April 2008	2.0.0	Removed STTKDMA_WaitForDMAToComplete Added STTKDMA_WaitForDMAToCompleteAddress Added support for Force firmware reload

Date	Version	Reason and Change
06 November 2007	1.3.0	Correction to example brisec00724. More DMA usage information. Updates reset, because a some users errors. Updated StartDma, because 32bit port causes the driver only to handle physical addresses Added missing fields from STTKDMA_DMAConfig_s
5 September 2007	1.2.6	TDES DMA Mode added to conform to FIPS Standard.
27 July 2007	1.2.5	API change Mode 2 only
13 April 2007	1.2.1	Add API for getting a counter from the tkdma IP block “STTKDMA_GetCounter”
29 March 2007	1.2.0	Changed the API for dma so that chaining mode work Changed the API for Des dma
9 January 2007	1.0.11	Add New api “STTKDMA_SetBufferType”
28 November 2006	1.0.10	Add Customer mode reporting
9 November 2006	1.0.9	More rewriting
6 November 2006	1.0.8	Add examples, rewriting bits.
27 September	1.0.7	Fix text for the DMA start commands
19 September 2006	1.0.6	Change the key format of DECRYPT_CK and DECRYPT_CW Add new flags for alt and legacy formats
19 July 2006	1.0.5	Modified DIRECT_CW command availability
14 July 2006	1.0.4	Fixed test dependency bug
13 June 2006	1.0.3	Added unique linux versions to release
19 May 2006	1.0.2	Administrative update
07 February 2006	1.0.1	Creation

1 Introduction

1.1 Driver Version

This document references driver version "STTKDMA_REL_3.7.0” for customer mode M1

1.2 Overview

This driver is an interface to the TKDMA block, which supports the following features:-

DMA Encryption/Decryption

This function allows memory to memory transfer of data buffers, with optional inline encryption or decryption. The user may select the mode and encryption/decryption algorithm to be used.

Transport key decryption

This function allows the decryption and storage of keys that will subsequently be used for transport stream decryption or copy protection of in-memory or HDD based data.

Hardware feature management

This feature provides the online enabling or disabling of various hardware functions.

1.3 Abbreviations

AES - Advanced Encryption Standard
CBC - Cipher-block chaining
CTR - Counter
CK - Common Key
CPCW - Copy Protection Control Word
CW - Control Word
DES - Data Encryption Standard
DMA - Direct Memory Access
EBC - Electronic code book
FK - Family Key
LSW - Least Significant Word
MSW - Most Significant Word
PDES - Programmable Descramblers
PID - Program Identifier
SCK - Chip-unique embedded secret value
TDES - Triple DES
TKD - Transport Key Decryption
TKDMA - Transport Key Decryption and Data Memory Access

2 Constants

2.1 Error constants

Value	Error Message	Comment
0x015a0000	STTKDMA_ERROR_DEVICE_BUSY	The device was busy. The user should retry the request
0x015a0001	STTKDMA_ERROR_OPTION_NOT_SUPPORTED	The requested option is not supported in the current security profile.
0x015a0002	STTKDMA_ERROR_DMA_FAILED	The DMA operation did not complete successfully.
0x015a0003	STTKDMA_ERROR_NOT_INITIALIZED	The driver has not been successfully initialized.
0x015a0004	STTKDMA_ERROR_COMMAND_FAILED	An error occurred in the TKDMA hardware or firmware, when issuing a command.
0x015a0005	STTKDMA_ERROR_NO_FIRMWARE_LOADED	There is no firmware loaded for the TKDMA hardware block.
0x015a0006	STTKDMA_ERROR_NO_FREE_DMA_SLOTS	There are no free slots to start a non-blocking DMA operation.
0x015a0007	STTKDMA_ERROR_DMA_ADDRESS_IN_USE	There is already an outstanding DMA with the same destination address.
0x015a0008	STTKDMA_ERROR_DMA_STATE	There has been an error in the DMA state machine.
0x015a0009	STTKDMA_ERROR_COMMAND_TIMEOUT	The driver has timed-out waiting for a response from the TKDMA hardware.
0x015a000a	STTKDMA_ERROR_INVALID_TKD_KEY	The key used for a TDES key decryption is invalid. This normally means that the two 64-bit halves of the TDES key are equal.
0x015a000b	STTKDMA_ERROR_INVALID_DMA_KEY	The key used for a TDES DMA is invalid. This normally means that the two 64-bit halves of the TDES key are equal. The DMA operation will complete, but the destination buffer will be set to zero
0x015a000c	STTKDMA_ERROR_NO_FREE_CPCW_SLOTS	If a STTKDMA_AllocateCPCWKey cannot allocate anymore keys then this error is reported.

Table 1 STTKDMA error constants

3 Types

STTKDMA_Algorithm_t

Description

Defines the cryptographic algorithms that may be used in the DMA configuration structure, `STTKDMA_DMAConfig_t`.

Definition

```
typedef enum{
    STTKDMA_ALGORITHM_PLAIN,
    STTKDMA_ALGORITHM_AES,
    STTKDMA_ALGORITHM_TDES,

}STTKDMA_Algorithm_t
```

Comments

Mode	Comment
Plain	Bypass the encryption and decryption and just do a DMA
AES	Set the encryption and decryption blocks for DMA through AES
TDES	Set the encryption and decryption blocks for DMA through TDES

Table 2 - DMA cryptographic algorithms

See Also

`STTKDMA_DMAConfig_t`
`STTKDMA_StartDMA`

STTKDMA_AlgorithmMode_t

Description

Defines the cryptographic chaining modesthat may be used in the DMA configuration structure, `STTKDMA_DMAConfig_t` in conjuction with the availebl cryptographic algorithms, defined by `STTKDMA_Algorithm_t`.

Definition

```
typedef enum{
    STTKDMA_ALGORITHM_MODE_ECB,
    STTKDMA_ALGORITHM_MODE_CBC,
    STTKDMA_ALGORITHM_MODE_CTR
}STTKDMA_ALGORITHMMode_t
```

Comments

Mode	AES	TDES	Comment
STTKDMA_ALGORITHM_ECB	Supported	Supported	Electronic code book mode
STTKDMA_ALGORITHM_CBC	Supported	Supported	Cipher block chaining mode
STTKDMA_ALGORITHM_CTR	Supported	Not supported	Counter mode

Table 3 DMA cryptographic chaining modes

Note that the driver only supports the ECB and CBC chaining modes for the TDES algorithm.

See Also

`STTKDMA_DMAConfig_t`
`STTKDMA_Algorithm_t`
`STTKDMA_StartDMA`

STTKDMA_Command_t

Description

Defines the commands available for the TKDMA engine for decrypting keys into secure storage.

Definition

```
typedef enum{
    STTKDMA_COMMAND_DECRYPT_FK,
    STTKDMA_COMMAND_DECRYPT_CK,

    STTKDMA_COMMAND_DECRYPT_CW,
    STTKDMA_COMMAND_DECRYPT_CPCW,
    STTKDMA_COMMAND_LEGACY_CW,
    STTKDMA_COMMAND_ST_SECURE_DECRYPT_CPCW,


    STTKDMA_COMMAND_DECRYPT_HK,
    STTKDMA_COMMAND_DECRYPT_PFA,
    STTKDMA_COMMAND_DECRYPT_SFA,
    STTKDMA_COMMAND_DECRYPT_KEY,
    STTKDMA_COMMAND_DECRYPT_SW,
    STTKDMA_COMMAND_DIRECT_CPCW,
    STTKDMA_COMMAND_DIRECT_CW,
    STTKDMA_COMMAND_LOAD_PROTECTED_CW,
    STTKDMA_COMMAND_LOAD_PROTECTED_CPCW,
    STTKDMA_COMMAND_DECRYPT_VIRT_KEY1,
    STTKDMA_COMMAND_DECRYPT_VIRT_KEY2
}STTKDMA_Command_t ;
```

Comments

Command	Comment
STTKDMA_COMMAND_DECRYPT_FK	Decrypts the Family Key (FK).
STTKDMA_COMMAND_DECRYPT_CK	Decrypts the Common Key (CK)
STTKDMA_COMMAND_DECRYPT_CW	Decrypts a Control Word (CW) into a secure key slot. Returns a key offset, which is passed to STPTI to reference the key.
STTKDMA_COMMAND_DECRYPT_CPCW	Decrypts a Copy Protection Control Word (CPCW) into a CPCW key slot. CPCWs are used for cryptographic DMA operations using the function STTKDMA_StartDMA.
STTKDMA_COMMAND_LEGACY_CW	Decrypt a Control Word (CW) using the legacy algorithm. Returns a key offset, which is passed to STPTI to reference the key.
STTKDMA_COMMAND_ST_SECURE_DECRYPT_CPCW	Decrypts a Copy Protection Control Word (CPCW) into a CPCW key slot using the ST secure algorithm. CPCWs are used for cryptographic DMA operations using the function STTKDMA_StartDMA.
STTKDMA_COMMAND_DECRYPT_HK	Decrypts the Hardware Key (HK).
STTKDMA_COMMAND_DECRYPT_PFA	Decrypts a value using the HK and stores the result to the o_hfm_pfa bus for Permanent Feature Activation.
STTKDMA_COMMAND_DECRYPT_SFA	Decrypts a vale using the HK and stores the result to the o_hfm_sfa bus for Session Feature Activation.
STTKDMA_COMMAND_DECRYPT_KEY	Decrypts a key using the HK and stores result to the o_mtp_key. This key can be used by the driver using the OTP_SCK_OVERRIDE flag.
STTKDMA_COMMAND_DECRYPT_SW	Decrypts a key using the HK and stores the result to the software read-back register. This is used to check that the HFM data is correct before writing permanent values.
STTKDMA_COMMAND_DIRECT_CPCW	Writes a Copy Protection Control Word (CPCW) directly to a CPCW slot without performing any decryption. CPCWs are used for cryptographic DMA operations using the function STTKDMA_StartDMA

