FEITIAN

iR301-U Smart Card Reader

Developer Guide

Revision History:

Date	Revision	Description
Jul. 2012	V1.0	Release of the first version
Mar. 2013	V1.1	Release of the second version
April. 2014	V1.2	Release of the third version
May, 2014	V1.3	Fixed describe error in FtDukptSetEncMod API
June 4, 2014	V1.4	Add background running support and add bR301 support

Software Developer's Agreement

All Products of Feitian Technologies Co., Ltd. (Feitian) including, but not limited to, evaluation copies, diskettes, CD-ROMs, hardware and documentation, and all future orders, are subject to the terms of this Agreement. If you do not agree with the terms herein, please return the evaluation package to us, postage and insurance prepaid, within seven days of their receipt, and we will reimburse you the cost of the Product, less freight and reasonable handling charges.

1. Allowable Use – You may merge and link the Software with other programs for the sole purpose of protecting those programs in accordance with the usage described in the Developer's Guide. You may make archival copies of the Software.

2. Prohibited Use – The Software or hardware or any other part of the Product may not be copied, reengineered, disassembled, decompiled, revised, enhanced or otherwise modified, except as specifically allowed in item 1. You may not reverse engineer the Software or any part of the product or attempt to discover the Software's source code. You may not use the magnetic or optical media included with the Product for the purposes of transferring or storing data that was not either an original part of the Product, or a Feitian provided enhancement or upgrade to the Product.

3. Warranty – Feitian warrants that the hardware and Software storage media are substantially free from significant defects of workmanship or materials for a time period of twelve (12) months from the date of delivery of the Product to you.

4. Breach of Warranty – In the event of breach of this warranty, Feitian's sole obligation is to replace or repair, at the discretion of Feitian, any Product free of charge. Any replaced Product becomes the property of Feitian.

Warranty claims must be made in writing to Feitian during the warranty period and within fourteen (14) days after the observation of the defect. All warranty claims must be accompanied by evidence of the defect that is deemed satisfactory by Feitian. Any Products that you return to Feitian, or a Feitian authorized distributor, must be sent with freight and insurance prepaid.

EXCEPT AS STATED ABOVE, THERE IS NO OTHER WARRANTY OR REPRESENTATION OF THE PRODUCT, EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

5. Limitation of Feitian's Liability – Feitian's entire liability to you or any other party for any cause whatsoever, whether in contract or in tort, including negligence, shall not exceed the price you paid for the unit of the Product that caused the damages or are the subject of, or indirectly related to the cause of action. In no event shall Feitian be liable for any damages caused by your failure to meet your obligations, nor for any loss of data, profit or savings, or any other consequential and incidental damages, even if Feitian has been advised of the possibility of damages, or for any claim by you based on any third-party claim.

6. Termination – This Agreement shall terminate if you fail to comply with the terms herein. Items 2, 3, 4 and 5 shall survive any termination of this Agreement.

Contents

Chapter 1. Overview **1**

Chapter 2. New features **2**

Chapter 3. Definitions **3**

Error codes 3

Chapter 4. API Reference **4**

3.1 SCardEstablishContext 4
3.2SCardListReaders 4
3.3SCardConnect 5
3.4 SCardStatus 7
3.5 SCardGetAttrib 8
3.6 SCardTransmit 9
3.7 SCardGetStatusChange 10
3.8 SCardDisconnect 11
3.9 SCardReleaseContext 12
4.0 FtGetSerialNum(private interface) 12
4.1 FtWriteFlash (private interface) 13
4.2 FtReadFlash(private interface) 14
4.3 FtSetTimeout (private interface) 15
4.4 FtDukptInit (private interface) 16
4.5 FtDukptSetEncMod (private interface) 17
4.6 FtDukptGetKSN(private interface) 17
4.7 FtDidEnterBackground(private interface) 22
4.8 ReaderInterfaceDelegate(private interface) 23

4.8.1 readerInterfaceDidChange(private interface) 23

4.8.1 cardInterfaceDidDetach(private interface) 23

4.9 @interface ReaderInterface(private interface) 23

4.9.1 setDelegate(private interface) 23

4.9.2 isReaderAttached(private interface) 24

4.9.3 isCardAttached(private interface) 24

5.0 Support background mode 27
5.1 Support bR301 also 28

Chapter 1. Overview

This chapter describes how to develop iR301U reader applications, including the development interfaces supported by the product (iR301U) and how to develop applications based on these interfaces.

FEITIAN iR301-U is specially engineered to accommodate a range of smart card applications. Developers use it as a platform to generate and deploy related products and services. Moreover, FEITIAN iR301-U is a terminal unit which is seamlessly integrated to all major systems of operation. Additional features such as the built-in inclusive support for different smart card interfaces has facilitated the wide scale and cross industry adoption of iR301-U.

iR301-U suits customers where security concerns are the most salient and satisfies the demand for a flexible solution for ID authentication, e-commerce, e-payment, information security and access control.

iR301-U and the rest of FEITIAN's line of smart card readers offer each customer a complete solution for all manner of utilizations.

Chapter 2. New features

The new reader has been published, included key management and data space.