Table 4 STTKDMA command constants

Command	Comment
STTKDMA_COMMAND_DIRECT_CW	Writes a Control Word (CW) into a secure key slot directly, without performing any decryption. Returns a key offset, which is passed to STPTI to reference the key.
STTKDMA_COMMAND_LOAD_PROTECTED_CW	Loads a protected key into a CW slot, have to provide a reference for the protected key and a slot for the CW to use. Recommend the use of STTKDMA_DecryptLadder() for this command.
STTKDMA_COMMAND_LOAD_PROTECTED_CPCW	Loads a protected key into a CPCW slot, have to provide a reference for the protected key and a slot for the CPCW to use. Recommend the use of STTKDMA_DecryptLadder() for this command.
STTKDMA_COMMAND_DECRYPT_VIRT_KEY1	This decrypts the first stage of a two stage key ladder using the SCK_ALT as the top level key. This command is only usable using STTKDMA_DecryptLadder() to ensure ease of use.
STTKDMA_COMMAND_DECRYPT_VIRT_KEY2	This decrypts the second stage of a two stage key ladder using VIRT_KEY1 as the key. This produces a CPCW that can then me used for memory to memory crypto operation. This command is only usable using STTKDMA_DecryptLadder() to ensure ease of use.

Table 4 STTKDMA command constants

Notes

Not all key decryption commands are available in each security profile. Please consult your security scheme provider for details of the key decryption operations available in your profile.

Note that the use of STTKDMA_DecryptLadder fort he above commands is now the prefered method as this is thread safe, as appropriate configuration is supplied with the command.

See Also

STTKDMA_DecryptKey
STTKDMA_DecryptLadder

STTKDMA_TKConfigCommand_t

Description

Defines configuration option and override flags, which can be set using the STTKDMA_ConfigureTK function.

Definition

```
typedef enum STTKDMA_TKConfigCommand_e
{
    STTKDMA_TKCFG_AES_NOT_TDES_FOR_CW
    STTKDMA_TKCFG_SECURE_CW_OVERRIDE
    STTKDMA_TKCFG_OTP_SCK_OVERRIDE
    STTKDMA_TKCFG_SCK_FOR_CW_OVERRIDE
    STTKDMA_TKCFG_SCK_ALT_FOR_CPCW_OVERRIDE
    STTKDMA_TKCFG_ALT_FORMAT
    STTKDMA_TKCFG_CPCW_ALT_FORMAT
    STTKDMA_TKCFG_DMA_ALT_FORMAT
    STTKDMA_TKCFG_LEGACY_KEYS
    STTKDMA_TKCFG_DMA_FIPS_FORMAT
}  STTKDMA_TKConfigCommand_t;
```

Comments

Command	Comment
STTKDMA_TKCFG_AES_NOT_TDES_FOR_CW	Selects which cryptorgraphic algorithm to use when decrypting keys with the function STTKDMA_DecryptKey. When TRUE uses AES, otherwise uses TDES.
STTKDMA_TKCFG_SECURE_CW_OVERRIDE	This overrides the Secure CW fuse to allow direct writes to CWs. (This is supported for all chips, and modes.)
STTKDMA_TKCFG_OTP_SCK_OVERRIDE	Use the hardware tied key or the OTP key instead of the SCK for a decrypt operation (Not supported for all chip modes).
STTKDMA_TKCFG_SCK_FOR_CW_OVERRIDE	Force SCK as the key issued to Decrypt CW (Not supported for all chip modes).
STTKDMA_TKCFG_SCK_ALT_FOR_CPCW_OVERRIDE	Force SCK ALT key as the key issued to Decrypt CPCW (Not support for all chip modes)
STTKDMA_TKCFG_ALT_FORMAT	Use the ALT key format and data format for the TK commands
STTKDMA_TKCFG_CPCW_ALT_FORMAT	For TDES only, when performing a decrypt CPCW, with either FK or SCK_ALT key, the CPCW key is set with a none FIPS compliant format. To get the CPCW key in a FIPS byte delivery order the flag STTKDMA_TKDMA_DMA_FIPS_FORMAT must be set. This configuration option is used in conjunction with STTKDMA_TKDMA_DMA_FIPS_FORMAT. (Not supported for all customer modes).
STTKDMA_TKCFG_DMA_ALT_FORMAT	This is used to set NIST compliant Data format for DMA instead of the default which is an ST only format used by the TKDMA IP block. For AES DMA's setting this flag puts the data into a compliant format which is the preferred case. This is the default case. For TDES this flag has an adverse effect so should be cleared and the DMA_FIPS_FORMAT below used instead. NOTE by default this is set to TRUE.
STTKDMA_TKCFG_DMA_FIPS_FORMAT	This is used only with TDES DMA, to make the data processed during a DMA to be FIPS compliant. If this is set then STTKDMA_TKCFG_DMA_ALT_FORMAT must be set to FALSE. NOTE this does not changed an AES DMA in any way.
STTKDMA_TKCFG_LEGACY_KEYS	Set the input keys to work the same way as they did in the driver pre version 1.0.7. This should be used in conjunction with STTKDMA_TKCFG_ALT_FORMAT, to get the keys in the right format inside the tkdma ip block. NOTE : This is almost always needed to be set to TRUE.

Table 1

.

DMA Type	STTKDMA_TKCFG_DMA_FIPS_FORMAT	STTKDMA_TKCFG_DMA_ALT_FORMAT
TDES	Set	Not Set
AES	Not Set	Set

Table 2

When performing crypto DMA's with TKDMA there are a set of flags depending on TDES or AES being used that ensure the DMA is in a STANDARD format.

If using AES:

Must ensure that the STTKDMA_TKCFG_DMA_ALT_FORMAT is set TRUE. This ensures that the byte delivery order for AES DMA crypto operations (encryption or decryption) conforms to the NIST specification. To ensure this the configuration is performed in the following order:

STTKDMA_TKConfig(STTKDMA_TKCFG_DMA_FIPS_FORMAT, FALSE);

STTKDMA_TKConfig(STTKDMA_TKCFG_DMA_ALT_FORMAT, TRUE);

If using TDES:

Must ensure that STTKDMA_TKCFG_DMA_FIPS_FORMAT is TRUE and STTKDMA_TKCFG_DMA_ALT_FORMAT is FALSE. This ensures that the Byte delivery order for TDES DMA crypto operation (encryption or decryption) conforms to the FIPS specification. To ensure this the following configuration is performed:

STTKDMA_TKConfig(STTKDMA_TKCFG_DMA_ALT_FORMAT, FALSE);

STTKDMA_TKConfig(STTKDMA_TKCFG_DMA_FIPS_FORMAT, TRUE);

If using the FK->CPCW or SCK_ALT->CPCW key ladder (customer mode specific) in TDES for generating the DMA key (CPCW), ensures that the keys also match the FIPS standard. The following setting is required:

STTKDMA_TKConfig(STTKDMA_TKCFG_CPCW_ALT_FORMAT, TRUE);

See Also

`STTKDMA_ConfigureTK`

STTKDMA_Buffer_t

Description

Defines the modes that Linux users buffers can support. This allows different types of allocated buffers to be used from user space as the DMA buffers.

Definition

```
typedef enum STKDMA_Buffer_e
{
   STTKDMA_BUFFER_KERNEL_COHERENT            = 0,
   STTKDMA_BUFFER_USER_SPACE                 = 1, /* default */
   STTKDMA_BUFFER_PHYSICAL                   = 2,
   STTKDMA_BUFFER_NO_PTE_CHECK               = 3
} STKDMA_Buffer_t;
```

Comments

Command	Comment
STTKDMA_BUFFER_KERNEL_COHERENT	for use only when working with a coherent kernel space buffer with user space address
STTKDMA_BUFFER_USER_SPACE	Default mode where by the DMA will copy the user space buffer in to a coherent kernel space buffer, perform the crypto DMA and then copy it back to a user space buffer.
STTKDMA_BUFFER_PHYSICAL	This is the recommended usage of buffers as produces the best performance. These are real physical address to sttkdma.
STTKDMA_BUFFER_NO_PTE_CHECK	Should only be used with buffers if it is guaranteed that the PTE flag is not set. This does not check the PTE flag, when the IOCTL is determining if the buffer if coherent or not. There is no cache management done with these buffers so this is up to the user. WARNING this will causes errors in the data, if any part of the buffers are in cache. It is recommended that this mode only be used in rare circumstances.