New features:

1. More security

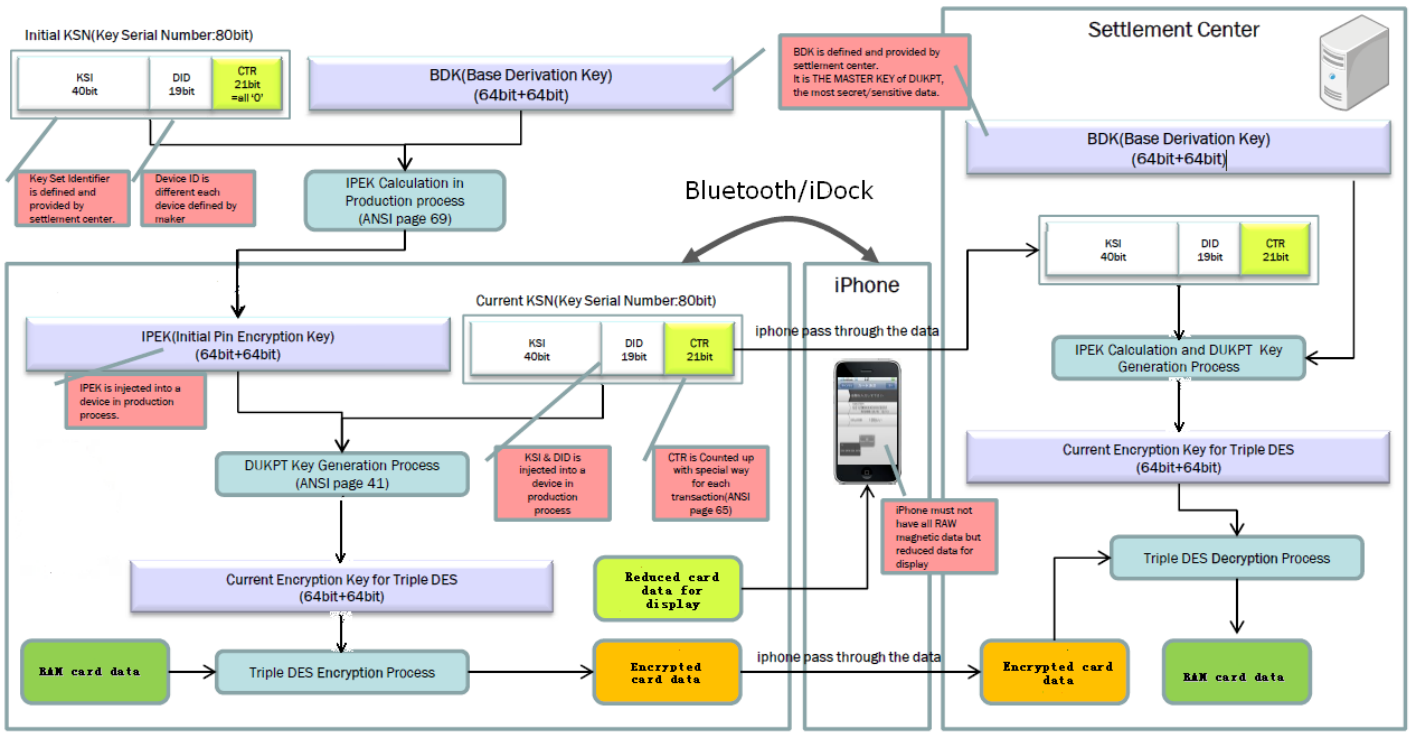
   DUKPT (Derived Unique Key Per Transaction) is a key management scheme in which for every transaction, a unique key is used which is derived from a fixed key. Therefore, if a derived key is compromised, future and past transaction data are still protected since the next or prior keys cannot be determined easily. DUKPT is specified in ANSI X9.24 part 1.

   The security standard defines card data of chip card can't be in iOS. Card reader device must be encrypting all transfer information before sending to iOS.

2. Supported 256bytes data space for customer, customer can through API to write/read data from reader.

We through below picture to give customer a clear concept of DUKPT:

http://en.wikipedia.org/wiki/DUKPT



Chapter 3. Definitions

Error codes

The following is a list of commonly used errors. Since different cards produce different errors they must map over to these error messages.

SCARD_S_SUCCESS
SCARD_E_INVALID_VALUE
SCARD_E_INVALID_PARAMETER
SCARD_E_INVALID_HANDLE
SCARD_E_INSUFFICIENT_BUFFER
SCARD_E_NO_SMARTCARD
SCARD_E_READER_UNAVAILABLE
SCARD_E_UNSUPPORTED_FEATURE
SCARD_F_COMM_ERROR
SCARD_E_NOT_TRANSACTED

Chapter 4. API Reference

3.1 SCardEstablishContext

Synopsis:

```
#include <winscard.h>
LONG SCardEstablishContext(DWORD dwScope,
    /*@unused@*/ LPCVOID pvReserved1,
    /*@unused@*/ LPCVOID pvReserved2,
    LPSCARDCONTEXT phContext);
```

Parameters:

dwScope	IN	Scope of the establishment
pvReserved1		unused
pvReserved2		unused
phContext	OUT	Returned reference to this connection

Description:

This function creates a communication context to the PC/SC Resource Manager. This must be the first function called in a PC/SC application.

Value of dwScope Meaning

SCARD_SCOPE_USER	Not used
SCARD_SCOPE_TERMINAL	Not used
SCARD_SCOPE_GLOBAL	Not used
SCARD_SCOPE_SYSTEM	Services on the local machine

Example:

```
SCARDCONTEXT    hContext;
LONG            rv;
rv = SCardEstablishContext(SCARD_SCOPE_SYSTEM, NULL, NULL, &hContext);
```

Returns:

SCARD_S_SUCCESS	Successful
SCARD_E_INVALID_VALUE	Invalid scope type passed
SCARD_E_INVALID_PARAMETER	Invalid parameter

3.2SCardListReaders

Synopsis:

```
#include <winscard.h>
LONG SCardListReaders(SCARDCONTEXT hContext,
    /*@null@*/ /*@out@*/ LPCSTR mszGroups,
    /*@null@*/ /*@out@*/ LPSTR mszReaders,
    /*@out@*/ LPDWORD pcchReaders);
```

Parameters:

hContext	IN	Connection context to the PC/SC Resource Manager
mszGroups	IN	List of groups to list readers (not used)
mszReaders	OUT	Multi-string with list of readers
pcchReaders	OUT	Size of multi-string buffer including NULL's

Description:

This function returns a list of currently available readers on the system. mszReaders is a pointer to a character string that is allocated by the application. If the application sends mszGroups and mszReaders as NULL then this function will return the size of the buffer needed to allocate in pcchReaders. The reader names is a multi-string and separated by a nul character ('\0') and ended by a double null character. "Reader A\0Reader B\0\0".

Example:

```
SCARDCONTEXT    hContext;
LPSTR           mszReaders;
DWORD           dwReaders;
LONG            rv;
rv = SCardEstablishContext(SCARD_SCOPE_SYSTEM, NULL, NULL, &hContext);
rv = SCardListReaders(hContext, NULL, NULL, &dwReaders);
mszReaders = malloc(sizeof(char)*dwReaders);
rv = SCardListReaders(hContext, NULL, mszReaders, &dwReaders);
```

Returns:

SCARD_S_SUCCESS	Successful
SCARD_E_INVALID_HANDLE	Invalid Scope Handle
SCARD_E_INSUFFICIENT_BUFFER	Reader buffer not large enough
SCARD_E_INVALID_PARAMETER	Invalid parameter

3.3SCardConnect

Synopsis:

```
#include <winscard.h>
LONG SCardConnect( SCARDCONTEXT hContext,
    LPCSTR szReader,
    DWORD dwShareMode,
    DWORD dwPreferredProtocols,
    LPSCARDHANDLE phCard,
```

LPDWORD pdwActiveProtocol);

Parameters:

hContext	IN	Connection context to the PC/SC Resource Manager
szReader	IN	Reader name to connect to
dwShareMode	IN	Mode of connection type: exclusive or shared
dwPreferredProtocols	IN	Desired protocol use
phCard	OUT	Handle to this connection
pdwActiveProtocol	OUT	Established protocol to this connection.

Description:

This function establishes a connection to the friendly name of the reader specified in szReader. The first connection will power up and perform a reset on the card. Value of dwShareMode Meaning
SCARD_SHARE_SHARED This application will allow others to share the reader
SCARD_SHARE_EXCLUSIVE This application will NOT allow others to share the reader
SCARD_SHARE_DIRECT Direct control of the reader, even without a card
SCARD_SHARE_DIRECT can be used before using SCardControl() to send control commands to the reader even if a card is not present in the reader.
Value of dwPreferredProtocols Meaning
SCARD_PROTOCOL_T0 Use the T=0 protocol
SCARD_PROTOCOL_T1 Use the T=1 protocol
SCARD_PROTOCOL_RAW Use with memory type cards
dwPreferredProtocols is a bit mask of acceptable protocols for the connection. You can use (SCARD_PROTOCOL_T0 | SCARD_PROTOCOL_T1) if you do not have a preferred protocol.

Example:

```
SCARDCONTEXT      hContext;
SCARDHANDLE       hCard;
DWORD             dwActiveProtocol;
LONG              rv;
rv = SCardEstablishContext(SCARD_SCOPE_SYSTEM, NULL, NULL, &hContext);
rv = SCardConnect(hContext, "Reader X", SCARD_SHARE_SHARED,
SCARD_PROTOCOL_T0, &hCard, &dwActiveProtocol);
```

Returns:

SCARD_S_SUCCESS	Successful
SCARD_E_INVALID_HANDLE	Invalid hContext handle
SCARD_E_INVALID_PARAMETER	Invalid parameter
SCARD_E_NO_SMARTCARD	no smart card
SCARD_E_READER_UNAVAILABLE	Could not power up the reader or card
SCARD_E_UNSUPPORTED_FEATURE	Protocol not supported

3.4 SCardStatus

Synopsis:

#include <winscard.h>

LONG SCardStatus(SCARDHANDLE hCard,
    LPSTR mszReaderNames,
    LPDWORD pcchReaderLen,
    LPDWORD pdwState,
    LPDWORD pdwProtocol,
    LPBYTE pbAtr,
    LPDWORD pcbAtrLen);

Parameters:

hCard	IN	Connection made from SCardConnect
mszReaderNames	IN OUT	Friendly name of this reader
pcchReaderLen	IN OUT	Size of the szReaderName multistring
pdwState	OUT	Current state of this reader
pdwProtocol	OUT	Current protocol of this reader
pbAtr	OUT	Current ATR of a card in this reader
pcbAtrLen	OUT	Length of ATR