Table 3

See Also

STTKDMA_SetBufferType

STTKDMA_DMAConfig_t

Description

This structure holds the TKDMA config data for use when calling STTKDMA_StartDMA

Definition

```
typedef struct STTKDMA_DMAConfig_s
{
    STTKDMA_Algorithm_t     algorithm;
    STTKDMA_AlgorithmMode_t algorithm_mode;
    BOOL                    refresh;
    BOOL                    decrypt_not_encrypt;
    BOOL                    sck_override;
    U32                     iv_seed[4];
    union
    {
      U32 pkt_size;
      U32 res1;
    };
    union
    {
      U32 msc_size;
      U32 res2;
    };
/*the following three elements are ignored in STTKDMA_StartDMA and required
for STTKDMA_ProcessDMA */
    BOOL           bDMA_ALT_Format;
    BOOL           bDMA_FIPS_Format;
    STKDMA_Buffer_t bufferType;
} STTKDMA_DMAConfig_t;
```

Elements

algorithm	Selects the algorithm to be used for the DMA. Not enabled in some customer modes
algorithm_mode	Selects which mode the algorithm should use.
refresh	TRUE if the CTR counter seed value or the CBC chained data will be loaded with the value iv_seed.
decrypt_not_encrypt	TRUE if the DMA should perform a decrypt operation rather than an encrypt operation. Ignored if the algorithm is STTKDMA_ALGORITHM_PLAIN.
sck_override	Use SCK instead of CPCW key. Not enabled in some customer modes.
iv_seed	This is the seed value for CTR or the initialisation vector for CBC.
res1	should be set to 0.
res2	should be set to 0.
bDMA_ALT_Format	When set this selects STTKDMA_TKCFG_DMA_ALT_FORMAT. See STTKDMA_ConfigureTK for fuller expalnation
bDMA_FIPS_Format	When set this selects STTKDMA_TKCFG_DMA_FIPS_FORMAT. See STTKDMA_ConfigureTK for fuller expalnation
bufferType	Sets the buffer addressing mode to use for Linux user-space DMAs (Linux user-space only)

Comments

This data is used to set up the DMA mode.

In chaining mode, there is only 1 set of vectors or seed store, so unless you can guarantee you are doing back-to-back DMAs the value should be refreshed at the start of each DMA.

In CTR mode the seed value will be the last seed value + (last dma size/16).

In CBC mode the initialisation vector value will be the last 4 words of encrypted data from the last DMA. NOTE if you are doing a TDES CBC see example “tdes_fips_cbc.c” for how to set the value.

See Also

STTKDMA_StartDMA
STTKDMA_ProcessDMA
STTKDMA_ConfigureTK
STTKDMA_SetBufferType

STTKDMA_init_param_t

Description

The parameters to set up STTKDMA

Definition

```
typedef struct
{
    int             interruptLevel;
    BOOL            forceReload;
    char            *firmwareDecscription1;
    char            *firmwareDecscription2;
    int             cpcwAllocate;
} STTKDMA_init_param_t;
```

Elements

`interruptLevel`	The level at which the signature check interrupt is configured. This parameter is unused on linux systems. The default value is 12 on the 7109.
`forceReload`	Force sttkdma to reload the firmware. NOTE by default if there is firmware running on the TKDMA ip block it will not reload the firmware
`firmwareDecscription1`	Not used
`firmwareDecscription2`	Not used
`cpcwAllocate`	This parameter turns on the internal manamgement of CPCW slot allocation. This can not be mixed with manual CPCW selection, an error will occur if this happens. This is used in conjunction with STTKDMA_AllocateCPCWKey(), and STTKDMA_DeallocateCPCWKey()

NOTE:- if the firmware descriptions are null it will use the fallback image. You must have the images the right way around else the driver will not boot and may not be able to reload the firmware at a later point.

STTKDMA_Key_t

Description

This structure is a container for a 128 bit key. The format for this structure to pass the keys to the STTKDMA driver is different depending on the function using it.

STTKDMA_DecryptKey()

For this the key in general has to be in little endian, but there may need to be other byte swaps on the key if using TDES. For this reson the DecryptLadder function as been defined that resolves all of the byte swapping of keys so for all new code the new DecryptLadder should be used.

This means that the first byte of the array is the LSB of the key so key_data[0] = LSB .... key_data[15] = MSB. This is usually backwards to the way the key is passed from the transport stream as FIPS and NIST formats for keys are MSB -> LSB.

STTKDMA_DecryptLadder

This is the preferred function to use for key decrypting as this handles any key swaps that need to be done internally within the driver. The user only has to ensure that the key is in MSB = lowest memory address format. This is generally how keys are transmitted over the medium.

Definition

```
typedef struct{
    U8  key_data[16];
}STTKDMA_Key_t
```

Elements

`key_data`	The key data.

When using STTKDMA_DecryptKey()

In general for most applications the key formats are shown below, but for these to work it is essential:

AES

key_data[0] = Key[7|0]

| |

key_data[15] = Key[127|120]

This usually means that a full 16 byte reversal is required before passing the key into the driver.

key_data[0] = key_data[15] .... key_data[15] = key_data[0]

TDES

Due to the way the TDES hardware acceleration works it is necessary here to perform some extra key manipulations before passing the key to the driver. These key manipulations are 3 basic operations:

1. If performing a Key Decryption for control word decryption and the FK then the you will need to swap the four words of the key.

key[1], key[2], key[3], key[4] becomes key[4], key[3], key[2], key[1] (note words U32 not bytes)

Then perform a 16 byte swap

key_data[0] = key_data[15] .... key_data[15] = key_data[0]

2. If performing a Key Decryption for the CPCW ladder then first need to perform a swap of words 1 and 2, and words 3 and 4:

key[1]=key[2], key[2]=key[1], and key[3]=key[4], key[4]=key[3] (note words U32 not bytes)

Then perform a 16 byte swap

key_data[0] = key_data[15] .... key_data[15] = key_data[0]

When using STTKDMA_DecryptKey()

STTKDMA_Key_t MyKey;

MyKey.key_data[0] = MSB, ..., MyKey.key_data[15] = LSB

STTKDMA_config_status_t

Description

This structure is a container for all possible format manipulations that can be made to the keys and data during the encrypt/decrypt process. This is used to get the status the driver is in and to determine the different formatting options that are currently being applied.

Definition

```
typedef struct STTKDMA_config_status_s
{
  BOOL STTKDMA_TKCFG_AES_NOT_TDES;
  BOOL STTKDMA_TKCFG_SEC_CW_OVERRIDE;
  BOOL STTKDMA_TKCFG_OPT_SCK_OVERRIDE;
  BOOL STTKDMA_TKCFG_SCK_FOR_CW_OVERRIDE;
  BOOL STTKDMA_TKCFG_ALT_SCK_FOR_CPCW_OVERRIDE;
  BOOL STTKDMA_TKCFG_ALT_FORMAT;
  BOOL STTKDMA_TKCFG_DMA_ALT_FORMAT;
  BOOL STTKDMA_TKCFG_LEGACY_KEYS;
  BOOL STTKDMA_TKCFG_DMA_FIPS_FORMAT;
  BOOL FUSE_SEC_CW;
  BOOL CLEAR_CW_KEYS_ENABLED;
  BOOL CLEAR_CPCW_KEYS_ENABLED;
} STTKDMA_config_status_t;
```

Constants

STTKDMA_TKCFG_AES_NOT_TDES	return the state of the TKD configuration flag STTKDMA_TKCFG_AES_NOT_TDES_FOR_CW
STTKDMA_TKCFG_SEC_CW_OVERRIDE	return the state of the TKD configuration flag STTKDMA_TKCFG_SEC_CW_OVERRIDE
STTKDMA_TKCFG_OPT_SCK_OVERRIDE	return the state of the TKD configuration flag STTKDMA_TKCFG_OPT_SCK_OVERRIDE
STTKDMA_TKCFG_SCK_FOR_CW_OVERRIDE	return the state of the TKD configuration flag STTKDMA_TKCFG_SCK_FOR_CW_OVERRIDE
STTKDMA_TKCFG_ALT_SCK_FOR_CPCW_OVERRIDE	return the state of the TKD configuration flag STTKDMA_TKCFG_ALT_SCK_FOR_CPCW_OVERRIDE
STTKDMA_TKCFG_ALT_FORMAT	return the state of the TKD configuration flag STTKDMA_TKCFG_ALT_FORMAT
STTKDMA_TKCFG_DMA_ALT_FORMAT	return the state of the TKD configuration flag STTKDMA_TKCFG_DMA_SLT_FORMAT
STTKDMA_TKCFG_LEGACY_KEYS	return the state of the TKD configuration flag STTKDMA_TKCFG_LEGACY_KEYS
STTKDMA_TKCFG_DMA_FIPS_FORMAT	return the state of the TKD configuration flag STTKDMA_TKCFG_DMA_FIPS_FORMAT
FUSE_SEC_CW	Returns that state of the fuse SEC_CW
CLEAR_CW_KEYS_ENABLED	Current not used
CLEAR_CPCW_KEYS_ENABLED	Current not used

Comments

Note that some of the status bits are not used in all customer modes.