Description:

This function returns the current status of the reader connected to by hCard. It's friendly name will be stored in mszReaderNames. pcchReaderLen will be the size of the allocated buffer for mszReaderNames, while pcbAtrLen will be the size of the allocated buffer for pbAtr. If either of these is too small, the function will return with SCARD_E_INSUFFICIENT_BUFFER and the necessary size in pcchReaderLen and pcbAtrLen. The current state, and protocol will be stored in pdwState and pdwProtocol respectively. pdwState is a DWORD possibly OR'd with the following values:

Value of pdwState	Meaning
SCARD_ABSENT	There is no card in the reader
SCARD_PRESENT	There is a card in the reader, but it has not been moved into position for use
SCARD_SWALLOWED	There is a card in the reader in position for use. The card is not powered
SCARD_POWERED	Power is being provided to the card, but the reader driver is unaware of the mode of the card
SCARD_NEGOTIABLE	The card has been reset and is awaiting PTS negotiation
SCARD_SPECIFIC	The card has been reset and specific communication protocols have been established

Value of pdwProtocol	Meaning
SCARD_PROTOCOL_T0	Use the T=0 protocol
SCARD_PROTOCOL_T1	Use the T=1 protocol

Example:

```
SCARDCONTEXT    hContext;
SCARDHANDLE     hCard;
```

```
DWORD           dwActiveProtocol;
DWORD           dwState, dwProtocol, dwAtrLen, dwReaderLen;
BYTE            pbAtr[MAX_ATR_SIZE];
rv = SCardEstablishContext(SCARD_SCOPE_SYSTEM, NULL, NULL, &hContext);
rv = SCardConnect(hContext, "Reader X", SCARD_SHARE_SHARED,
SCARD_PROTOCOL_T0, &hCard, &dwActiveProtocol);
dwAtrLen = sizeof(pbAtr);
rv=SCardStatus(hCard, NULL, &dwReaderLen, &dwState, &dwProtocol,pbAtr, &dwAtrLen);
```

Returns:

SCARD_S_SUCCESS	Successful
SCARD_E_INSUFFICIENT_BUFFER	Not enough allocated memory for mszReaderNames or for pbAtr

3.5 SCardGetAttrib

Synopsis:

```
#include <winscard.h>
LONG SCardGetAttrib(SCARDHANDLE hCard,
    DWORD dwAttrId,
    LPBYTE pbAttr,
    LPDWORD pcbAttrLen);
```

Parameters:

hCard	IN	Connection made from SCardConnect
dwAttrId	IN	Identifier for the attribute to get
pbAttr	OUT	Pointer to a buffer that receives the attribute
pcbAttrLen	IN/OUT	Length of the pbAttr buffer in bytes

Description:

This function get an attribute from the IFD Handler. The list of possible attributes is:

- SCARD_ATTR_ATR_STRING

Example:

```
LONG            rv;
SCARDCONTEXT    hContext;
SCARDHANDLE     hCard;
DWORD           dwActiveProtocol;
unsigned char   pbAtr[MAX_ATR_SIZE];
DWORD           dwAtrLen;
rv = SCardEstablishContext(SCARD_SCOPE_SYSTEM, NULL, NULL, &hContext);
rv = SCardConnect(hContext, "Reader X", SCARD_SHARE_SHARED,
SCARD_PROTOCOL_RAW &hCard, &dwActiveProtocol);
rv = SCardGetAttrib(hCard, SCARD_ATTR_ATR_STRING, pbAtr, &dwAtrLen);
```

Returns:

SCARD_S_SUCCESS	Successful
SCARD_E_INVALID_HANDLE	Invalid hCard handle
SCARD_E_INVALID_PARAMETER	Invalid parameter
SCARD_E_INSUFFICIENT_BUFFER	receive buffer not large enough
SCARD_E_NOT_TRANSACTED	Data exchange not successful
SCARD_E_SHARING_VIOLATION	Someone else has exclusive rights
SCARD_E_READER_UNAVAILABLE	The reader has been removed

3.6 SCardTransmit

Synopsis:

```
#include <winscard.h>
LONG SCardTransmit(SCARDHANDLE hCard,
    const SCARD_IO_REQUEST *pioSendPci,
    LPCBYTE pbSendBuffer,
    DWORD cbSendLength,
    SCARD_IO_REQUEST *pioRecvPci,
    LPBYTE pbRecvBuffer,
    LPDWORD pcbRecvLength);
```

Parameters:

hCard	IN	Connection made from SCardConnect
pioSendPci	IN/OUT	Structure of protocol information
pbSendBuffer	IN	APDU to send to the card
cbSendLength	IN	Length of the APDU
pioRecvPci	IN/OUT	Structure of protocol information
pbRecvBuffer	OUT	Response from the card
pcbRecvLength	IN/OUT	Length of the response

Description:

This function sends an APDU to the smart card contained in the reader connected to by SCardConnect(). The card responds from the APDU and stores this response in pbRecvBuffer and it's length in SpcbRecvLength. SSendPci and SRecvPci are structures containing the following:

```
typedef struct {
    DWORD dwProtocol; /* SCARD_PROTOCOL_T0 or SCARD_PROTOCOL_T1 */
    DWORD cbPciLength; /* Length of this structure - not used */
} SCARD_IO_REQUEST;
```