See Also

STTKDMA_ConfigureTK()

STTKDMA_LadderConfig_t

Description

This structure configures non-standard options for a key decryption performed by the STTKDMA_DecryptLadder() function

Definition

```
typedef enum STTKDMA_LadderConfig_e
{
    STTKDMA_CFG_AES_NOT_TDES
    STTKDMA_CFG_SECURE_CW_OVERRIDE
    STTKDMA_CFG_MTP_SCK_OVERRIDE
    STTKDMA_CFG_SCK_FOR_CW_OVERRIDE
    STTKDMA_CFG_SCK_ALT_FOR_CPCW_OVERRIDE
    STTKDMA_CFG_ALT_FORMAT
    STTKDMA_CFG_FIPS_CPCW_SWAP
    STTKDMA_CFG_NO_KEY_SWAP
} STTKDMA_LadderConfig_t;
```

Constants

STTKDMA_CFG_AES_NOT_TDES	Forces AES decryption instead of TDES when decrypting a key.
STTKDMA_CFG_SECURE_CW_OVERRIDE	Allows SECURE_CW to be set by software from which time all CWs must be decrypted; direct CW operations will no longer be valid.
STTKDMA_CFG_MTP_SCK_OVERRIDE	Allows HFM key overrides to be made. Contact Security Support for details if required.
STTKDMA_CFG_SCK_FOR_CW_OVERRIDE	Allows a CW secret key override. This is limited to certain security profiles.
STTKDMA_CFG_SCK_ALT_FOR_CPCW_OVERRIDE	Allows a CPCW key ladder override. This is limited to certain security profiles.
STTKDMA_CFG_ALT_FORMAT	Allows swapping of the key data passing into the crypto engine. In general this is only used for the DECRYPT_CW command with a TDES decryption algorithm in order to ensure FIPS standard compliance. It may also be used to work around some legacy compatibility issues. See examples of STTKDMA_DecryptLadder() for its use.
STTKDMA_CFG_FIPS_CPCW_SWAP	In general this is only used for the DECRYPT_CPCW command with a TDES decryption algorithm in order to ensure FIPS standard compliance. See examples of STTKDMA_DecryptLadder() for its use.
STTKDMA_CFG_NO_KEY_SWAP	This disables any automatic key swaps that would be performed by STTKDMA_DecryptLadder(). The user will need to perform these swaps manually. This is generally not used, except for legacy compatibility.

Comments

Note that the above configurations are not available for all security profiles. Some security profiles may have hardware-enforced rules, which will take precedence over the configurations set here.

Most of the above configurations will not be used by the majority of users - they are provided to maintain legacy compatibility.

The configurations of interested to most users are: `STTKDMA_CFG_AES_NOT_TDES` to select AES key decryption; `STTKDMA_CFG_ALT_FORMAT` when decrypting a CW using TDES, and `STTKDMA_CFG_FIPS_CPCW_SWAP` when decrypting a CPCW using TDES.

See Also

STTKDMA_KeyOperation_t

STTKDMA_DecryptLadder()

STTKDMA_KeyOperation_t

Description

This structure defines a single key decryption operation to be performed by the STTKDMA_DecryptLadder() function. Normally an array of these structures is passed to the function to perform all of the key decryption operations for a given key ladder.

This structure defines the encrypted keys to use, the decrypt command to perform on it, where the key should end up.

Definition

```
typedef struct STTKDMA_config_status_s
{
    STTKDMA_Command_t          keyCommand;
    STTKDMA_LadderConfig_t     Config;
    STTKDMA_Key_t              *encryptedKey_p;
    STTKDMA_Key_t              *PTIOffset_p;
    ST_ErrorCode_t             CmdErr;
    U8                         KeySlot;
} STTKDMA_KeyOperation_t;
```

Constants

keyCommand	This is the key decryption command operation that is required e.g. STTKDMA_COMMAND_DECRYPT_CPCW.
Config	This is a set of STTKDMA_LadderConfig_t options, which may be bitwise-OR'ed together to produce the overrides required for the key decryption.
encryptedKey_p	Pointer to the encrypted key to decrypt.
PTIOffset_p	Pointer to a buffer for returning a PTI offset of the decrypted key (for commands which return an offset)
CmdErr	This is set to the status code returned when STTKDMA_DecryptLadder peforms this command. Will be set to ST_NO_ERROR on success.
KeySlot	The key slot to decrypt the key into. For CW decryptions this must be between 0 and 49 and for CPCWs between 0 and 7. This field is ignored for other commands.

Comments

Note that all keys must be passed in MSB-first format. I.e. the most-significant byte is at the lowest memory address so that encryptedKey_p[0] is the most-significant byte and encryptedKey_p[15] (or encryptedKey_p[7] for 8-byte keys) is the least-significant byte.

When performing an STTKDMA_COMMAND_LOAD_PROTECTED_CW or STTKDMA_COMMAND_LOAD_PROTECTED_CPCW the encryptedKey_p points to an STTKDMA_Key_t structure where the last 4 bytes contains an interger reference to a protected key to use. This reference will be that supplied when the protected key was loaded by aDRM driver.

```
STTKDMA_KeyOperation_t MyOps;
STTKDMA_Key_t KeyRef;
STTKDMA_Key_t KeyOffset;

MyOps.keyCommand= STTKDMA_COMMAND_LOAD_PROTECTED_CW;
MyOps.Config = 0;
MyOps.encryptedKey_p= &KeyRef;
MyOps.PTIOffset_p = &KeyOffset;
MyOps.CmdErr =0;
MyOps.KeySlot=1;
```

/* set the LSW of the key to the virtual reference */

(U32)&KeyRef.key_data[12]= uProtectedKeyRef;

See Also

`STTKDMA_LadderConfig_t`

`STTKDMA_DecryptLadder()`

4 Functions

STTKDMA_Init	Initialises the driver.
STTKDMA_GetRevision	Returns the driver revision string and related information.
STTKDMA_Term	Terminates the driver.

Table 5 : Common functions

STTKDMA_StartDMA	Starts a DMA operation.
STTKDMA_ProcessDMA	Starts a DMA operation in the same way as STTKDMA_StartDMA.
STTKDMA_WaitForDMAToCompleteAddress	Waits a non-blocking DMA to complete.
STTKDMA_Reset	Resets intermediate key ladder stages.
STTKDMA_ReadPublicID	Reads the device public identifier.
STTKDMA_ConfigureTK	Configures special overrides modes for key ladders and encryption engines.
STTKDMA_DecryptKey	Decrypts an encrypted key into secure storage.
STTKDMA_CustomerMode	Returns the security customer mode of the device.
STTKDMA_SetBufferType	Sets the buffer addressing mode to use for Linux user-space DMAs (Linux user-space only).
STTKDMA_GetCounter	Returns the 64-bit counter value from the TKDMA hardware block.
STTKDMA_CurrentConfigState	Returns the current status of the driver override flags.
STTKDMA_DecryptContainer	Decrypts an encrypted key container
STTKDMA_DecryptVirtContainer	Decrypts an encrypted key container
STTKDMA_DecryptLadder	Performs one or more key decryptions in a thread-safe and standards-compliant manner.
STTKDMA_AllocateCPCWKey	Allocates a handle to use for a CPCW slot.
STTKDMA_DeallocateCPCWKey	Deallocates a handle to a CPCW slot.

Table 6 : STTKDMA driver-specific functions

STTKDMA_GetRevision

Description

Returns the driver revision string and related information.

Definition

```
ST_Revision_t STTKDMA_GetRevision(void);
```

Arguments

None.

Return Value

`Revision`	The driver version

Comments

The returned revision is a character string which identifies the version of the STTKDMA driver.

This function may be called at any time (including before initialization of the driver). After the driver has been initialized, the revision string contains additional information useful for security support.

When requesting driver support from the security support team, please provide the returned revision string after driver initialisation.

STTKDMA_Init

Description

Initializes the STTKDMA driver.

Definition

```
ST_ErrorCode_t STTKDMA_Init(const ST_DeviceName_t DeviceName,
                            STTKDMA_init_param_t    param);
```

Arguments

DeviceName	STAPI device name to use for initialization and registering wiht STAPI. The same device name must be provided when calling STTKDMA_Term
param	The initialization parameters.

Return Value

ST_ERROR_ALREADY_INITIALIZED	Driver is already initialized.
ST_ERROR_BAD_PARAMETER	One or more of the parameters is invalid.
ST_ERROR_INTERRUPT_INSTALL	Failed to install the driver's interrupts.
STTKDMA_ERROR_NO_FIRMWARE_LOADED	There was a problem loading firmware.
ST_ERROR_UNKNOWN_DEVICE	There was a mismatch between the security profile of the device and the firmware version attempting to be loaded. In STAPI systems with STCLOCK or STPOWER present, this error can be returned if an problem occurred when registering with either of these drivers.
ST_NO_ERROR	Driver successfully initialized.

Comments

This function creates all the resources that the driver requires for operation and performs the necessary hardware initialisation. It must be called before any other STTKDMA API functions with the exception of STTKDMA_GetRevision .

STTKDMA_Term

Description

Terminates the STTKDMA driver and frees all resources allocated by the driver.

Definition

```
ST_ErrorCode_t STTKDMA_Term(const ST_DeviceName_t DeviceName);
```

Arguments

DeviceName	Client Identifier, should match the identifier used when STTKDMA_Init .was called

Return Value

ST_ERROR_BAD_PARAMETER	The device name is invalid.
STTKDMA_ERROR_NOT_INITIALIZED	The driver has not been succesfully initialized.
ST_ERROR_SUSPENDED	The driver has been suspended due to the current power profile.
ST_NO_ERROR	Successfully terminated the driver

Comments

After calling this function, STTKDMA_Init must be called before calling any other STTKDMA API functions (with the exception of STTKDMA_GetRevision).