Value of pioSendPci Meaning

SCARD_PCI_T0	Pre-defined T=0 PCI structure
SCARD_PCI_T1	Pre-defined T=1 PCI structure

Example:

```
LONG            rv;
SCARDCONTEXT    hContext;
SCARDHANDLE     hCard;
DWORD           dwActiveProtocol, dwSendLength, dwRecvLength;
SCARD_IO_REQUEST    pioRecvPci;
BYTE    pbRecvBuffer[10];
BYTE    pbSendBuffer[] = { 0xC0, 0xA4, 0x00, 0x00, 0x02, 0x3F, 0x00 };
rv = SCardEstablishContext(SCARD_SCOPE_SYSTEM, NULL, NULL, &hContext);
rv = SCardConnect(hContext, "Reader X", SCARD_SHARE_SHARED,
    SCARD_PROTOCOL_T0, &hCard, &dwActiveProtocol);
dwSendLength = sizeof(pbSendBuffer);
dwRecvLength = sizeof(pbRecvBuffer);
rv = SCardTransmit(hCard, SCARD_PCI_T0, pbSendBuffer, dwSendLength,
&pioRecvPci, pbRecvBuffer, &dwRecvLength);
```

Returns:

SCARD_S_SUCCESS	Successful
SCARD_E_INVALID_HANDLE	Invalid hCard handle
SCARD_E_INSUFFICIENT_BUFFER	receive buffer not large enough
SCARD_E_NOT_TRANSACTED	Data exchange not successful
SCARD_E_INVALID_PARAMETER	invalid parameter
SCARD_E_INVALID_VALUE	Invalid Protocol, reader name, etc

3.7 SCardGetStatusChange

Synopsis:

```
#include <winscard.h>
LONG SCardGetStatusChange(SCARDCONTEXT hContext,
    DWORD dwTimeout,
    LPSCARD_READERSTATE rgReaderStates,
    DWORD cReaders);
```

Parameters:

hContext	IN	Connection context to the PC/SC Resource Manager
dwTimeout	IN	Maximum waiting time (in miliseconds) for status change, zero (or INFINITE) for infinite
rgReaderStates	IN/OUT	Structures of readers with current states
cReaders	IN	Number of structures

Description:

This function blocks execution until the current availability of the cards in a specific set of readers changes.

The caller supplies a list of readers to be monitored through an SCARD_READERSTATE array and the maximum amount of time, in seconds, that it is willing to wait for an action to occur on one of the listed readers. The function returns when there is a change in availability, having filled in the *dwEventState* members of the **SCARD_READERSTATE** structures appropriately.

Example:

```
SCARDCONTEXT            hContext;
SCARD_READERSTATE_A     rgReaderStates[1];
LONG      rv;
rv = SCardEstablishContext(SCARD_SCOPE_SYSTEM, NULL, NULL, &hContext);
rgReaderStates[0].szReader = "Reader X";
rgReaderStates[0].dwCurrentState = SCARD_STATE_UNAWARE;
rv = SCardGetStatusChange(hContext, INFINITE, rgReaderStates, 1);
printf("reader state: 0x%04X\n", rgReaderStates[0].dwEventState);
```

Returns:

SCARD_S_SUCCESS	Successful
SCARD_E_READER_UNAVAILABLE	The reader is unavailable

3.8 SCardDisconnect

Synopsis:

```
#include <winscard.h>
LONG SCardDisconnect(SCARDHANDLE hCard,
      DWORD dwDisposition);
```

Parameters:

hCard	IN	Connection made from SCardConnect
dwDisposition	IN	Reader function to execute

Description:

This function terminates a connection to the connection made through SCardConnect.

dwDisposition can have the following values:

Value of dwDisposition	Meaning
SCARD_LEAVE_CARD	Do nothing
SCARD_RESET_CARD	Reset the card (warm reset)
SCARD_UNPOWER_CARD	Unpower the card (cold reset)
SCARD_EJECT_CARD	Eject the card

Example:

```
SCARDCONTEXT hContext;
```

```
SCARDHANDLE hCard;
DWORD dwActiveProtocol;
LONG rv;
rv = SCardEstablishContext(SCARD_SCOPE_SYSTEM, NULL, NULL, &hContext);
rv = SCardConnect(hContext, "Reader X", SCARD_SHARE_SHARED,
SCARD_PROTOCOL_T0, &hCard, &dwActiveProtocol);
rv = SCardDisconnect(hCard, SCARD_UNPOWER_CARD);
```

Returns:

SCARD_S_SUCCESS	Successful
SCARD_E_INVALID_HANDLE	Invalid hCard handle
SCARD_E_INVALID_VALUE	Invalid dwDisposition

3.9 SCardReleaseContext

Synopsis:

```
#include <winscard.h>
LONG SCardReleaseContext(SCARDCONTEXT hContext);
```

Parameters:

hContext	IN	Connection context to be closed

Description:

This function destroys a communication context to the PC/SC Resource Manager. This must be the last function called in a PC/SC application.