See Also

STTKDMA_Init

STTKDMA_StartDMA

Description

Starts a DMA operation.

Definition

```
ST_ErrorCode_t STTKDMA_StartDMA(
    STTKDMA_DMAConfig_t*    Config,
    U8*                     SourceBuffer,
    U8*                     DestBuffer,
    U32                     DataSize,
    U8                      KeyIndex,
    BOOL                    Block);
```

Arguments

`Config`	The configuration information for the requested DMA operation.
`SourceBuffer`	Pointer to the DMA source buffer. This must be a pointer to a 32-byte-aligned physical address. Under Linux user-space this must point to the type of buffer as specified by `STTKDMA_SetBufferType()`.
`DestBuffer`	Pointer to the DMA destination buffer. This must be a pointer to a 32-byte-aligned physical address. Under Linux user-space this must point to the type of buffer as specified by `STTKDMA_SetBufferType()`.
`DataSize`	Size, in bytes, of the data to DMA. For Plain, AES or TDES DMAs must be >=128 and a multiple of 32.
`KeyIndex`	Selects the CPCW index of the key to use for cryptographic DMA operations. Ignored for Plain mode DMAs or if `Config->sck_override` is set
`Block`	When TRUE the function will block until the DMA has completed. When FALSE the function will return after adding the DMA to the internal DMA queue. The user must call `STTKDMA_ProcessDMA` to determine whether the DMA has completed. Note that when running in Linux user-space this parameter must be set to TRUE.

Return Value

`ST_NO_ERROR`	The DMA was issued successfully. For blocking DMAs this also indicates that the DMA completed successfully..
`ST_ERROR_BAD_PARAMETER`	One or more of the parameters is invalid.
`STTKDMA_ERROR_NOT_INITIALIZED`	The driver has not been succesfully initialized.
`ST_ERROR_SUSPENDED`	The driver has been suspended due to the current power profile.
`STTKDMA_ERROR_OPTION_NOT_SUPPORTED`	One or more of the requested parameters isis not supported in current security profile.
`STTKDMA_ERROR_DMA_ADDRESS_IN_USE`	A non-blocking DMA already pending for the specified destination address.
`STTKDMA_ERROR_NO_FREE_DMA_SLOTS`	The internal DMA queue is full (too many outstanding non-blocking DMA requested).

STTKDMA_ERROR_COMMAND_FAILED	Internal error occurred with the TKDMA hardware block.
STTKDMA_ERROR_COMMAND_TIMEOUT	The function timed-out waiting for a blocking DMA to complete. This normally indicates a bad address was passed for either the DMA source or destination address.
STTKDMA_ERROR_INVALID_DMA_KEY	The DMA key used for a TDES cryptographic DMA is invalid. This occurs when the two 64-bit halves of a TDES-key are the same. This additional check is only supported by some devices.

Comments

When this function is called the driver determines the validity of the DMA and the configuration parameters.

The DMA operation is added to an internal queue of DMA operations to be performed by the

If the user has requested a blocking DMA, the function will block until the either the DMA has completed, ended in an error or timed-out. The result of the DMA will be indicated by the return code.

If the user has requested a non-blocking DMA, the function will return as soon as the DMA operation has been added to the internal queue. The use must call STTKDMA_ProcessDMA to determine the result of the DMA (whether it has completed or ended in an error). Note that a non-blocking DMA operation occupies a slot in the internal DMA queue until STTKDMA_ProcessDMA has been called to read-back its status. Therefore it is necessary for the user to call STTKDMA_ProcessDMA for each non-blocking DMA issued, otherwise the internal DMA queue will become full, resulting in STTKDMA_ERROR_NO_FREE_DMA_SLOTS being returned.

Non-blocking DMA operations cannot be performed from Linux user-space or if using using the STAPLER support-library in bare (no operating system) mode. In this case the function will return ST_ERROR_OPTION_NOT_SUPPORTED.

See Also

STTKDMA_DMAConfig_t
STTKDMA_ProcessDMA
STTKDMA_ProcessDMA

STTKDMA_ProcessDMA

Description

Starts a DMA operation.

Definition

```
ST_ErrorCode_t STTKDMA_ProcessDMA(
    STTKDMA_DMAConfig_t*    Config,
    U8*                     SourceBuffer,
    U8*                     DestBuffer,
    U32                     DataSize,
    U8                      KeyIndex,
    BOOL                    Block);
```

Arguments

`Config`	The configuration information for the requested DMA operation.
`SourceBuffer`	Pointer to the DMA source buffer. This must be a pointer to a 32-byte-aligned physical address. Under Linux user-space this must point to the type of buffer as specified by `STTKDMA_SetBufferType()`.
`DestBuffer`	Pointer to the DMA destination buffer. This must be a pointer to a 32-byte-aligned physical address. Under Linux user-space this must point to the type of buffer as specified by `STTKDMA_SetBufferType()`.
`DataSize`	Size, in bytes, of the data to DMA. For Plain, AES or TDES DMAs must be >=128 and a multiple of 32.
`KeyIndex`	Selects the CPCW index of the key to use for cryptographic DMA operations. Ignored for Plain mode DMAs or if `Config->sck_override` is set
`Block`	When TRUE the function will block until the DMA has completed. When FALSE the function will return after adding the DMA to the internal DMA queue. The user must call `STTKDMA_ProcessDMA` to determine whether the DMA has completed. Note that when running in Linux user-space this parameter must be set to TRUE.

Return Value

`ST_NO_ERROR`	The DMA was issued successfully. For blocking DMAs this also indicates that the DMA completed successfully..
`ST_ERROR_BAD_PARAMETER`	One or more of the parameters is invalid.
`STTKDMA_ERROR_NOT_INITIALIZED`	The driver has not been succesfully initialized.
`ST_ERROR_SUSPENDED`	The driver has been suspended due to the current power profile.
`STTKDMA_ERROR_OPTION_NOT_SUPPORTED`	One or more of the requested parameters isis not supported in current security profile.
`STTKDMA_ERROR_DMA_ADDRESS_IN_USE`	A non-blocking DMA already pending for the specified destination address.
`STTKDMA_ERROR_NO_FREE_DMA_SLOTS`	The internal DMA queue is full (too many outstanding non-blocking DMA requested).

STTKDMA_ERROR_COMMAND_FAILED	Internal error occurred with the TKDMA hardware block.
STTKDMA_ERROR_COMMAND_TIMEOUT	The function timed-out waiting for a blocking DMA to complete. This normally indicates a bad address was passed for either the DMA source or destination address.
STTKDMA_ERROR_INVALID_DMA_KEY	The DMA key used for a TDES cryptographic DMA is invalid. This occurs when the two 64-bit halves of a TDES-key are the same. This additional check is only supported by some devices.

Comments

When this function is called the driver determines the validity of the DMA and the configuration parameters. Whereas STTKDMA_StartDMA requires a prior call to STTKDMA_ConfigureTK, this function sets the ALT or FIPS mode in the Config structure. For the Linux call the buffer type which was set with a prior call to STTKSetBuffer(when using STTKDMA_StartDMA) is now also set in the Config structure. This makes the call multithreaded and re-entrant.

The DMA operation is added to an internal queue of DMA operations to be performed by the DMA engine.

If the user has requested a blocking DMA, the function will block until the either the DMA has completed, ended in an error or timed-out. The result of the DMA will be indicated by the return code.

If the user has requested a non-blocking DMA, the function will return as soon as the DMA operation has been added to the internal queue. The use must call STTKDMA_ProcessDMA to determine the result of the DMA (whether it has completed or ended in an error). Note that a non-blocking DMA operation occupies a slot in the internal DMA queue until STTKDMA_ProcessDMA has been called to read-back its status. Therefore it is necessary for the user to call STTKDMA_ProcessDMA for each non-blocking DMA issued, otherwise the internal DMA queue will become full, resulting in STTKDMA_ERROR_NO_FREE_DMA_SLOTS being returned.

Non-blocking DMA operations cannot be performed from Linux user-space or if using using the STAPLER support-library in bare (no operating system) mode. In this case the function will return ST_ERROR_OPTION_NOT_SUPPORTED.

See Also

STTKDMA_DMAConfig_t
STTKDMA_ProcessDMA

STTKDMA_WaitForDMAToCompleteAddress

Description

Waits a non-blocking DMA to complete.

Definition

```
ST_ErrorCode_t STTKDMA_WaitForDMAToCompleteAddress(
    U32 Timeout,
    BOOL* Complete,
    U32 DstAddress);
```

Arguments

`Timeout`	Maximum yime in milliseconds that the function should wait for the DMA operation to complete if it has not already completed.
`Complete`	Set to TRUE if the specified DMA operation has completed successfully.
`DstAddress`	The DMA destination address of the DMA to check. This should be the same DMA destination address used when STTKDMA_`StartDMA` was called. This address is used to uniquely identify the DMA operation, since the driver only allows a single outstanding DMA operation for a given destination address.

Return Value

`ST_NO_ERROR`	The function returned without error. The DMA may not have yet completed (if the user-specified timeout has elapsed). The user should check the value set in `Complete` to determine if the DMA has completed.
`ST_ERROR_BAD_PARAMETER`	One or more of the parameters is invalid or no outstanding DMA exists for the specified destination address.
`STTKDMA_ERROR_NOT_INITIALIZED`	The driver has not been succesfully initialized.
`ST_ERROR_SUSPENDED`	The driver has been suspended due to the current power profile.
`STTKDMA_ERROR_COMMAND_FAILED`	The requested DMA caused an internal TKDMA hardware error.
`STTKDMA_ERROR_COMMAND_TIMEOUT`	The function timed-out waiting for a blocking DMA to complete. This normally indicates a bad address was passed for either the DMA source or destination address.
`STTKDMA_ERROR_INVALID_DMA_KEY`	The requested DMA caused an internal TKDMA hardware error. This occurs when the two 64-bit halves of a TDES key are the same. This additional check is only supported by some devices.
`STTKDMA_ERROR_DMA_STATE`	An error has occurred with the state of the internal DMA queue.

Comments

This function will block until either the time-out period has expired or the DMA for the operation pointed to by the address has completed. If the time-out is set to zero the function will block until the DMA has completed.

If the DMA does not complete within the specifed timeout the function will return `ST_NO_ERROR` and `Complete` will be set FALSE.

See Also

STTKDMA_StartDMA

STTKDMA_Reset

Description

Resets intermediate key ladder stages

Definition

```
ST_ErrorCode_t STTKDMA_Reset (void);
```

Arguments

None.

Return Value

ST_NO_ERROR	All intermediate key ladder stages have been successfully invalidated
ST_ERROR_BAD_PARAMETER	One or more of the parameters is invalid.
STTKDMA_ERROR_NOT_INITIALIZED	The driver has not been succesfully initialized.
ST_ERROR_SUSPENDED	The driver has been suspended due to the current power profile.
STTKDMA_ERROR_COMMAND_FAILED	The TKDMA hardware encountered an error.
STTKDMA_ERROR_COMMAND_TIMEOUT	The function timed-out waiting for a response from the TKDMA hardware.
STTKDMA_ERROR_OPTION_NOT_SUPPORTED	The command is not available on non-secure device profiles.

Comments

This function invalidates intermediate key ladder stages, such as the CK and FK keys. Once invalidated the intermediate key ladder stages cannot be used to perform any further decryptions until a new key value is decrypted into them.

It is recommended that the user calls this function after driver initialization and before attempting any key decryption operations.

Note that is is recommended that the user calls STTKDMA_ReadPublicID after calling this function to ensure that the key ladders are correctly initialized.

See Also

STTKDMA_ReadPublicID

STTKDMA_ReadPublicID

Description

Reads the device public identifier.

Definition

```
ST_ErrorCode_t STTKDMA_ReadPublicID (STTKDMA_Key_t* Key);
```

Arguments

Key	Pointer populated with the returned public identifier

Return Value

ST_NO_ERROR	Succesfully read the public identifier
ST_ERROR_BAD_PARAMETER	The parameter is invalid.
STTKDMA_ERROR_NOT_INITIALIZED	The driver has not been succesfully initialized.
ST_ERROR_SUSPENDED	The driver has been suspended due to the current power profile.
STTKDMA_ERROR_COMMAND_FAILED	The TKDMA hardware encountered an error.
STTKDMA_ERROR_COMMAND_TIMEOUT	The function timed-out waiting for a response from the TKDMA hardware.
STTKDMA_ERROR_OPTION_NOT_SUPPORTED	The command is not available on non-secure device profiles.

Comments

Upon a successfully call to this function the public identifier will occupy the first 4-bytes of Key, ie. Key[0:3]. All other bytes of Key will be set to zero.

The user is recommended to call this function after any calls to STTKDMA_Reset to ensure that the key ladders are initialized to a known state.

See Also

STTKDMA_Key_t
STTKDMA_Reset

STTKDMA_ConfigureTK

Description

Configures special overrides modes for key ladders and encryption engines, such as the cryptographic algorithm used to decrypt keys and some key overrides.

Definition

```
ST_ErrorCode_t STTKDMA_ConfigureTK (STTKDMA_TKConfigCommand_t Command,
BOOL On);
```

Arguments

Command	The override to switch on or off
On	TRUE if the override should be switched on otherwise FALSE.

Return Value

ST_NO_ERROR	Succesfully set the specified override flag
ST_ERROR_BAD_PARAMETER	The parameter is invalid.
STTKDMA_ERROR_NOT_INITIALIZED	The driver has not been succesfully initialized.
ST_ERROR_SUSPENDED	The driver has been suspended due to the current power profile.
STTKDMA_ERROR_COMMAND_FAILED	The TKDMA hardware encountered an error.
STTKDMA_ERROR_COMMAND_TIMEOUT	The function timed-out waiting for a response from the TKDMA hardware.
STTKDMA_ERROR_OPTION_NOT_SUPPORTED	The override is not available in the current security profile

Comments

Please refer to the documentation on STTKDMA_TKConfigCommand_t for more details on the use of each configuration option. Please note that not all configuration options are availble for each security profile.

The user can use the function STTKDMA_CurrentConfigState to read the current state of the override flags.

See Also

STTKDMA_TKConfigCommand_t
STTKDMA_CurrentConfigState

STTKDMA_DecryptKey

Description

Decrypts an encrypted key into secure storage.

If writing a new implementation it is recommended that the DecryptLadder() function is used instead, as it performs all of the key swapping internally within the driver saving the user from having to do it manually.

Definition

```
ST_ErrorCode_t STTKDMA_DecryptKey (STTKDMA_Command_t Command,
    STTKDMA_Key_t* Key,
    U8 Index,
    STTKDMA_Key_t* Offset);
```

Arguments

Command	The key decryption command to perform. See STTKDMA_Command_t.
Key	Pointer to the encrypted key data.
Index	The index within the CW or CPCW tables in which the decrypted key should be stored (for CW or CPCW decryptions)
Offset	The key offset for PDES or returned key data. This is only used for some key decryption operations.

Return Value

ST_NO_ERROR	Successfully performed the specified key decryption.
ST_ERROR_BAD_PARAMETER	One of more of the parameters is invalid.
STTKDMA_ERROR_NOT_INITIALIZED	The driver has not been succesfully initialized.
ST_ERROR_SUSPENDED	The driver has been suspended due to the current power profile.
STTKDMA_ERROR_COMMAND_FAILED	The TKDMA hardware encountered an error.
STTKDMA_ERROR_COMMAND_TIMEOUT	The function timed-out waiting for a response from the TKDMA hardware.
STTKDMA_ERROR_OPTION_NOT_SUPPORTED	The specified key operation is not available in the current security profile.

Comments

Note that all commands are not available for any given security profile. Please contact your security support representative if you wish the clarify the available key decryption operations for your security profile.

The Index parameter should be between 0 and 7 for CPCW decryptions and between 0 and 49 for CW decryptions. For all other key decryptions, its valueis ignored.

The secret key used for the key decryption operation and the algorithm of the decrypt operation takes account of the current overrides set with STTKDMA_ConfigureTK and any hardware-enforced overrides for the current security profile.7

See Also

STTKDMA_Key_t
STTKDMA_Command_t

STTKDMA_CustomerMode

Description

Returns the security customer mode of the device.

Definition

```
U32 STTKDMA_CustomerMode(void);
```

Comments

Returns the security mode of the device. The security mode should match the mode of the STTKDMA driver and firmware that the user is using. If this is not the case, the user must contact their security support representative to obtain the correct STTKDMA driver and firmware.

STTKDMA_SetBufferType()

Description

Sets the buffer addressing mode to use for Linux user-space DMAs (Linux user-space only).

Definition

```
ST_ErrorCode_t STTKDMA_SetBufferType(STTKDMA_Buffer_t bufferType);
```

Arguments

bufferType	Which addressing mode to use for DMA buffers passed to STTKDMA_StartDMA. See STTKDMA_Buffer_t for information on the supported buffer addressing modes.

Return Value

ST_NO_ERROR	Successfully set the required buffer addressing mode.
ST_ERROR_BAD_PARAMETER	An invalid addressing mode was requested.
STTKDMA_ERROR_NOT_INITIALIZED	The driver has not been succesfully initialized.

Comments

This function is only available under the Linux user-space implementation of the STTKDMA driver.

If the function is not called, the driver default addressing mode for DMA buffers is used (STTKDMA_BUFFER_USER_SPACE).

See Also

STTKDMA_Buffer_t

STTKDMA_GetCounter

Description

Returns the 64-bit counter value from the TKDMA hardware block.

Definition

```
U32 STTKDMA_GetCounter(
    U32 *HiCounter,
    U32 *LowCounter,
    long long *mSec);
```

Arguments

HiCounter	A pointer to set with the 32 most-signifcant counter bits.
lowCounter	A pointer to set with the 32 least-signifcant counter bits.
mSec	A pointer to set with the counter value translated to milli-seconds.

Return Value

ST_NO_ERROR	Successfully read the TKDMA counter.
ST_ERROR_BAD_PARAMETER	One of more of the parameters is invalid.
STTKDMA_ERROR_NOT_INITIALIZED	The driver has not been succesfully initialized.
ST_ERROR_SUSPENDED	The driver has been suspended due to the current power profile.
STTKDMA_ERROR_COMMAND_FAILED	The TKDMA hardware encountered an error.
STTKDMA_ERROR_COMMAND_TIMEOUT	The function timed-out waiting for a response from the TKDMA hardware.
STTKDMA_ERROR_OPTION_NOT_SUPPORTED	The counter is not availlable on the current security profile.