Example:

```
SCARDCONTEXT    hContext;
LONG            rv;
rv = SCardEstablishContext(SCARD_SCOPE_SYSTEM, NULL, NULL, &hContext);
rv = SCardReleaseContext(hContext);
```

Returns:

SCARD_S_SUCCESS	Successful
SCARD_E_INVALID_HANDLE	Invalid hContext handle

4.0 FtGetSerialNum(private interface)

Synopsis:

```
#include <winscard.h>
LONG FtGetSerialNum(unsigned int reader_index,
```

unsigned int length,
char * buffer);

Parameters:

reader_index	IN	reader index
length	IN	length of buffer(>=8)
buffer	OUT	Serial number

Description:

This function userd to get serial number of reader.

Example:

```
SCARDCONTEXT    hContext;
SCARDHANDLE     hCard;
DWORD           dwActiveProtocol;
LONG            rv;
Char            buffer[20] = {0};
rv = SCardEstablishContext(SCARD_SCOPE_SYSTEM, NULL, NULL, &hContext);
rv = SCardConnect(hContext, "Reader X", SCARD_SHARE_SHARED,
SCARD_PROTOCOL_T0, &hCard, &dwActiveProtocol);
rv = FtGetSerialNum(0, sizeof(buffer), buffer);
```

Returns:

SCARD_S_SUCCESS	Successful
SCARD_F_COMM_ERROR	Get serial Num failed
SCARD_E_INVALID_PARAMETER	Invalid parameter

4.1 FtWriteFlash (private interface)

Synopsis:

```
#include <winscard.h>
LONG FtWriteFlash(unsigned int reader_index,
    unsigned char bOffset,
    unsigned char blength,
    unsigned char buffer[]);
```

Parameters:

reader_index	IN	reader index
bOffset	IN	Offset of flash to write
blength	IN	The length of data
buffer	IN	The data for write

Description:

This function userd to write data to flash.

Example:

```
SCARDCONTEXT       hContext;
SCARDHANDLE        hCard;
DWORD              dwActiveProtocol;
LONG               rv;
unsigned char buffer[255] ={0};
rv = SCardEstablishContext(SCARD_SCOPE_SYSTEM, NULL, NULL, &hContext);
rv = SCardConnect(hContext, "Reader X", SCARD_SHARE_SHARED,SCARD_PROTOCOL_T0, &hCard,
&dwActiveProtocol);
for (int i=0; i< 255; i++) {
      buffer[i]= i;
}
rv = FtWriteFlash(0, 0,255, buffer);
```

Returns:

SCARD_S_SUCCESS	Successful
SCARD_F_COMM_ERROR	write data failed
SCARD_E_INVALID_PARAMETER	Invalid parameter

4.2 FtReadFlash(private interface)

Synopsis:

```
#include <winscard.h>
LONG FtReadFlash(unsigned int reader_index,
      unsigned char bOffset,
      unsigned char blength,
      unsigned char buffer[]);
```

Parameters:

reader_index	IN	reader index
bOffset	IN	Offset of flash to write
blength	IN	The length of read data
buffer	OUT	The read data

Description:

This function used to read data from flash.

Example:

```
SCARDCONTEXT       hContext;
SCARDHANDLE        hCard;
DWORD              dwActiveProtocol;
```

```
LONG                rv;
unsigned char buffer[255] ={0};
rv = SCardEstablishContext(SCARD_SCOPE_SYSTEM, NULL, NULL, &hContext);
rv = SCardConnect(hContext, "Reader X", SCARD_SHARE_SHARED, SCARD_PROTOCOL_T0, &hCard,
&dwActiveProtocol);
rv = FtReadFlash (0, 0,255, buffer);
```

Returns:

SCARD_S_SUCCESS	Successful
SCARD_F_COMM_ERROR	write data failed
SCARD_E_INVALID_PARAMETER	Invalid parameter

4.3 FtSetTimeout (private interface)

Synopsis:

```
#include <winscard.h>
LONG FtSetTimeout (SCARDCONTEXT hContext,
      DWORD dwTimeout)
```

Parameters:

SCARDCONTEXT	IN	Connection context to the PC/SC Resource Manager	
DWORD	IN	dwTimeou t (the unit value is "s")	New transmission timeout value of between 301 and card (millisecond)

Description:

This function use to set timeout, the default time out is 6 s.

When you using this function, the dwTimeout value must be higher than 1s.

The function New transmission timeout value of between 301 and card.

Example:

```
SCARDCONTEXT    hContext;
DWORD           dwTimeout;
LONG            rv;
unsigned char buffer[255] ={0};
rv = SCardEstablishContext(SCARD_SCOPE_SYSTEM, NULL, NULL, &hContext);
rv = SCardConnect(hContext, "Reader X", SCARD_SHARE_SHARED, SCARD_PROTOCOL_T0, &hCard,
&dwActiveProtocol);
rv = FtSetTimeout (hContext, 6);
```

Returns:

SCARD_S_SUCCESS	Successful
SCARD_F_COMM_ERROR	write data failed
SCARD_E_INVALID_PARAMETER	Invalid parameter

4.4 FtDukptInit (private interface)

Synopsis:

#include <winscard.h>

LONG FtDukptInit(SCARDHANDLE hCard,
unsigned char *encBuf,
unsigned int nLen);