Comments

The counter value is read from the internal TKDMA hardware counter. The counter gives a count value since the TKDMA hardware block was started. The millisecond value returns a value scaled against the number of counter ticks per millisecond.

The time taken for the counter to wrap varies according to the individual device. For current devices the minimum counter wrap time is approximately 9.22 x 10^10 seconds.

The TKDMA hardware counter cannot be written directly from the host processor.

STTKDMA_CurrentConfigState

Description

Returns the current status of the driver override flags.

Definition

```
ST_ErrorCode_t STTKDMA_CurrentConfigState(STTKDMA_config_status_t
*status);
```

Arguments

status	Pointer to a struct to be populated the state of the driver overrides.

Return Value

ST_NO_ERROR	Successfully read the driver override flags.
ST_ERROR_BAD_PARAMETER	The passed parameter is invalid.
STTKDMA_ERROR_NOT_INITIALIZED	The driver has not been succesfully initialized.
ST_ERROR_SUSPENDED	The driver has been suspended due to the current power profile.

Comments

This function is useful during development to determine the state the driver override flags at any given time. This can help to diagnose unexpected cryptographic results by indicating which overrides are set.

See Also

STTKDMA_config_status_t

ST_ErrorCode_t STTKDMA_DecryptContainer()

Description

Decrypts an encrypted key container, Physical memory.

Definition

```
ST_ErrorCode_t STTKDMA_DecryptContainer( U8* pKey1,
                                         U8* pKey2,
                                         U8* pEContainer);
```

Arguments

pKey1	Pointer to the encrypted protecting key for the key container.
pKey2	Pointer to the encrypted key for the container (encrypted with pKey1).
pEContainer	Pointer to the encrypted container structure (encrypted with pKey2).

Return Value

ST_NO_ERROR	Successfully decrypted the key container.
ST_ERROR_BAD_PARAMETER	One of more of the parameters is invalid.
STTKDMA_ERROR_NOT_INITIALIZED	The driver has not been succesfully initialized.
ST_ERROR_SUSPENDED	The driver has been suspended due to the current power profile.
STTKDMA_ERROR_COMMAND_FAILED	The TKDMA hardware encountered an error. This could be due the an incorrect key, container pointer or a failure of the container checksum.
STTKDMA_ERROR_COMMAND_TIMEOUT	The function timed-out waiting for a response from the TKDMA hardware.
STTKDMA_ERROR_OPTION_NOT_SUPPORTED	The counter is not availlable on the current security profile.

Comments

This function decrypts a key container into a protected area of memory that the host does not have access to. This decrypted container can then be used to load keys by other security devices

This functionality is only available in some security profiles.

Please refer to related security documentation for details on the key container format and encryption scheme.

ST_ErrorCode_t STTKDMA_DecryptVirtContainer()

Description

Decrypts an encrypted key container, Form virtual memory.

Definition

```
ST_ErrorCode_t STTKDMA_DecryptVirtContainer( U8* pKey1,
                                             U8* pKey2,
                                             U8* pEContainer);
```

Arguments

pKey1	Pointer to the encrypted protecting key for the key container.
pKey2	Pointer to the encrypted key for the container (encrypted with pKey1).
pEContainer	Pointer to the encrypted container structure (encrypted with pKey2).

Return Value

ST_NO_ERROR	Successfully decrypted the key container.
ST_ERROR_BAD_PARAMETER	One of more of the parameters is invalid.
STTKDMA_ERROR_NOT_INITIALIZED	The driver has not been succesfully initialized.
ST_ERROR_SUSPENDED	The driver has been suspended due to the current power profile.
STTKDMA_ERROR_COMMAND_FAILED	The TKDMA hardware encountered an error. This could be due the an incorrect key, container pointer or a failure of the container checksum.
STTKDMA_ERROR_COMMAND_TIMEOUT	The function timed-out waiting for a response from the TKDMA hardware.
STTKDMA_ERROR_OPTION_NOT_SUPPORTED	The counter is not availlable on the current security profile.

Comments

This function decrypts a key container into a protected area of memory that the host does not have access to. This decrypted container can then be used to load keys by other security devices

This functionality is only available in some security profiles.

Please refer to related security documentation for details on the key container format and encryption scheme.

STTKDMA_DecryptLadder()

Description

Performs one or more key decryptions in a thread-safe and standards-compliant manner. This is the prefered mechanism for decrypting any keys as all key swaps are done internally within the driver. All keys are suppied in MSB lowest address format.

Definition

```
ST_ErrorCode_t STTKDMA_DecryptLadder(STTKDMA_KeyOperations_t *pKeyOp
                                     U32                     uNumberOfOps);
```

Arguments

pKeyOps	An array of at least STTKDMA_KeyOperation_t struct.
uNumberOfOps	The number of STTKDMA_KeyOperation_tpointed to by pKeyOps.

Return Value

ST_NO_ERROR	Successfully performed all of the requested key decryptions.
ST_ERROR_BAD_PARAMETER	One of more of the parameters is invalid.
STTKDMA_ERROR_NOT_INITIALIZED	The driver has not been succesfully initialized.
ST_ERROR_SUSPENDED	The driver has been suspended due to the current power profile.
STTKDMA_ERROR_COMMAND_FAILED	The TKDMA hardware encountered an error.
STTKDMA_ERROR_COMMAND_TIMEOUT	The function timed-out waiting for a response from the TKDMA hardware.
STTKDMA_ERROR_OPTION_NOT_SUPPORTED	One of the specified key operation is not available in the current security profile.

Please note that this function performs each specified key operation in sequence until either all have been completed or it encounters an error. If an error is encountered, the Error field of the corresponding STTKDMA_KeyOperation_t structure is set to the corresponding error code and this error code is also returned.

Comments

This function is the recommended way of performing key decryptions. It allows multiple key decryptions to be performed in a thread-safe manner and without the need for users to manipulate the byte ordering of their keys.

Key decryptions are performed according to the relevant (FIPS or NIST) standard

Note that all keys must be passed in MSB-first format. I.e. the most-significant byte is at the lowest memory address so that Key[0] is the most-significant byte and Key[15] (or Key[7] for 8-byte keys) is the least-significant byte.

The configurations of interested to most users are: STTKDMA_CFG_AES_NOT_TDES to select AES key decryption; STTKDMA_CFG_ALT_FORMAT when decrypting a CW using TDES, and STTKDMA_CFG_FIPS_CPCW_SWAP when decrypting a CPCW using TDES.

STTKDMA_DecryptLadder is the only way to process the virtual two stage ladder usign the STTKDMA_COMMAND_DECRYPT_VIRT_KEY1, STTKDMA_COMMAND_DECRYPT_VIRT_KEY2 commands. These commands have also been produced to only support FIPS and NIST compliant ladders see the example below for their usage.

Please refer to the code below and the compilable examples, packaged with the driver, for demonstration of standards-compliant key decryption

Examples

These two examples assume that the driver has already been initialized:

1. Setting direct (clear) CWs

```
STTKDMA_Key_t RetKey[5];
STTKDMA_Key_t CW;
ST_ErrorCode_t err;
U8 TestKeyCW[16]= {0,1,2,3,4,5,6,7,8,9,0xa,0xb,0xc,0xd,0xe,0xf};

/* Set-up the key ladder for 5 clear CW's (slots 0,5,10,15,20)*/
STTKDMA_KeyOperation_t  LadderOps[] = {
    {STTKDMA_COMMAND_DIRECT_CW, 0, &CW, &RetKey[0], 0,  0},
    {STTKDMA_COMMAND_DIRECT_CW, 0, &CW, &RetKey[1], 0,  5},
    {STTKDMA_COMMAND_DIRECT_CW, 0, &CW, &RetKey[2], 0, 10},
    {STTKDMA_COMMAND_DIRECT_CW, 0, &CW, &RetKey[3], 0, 15},
    {STTKDMA_COMMAND_DIRECT_CW, 0, &CW, &RetKey[4], 0, 20}
};

memcpy(&CW, TestKeyCW, sizeof(STTKDMA_Key_t));
if ((err=STTKDMA_DecryptLadder(LadderOps, 5))!= ST_NO_ERROR)
{
    printf("Error Decrypting CW's : 0x%08x\n", err);
}
```

2. Decrypting a CW using a 2-stage TDES key ladder

```
STTKDMA_Key_t RetKey;
STTKDMA_Key_t ECK, ECW;

/* Set-up the key ladder for the STTKDMA_DecryptLadder using TDES */
STTKDMA_KeyOperation_t  LadderOps[] = {
    {STTKDMA_COMMAND_DECRYPT_CK, 0, &ECK, NULL, 0, 0},
    {STTKDMA_COMMAND_DECRYPT_CW, STTKDMA_CFG_ALT_FORMAT, &ECW, &RetKey, 0,
0}
};
memcpy(&ECK, TestKeyECK, sizeof(STTKDMA_Key_t));
memcpy(&ECW, TestKeyECW, sizeof(STTKDMA_Key_t));
if ((err=STTKDMA_DecryptLadder(LadderOps, 2))!= ST_NO_ERROR)
{
    printf("Error Decrypting CPCW's : 0x%08x\n", err);
```

```
}
```

3. Setting a direct (clear) CPCW

```
STTKDMA_Key_t    CPCW;
ST_ErrorCode_t err;
static U8 TestKeyCPCW[16]= {0,1,2,3,4,5,6,7,8,9,0xa,0xb,0xc,0xd,0xe,0xf};

/* Set-up the key ladder for the DIRECT CPCW copying the same key to CPCW 0,1,2,3,
and 4. As there is no returned key for the DIRECT_CPCW we do not need to supply a
returned key function */
STTKDMA_KeyOperation_t  LadderOps[] = {
    {STTKDMA_COMMAND_DIRECT_CPCW, 0, &CPCW, NULL, 0, 0},
    {STTKDMA_COMMAND_DIRECT_CPCW, 0, &CPCW, NULL, 0, 1},
    {STTKDMA_COMMAND_DIRECT_CPCW, 0, &CPCW, NULL, 0, 2},
    {STTKDMA_COMMAND_DIRECT_CPCW, 0, &CPCW, NULL, 0, 3},
    {STTKDMA_COMMAND_DIRECT_CPCW, 0, &CPCW, NULL, 0, 4},
};
memcpy(&CPCW, TestKeyCPCW, sizeof(STTKDMA_Key_t));
if ((err=STTKDMA_DecryptLadder(LadderOps, 5))!= ST_NO_ERROR)
{
    printf("Error Decrypting CPCW's : 0x%08x\n", err);
}
```

4. Decrypting a CPCW using a 2-stage TDES key ladder

```
STTKDMA_Key_t EFK, ECPCW;
static U8 TestKeyEFK[16]= {0xf,0xe,0xd,0xc,0xb,0xa,9,8,7,6,5,4,3,2,1,0};
static U8 TestKeyECPCW[16]= {0,1,2,3,4,5,6,7,8,9,0xa,0xb,0xc,0xd,0xe,0xf};

/* Set-up a two-stage key ladder using TDES */
STTKDMA_KeyOperation_t  LadderOps[] = {
    {STTKDMA_COMMAND_DECRYPT_FK, 0, &EFK, NULL, 0, 0},
    {STTKDMA_COMMAND_DECRYPT_CPCW, STTKDMA_CFG_FIPS_CPCW_SWAP, &ECPCW, NULL, 0, 0},
};

memcpy(&EFK, TestKeyEFK, sizeof(STTKDMA_Key_t));
memcpy(&ECPCW, TestKeyECPCW, sizeof(STTKDMA_Key_t));
if ((err=STTKDMA_DecryptLadder(LadderOps, 2))!= ST_NO_ERROR)
{
    printf("Error Decrypting CPCW's : 0x%08x :\n", err);
}
```

5. Multiple key ladders in a single operation

The array of key decryption commands is performed sequentially, which allows for multiple key ladders to be decrypted using a single function call.

In this example we show a series of ladder operations to decrypt a series of CPCW keys, each of which uses a different protecing FK key. We follow this by setting a series of direct (clear) CPCW keys.

```
STTKDMA_KeyOperation_t  LadderOps[] = {
{STTKDMA_COMMAND_DECRYPT_FK, 0, &EFK1, NULL, 0, 0},
{STTKDMA_COMMAND_DECRYPT_CPCW, STTKDMA_CFG_FIPS_CPCW_SWAP, &ECPCW1, NULL, 0, 1},
{STTKDMA_COMMAND_DECRYPT_FK, 0, &EFK2, NULL, 0, 0},
{STTKDMA_COMMAND_DECRYPT_CPCW, STTKDMA_CFG_FIPS_CPCW_SWAP, &ECPCW2, NULL, 0, 2},
{STTKDMA_COMMAND_DECRYPT_FK, 0, &EFK3, NULL, 0, 0},
{STTKDMA_COMMAND_DECRYPT_CPCW, STTKDMA_CFG_FIPS_CPCW_SWAP, &ECPCW3, NULL, 0, 3},
{STTKDMA_COMMAND_DIRECT_CPCW, 0, &CPCW4, NULL, 0, 4},
{STTKDMA_COMMAND_DIRECT_CPCW, 0, &CPCW5, NULL, 0, 5},
```

```
{STTKDMA_COMMAND_DIRECT_CPCW, 0, &CPCW6, NULL, 0, 6},
};
```

6. Loading a Protected CW or CPCW

```
STTKDMA_Key_t KeyProtectedRefCW;
STTKDMA_Key_t KeyProtectedRefCPCW;
STTKDMA_Key_t KeyRef;

/* Set-up a two-stage key ladder using LOAD_PROTECTED_CW and CPCW */
STTKDMA_KeyOperation_t  LadderOps[] = {
    {STTKDMA_COMMAND_LOAD_PROTECTED_CW, 0, &KeyProtectedRefCW, &KeyRef, 0, 0},
    {STTKDMA_COMMAND_LOAD_PROTECTED_CPCW, 0,&KeyProtectedRefCPCW, NULL, 0, 0},
};

/* clear the reference key used by PTI to access the key */
memset(&KeyRef, 0, sizeof(STTKDMA_Key_t));

/* setup the reference for the protected keys these are normally supplied
by the DRM scheme but constants are used here. there are 56 in total. */
*(U32*)&KeyProtectedRefCW[12]= 32;
*(U32*)&KeyProtectedRefCW[12]= 19;

if ((err=STTKDMA_DecryptLadder(LadderOps, 2))!= ST_NO_ERROR)
{
    printf("Error Decrypting CPCW's : 0x%08x :\n", err);
}
```

7. Using the Two Stage Virtual Ladder

```
STTKDMA_Key_t KeyEncV1;
STTKDMA_Key_t KeyEncV2;

STTKDMA_KeyOperation_t  LadderOps[2];

/* Set-up a two-stage key ladder using LOAD_PROTECTED_CW and CPCW */
if(AES_Ladder)
{
    LadderOps[]= {
        {STTKDMA_COMMAND_DECRYPT_VIRT_KEY1, STTKDMA_CFG_AES_NOT_TDES,
         &KeyEncV1, NULL, 0, 0},
        {STTKDMA_COMMAND_DECRYPT_VIRT_KEY2, STTKDMA_CFG_AES_NOT_TDES,
         &KeyEncV2, NULL, 0, 0}
    };
}
else
{
    LadderOps[] = {
        {STTKDMA_COMMAND_DECRYPT_VIRT_KEY1, 0, &KeyEncV1, NULL, 0, 0},
        {STTKDMA_COMMAND_DECRYPT_VIRT_KEY2, STTKDMA_CFG_FIPS_CPCW_SWAP,
         &KeyEncV2, NULL, 0, 0}
    };
}
```

```
/* setup the required cpcw key slot */
LadderOps[1].KeySlot = Index;

/* clear the encrypted keys */
memset(&KeyEncV1, 0, sizeof(STTKDMA_Key_t));
memset(&KeyEncV2, 0, sizeof(STTKDMA_Key_t));

if ((err=STTKDMA_DecryptLadder(LadderOps, 2))!= ST_NO_ERROR)
{
    printf("Error Decrypting CP Ladder: 0x%08x :\n", err);
}
```

See Also

STTKDMA_KeyOperation_t
STTKDMA_LadderConfig_t
STTKDMA_Command_t
STTKDMA_Key_t

ST_ErrorCode_t STTKDMA_AllocateCPCWKey()

Description

Allocates a CPCW key slot for use with crypto DMA's, without needing to know about the underlying hardware.

Definition

```
ST_ErrorCode_t STTKDMA_AllocateCPCWKey(U8 *cpcwKey);
```

Arguments

cpcwKey	contains the virtual key index after allocation.

Return Value

ST_NO_ERROR	Successfully decrypted the key container.
ST_ERROR_BAD_PARAMETER	One of more of the parameters is invalid.
STTKDMA_ERROR_NOT_SUPPORTED	The driver has not been succesfully initialized.
STTKDMA_ERROR_NO_FREE_CPCW_SLOTS	The counter is not availlable on the current security profile.

Comments

This function allocates a hardware CPCW key slot to a virtual slot so that cryoto DMA's can be performed without the need to know which slot is being used. This is best used when a system is being developed for multiple rolls such as HDD encryption/decryption, DRM and PVR activities. This ensures that there are always hardware slots available as a slot is allocated, useded and then released.

ST_ErrorCode_t STTKDMA_DeallocateCPCWKey()

Description

Deallocates a CPCW key slot for use with crypto DMA's, without needing to know about the underlying hardware.

Definition

```
ST_ErrorCode_t STTKDMA_AllocateCPCWKey(U8 cpcwKey);
```

Arguments

cpcwKey	The virtual key index to return back to the system.

Return Value

ST_NO_ERROR	Successfully deallocated the CPCW slot.
ST_ERROR_BAD_PARAMETER	Key index does exist.
STTKDMA_ERROR_NOT_SUPPORTED	The driver has not been succesfully initialized.

Comments

This function deallocates a previously allocated CPCW key slot. This is best used when a system is being developed for multiple rolls such as HDD encryption/decryption, DRM and PVR activities. This ensures that there are always hardware slots available as a slot is allocated, used and then released.