Parameters:

hCard IN Connection made from SCardConnect(Ignore this parameter and just set to zero in iOS system)

encBuf IN Ciphertext use TDES_ECB_PKCS7/AES_ECB_PKCS7 (See "Key C"), the data is IPEK+KSN+0XFFFF+CRC

nLen IN encBuf length(40(TDES_ECB_PKCS7 ciphertext length) 48(AES_ECB_PKCS7 ciphertext length))

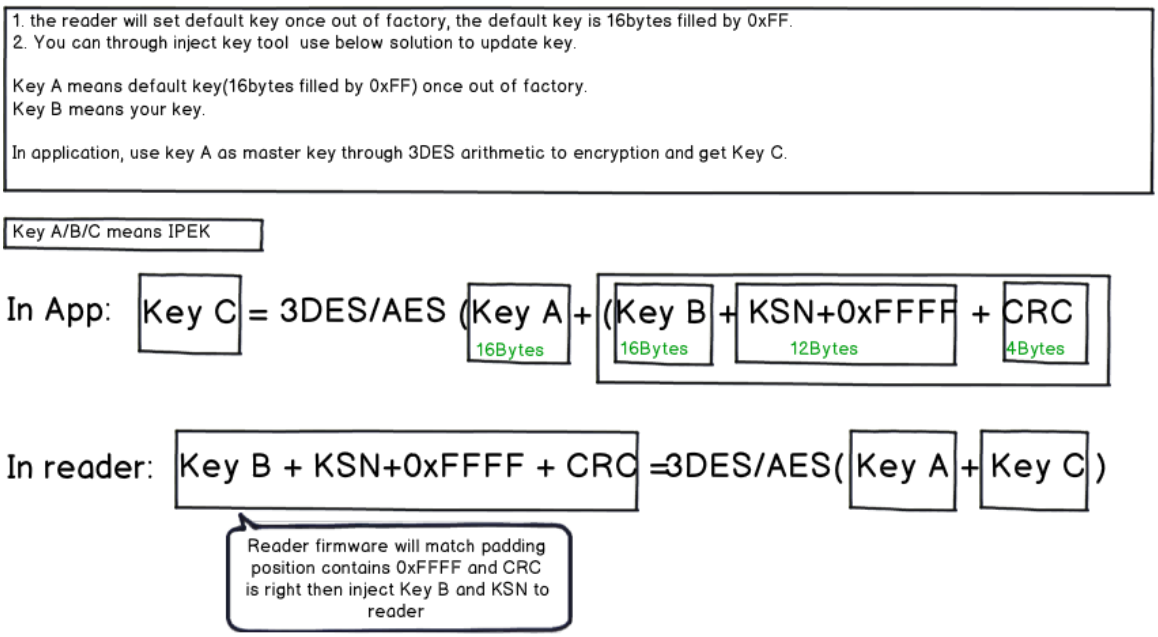
Description:

Init iR301 new ipek and ksn for dukpt.

The IPEK is based on computer between KSN (Key Serial Number) and BDK(Base Derivation Key). More information, please follow below documents.

Section DUKPT http://download.ftsafe.com/files/reader/ANSIX9.24PART1-2004.pdf

Here is a solution of init DUKPT:



Example:

Please refer in FtDukptGetKSN example

4.5 FtDukptSetEncMod (private interface)

Synopsis:

#include <winscard.h>
LONG FtDukptSetEncMod(SCARDHANDLE hCard,unsigned int bEncrypt,unsigned int bEncFunc,

unsigned int bEncType)

Parameters:

hCard [IN] Connection made from SCardConnect(Ignore this parameter and just set to zero in iOS system)

bEncrypt [IN] 0: Non- encrypted communication 1: Encrypted communication

bEncFunc [IN] 0:using 3DES to encryption 1:using AES to encryption

bEncType [IN] 0:Two ways to encryption(send and return data) 1: one-way encryption(only return data encrypted

Description:

To set DUKPT's encryption mode

Ntice:Only can set this mode when power off from card.,

Example:

Please refer in FtDukptGetKSN example

4.6 FtDukptGetKSN(private interface)

#include <winscard.h>

LONG FtDukptGetKSN(SCARDHANDLE hCard, unsigned int * pnlength,unsigned char *buffer)

Parameters:

hCard [IN] Connection made from SCardConnect(Ignore this parameter and just set to zero in iOS system)

pnlength [IN/OUT] KSN's length

buffer [IN/OUT] KSN

Description:

To get reader's KSN ,

Example:

```
    void derive_IPEK(BYTE *bdk, BYTE *ksn, BYTE *ipek);

    void derive_PEK(BYTE *ipek, BYTE *ksn, BYTE *pek);

    int  TDES_ECB_PKCS7(unsigned char *inBuf, unsigned int inLen, unsigned char *outBuf,
unsigned  int *outLen, unsigned char *key,unsigned  int keyLen, int mode);

    int  AES_ECB_PKCS7(unsigned char *inBuf, unsigned int inLen, unsigned char *outBuf,
unsigned  int *outLen, unsigned char *key,unsigned  int keyLen, int mode);


    #define MODE_ENCRYPT               1

    #define MODE_DECRYPT               0


    static unsigned int crc_table[256];


    static void init_crc_table(void);

    static unsigned int crc32(unsigned int crc, unsigned char * buffer, unsigned int size);


    /*init crc table*/

    static void init_crc_table(void)

    {

        unsigned int c;

        unsigned int i, j;

        for (i = 0; i < 256; i++) {

            c = (unsigned int)i;

            for (j = 0; j < 8; j++) {

                if (c & 1)

                    c = 0xedb88320L ^ (c >> 1);

                else

                    c = c >> 1;

            }
```

```
            crc_table[i] = c;
        }
    }
    /*Calculate crc*/
    static unsigned int crc32(unsigned int crc,unsigned char *buffer, unsigned int size)
    {
        unsigned int i;
        for (i = 0; i < size; i++) {
            crc = crc_table[(crc ^ buffer[i]) & 0xff] ^ (crc >> 8);
        }
        return crc ;
    }
    static void i2dw(int value, unsigned char buffer[])
    {
        buffer[0] = value & 0xFF;
        buffer[1] = (value >> 8) & 0xFF;
        buffer[2] = (value >> 16) & 0xFF;
        buffer[3] = (value >> 24) & 0xFF;
    } /* i2dw */

    unsigned char BDKBuffer[64]="\x01\x23\x45\x67\x89\xAB\xCD\xEF
    \xFE\xDC\xBA\x98\x76\x54\x32\x10";
    unsigned char KSNBuffer[64]="\xff\xff\x98\x76\x54\x32\x10\xe0\x00\x00";
    unsigned int KSNLength=0;
    unsigned char IPEK[16]={0};
    unsigned char PEK[16]={0};
    unsigned                                                    char
OLDIPEK[16]="\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff
    \xff\xff";
    unsigned char encBuffer[64]={0};
```

```
unsigned char unencBuffer[64]={0};
unsigned int encLength=0;
unsigned char SendBuffer[5]="\x00\x84\x00\x00\x08";
unsigned int SendLength = sizeof(SendBuffer);
unsigned char ReBuffer[512]={0};
unsigned int ReLength = sizeof(ReBuffer);
unsigned int CRC=0xffffffff;
LONG ReturnValue=0;

BOOL encType=FALSE;
BOOL encFunc=TRUE;
BOOL encCrypt=TRUE;

if(16!=strlen(BDKBuffer)){
    printf("BDK data's format is error.\n");
}
if(10!=strlen(KSNBuffer)){
    printf("KSN data's format is error.\n");
}
derive_IPEK(BDKBuffer, KSNBuffer, IPEK);
KSNBuffer[7]= KSNBuffer[7]&0xE0;
KSNBuffer[8]=0x00;
KSNBuffer[9]=0x00;
memcpy(unencBuffer,IPEK,16);
memcpy(unencBuffer+16,KSNBuffer,10);
unencBuffer[26]=0xFF;
unencBuffer[27]=0xFF;

init_crc_table();
```

```
CRC=crc32(CRC, unencBuffer,28);
i2dw(CRC, unencBuffer+28);

//Thde default encryption mode is 3DES
TDES_ECB_PKCS7(unencBuffer,32, encBuffer, &encLength, OLDIPEK,16, MODE_ENCRYPT);
ReturnValue=FtDukptInit(0, encBuffer, encLength);
if(ReturnValue!=0){
    printf("FtDukptInit error %08x\n", ReturnValue);
}
ReturnValue =FtDukptSetEncMode(0, encrypt, encFunc, encType);
if(ReturnValue!=0){
    printf("FtDukptSetEncMode error %08x\n", ReturnValue);
}
if(encrypt&& encType==FALSE){
    memset(KSNBuffer,0,sizeof(KSNBuffer));
memset(IPEK,0,sizeof(IPEK));
    ReturnValue =FtDukptGetSN(0,&KSNLength,KSNBuffer)
    if(ReturnValue!=0){
        printf("FtDukptGetSN error %08x\n", ReturnValue);
}
    derive_IPEK(BDKBuffer,KSNBuffer,IPEK);
derive_PEK(IPEK, KSNBuffer,PEK);
if(encFunc){
        AES_ECB_PKCS7(SendBuffer,5, ReBuffer, &ReLength, PEK,16, MODE_ENCRYPT);
}else{
        TDES_ECB_PKCS7(SendBuffer,5, ReBuffer,& ReLength,PEK,16, MODE_ENCRYPT);
}
Memcpy(SendBuffer, ReBuffer, ReLength);
}
```

```
    SCARD_IO_REQUEST pioSendPci
    ReturnValue=SCardTransmit(gCardHandle,&pioSendPci,SendBuffer,ReLength,NULL,ReBuf
fer,
    &ReLength);
    if(ReturnValue != SCARD_S_SUCCESS){
        printf("SCardTransmit error %08x\n", ReturnValue);
    }else{
        if(encrypt && ReLength >10){
            memset(IPEK,0,sizeof(IPEK));
            memset(PEK,0,sizeof(PEK));
            memset(KSNBuffer,0,sizeof(KSNBuffer));
            memcpy(KSNBuffer,ReBuffer+ReLength-10,10);
            derive_IPEK(BDKBuffer,KSNBuffer,IPEK);
    derive_PEK(IPEK, KSNBuffer,PEK);
    if(encFunc){
            AES_ECB_PKCS7(ReBuffer,    ReLength,    SendBuffer,    &ReLength,    PEK,16,
MODE_DECRYPT);
    }else{
            TDES_ECB_PKCS7(ReBuffer,    ReLength,    SendBuffer,&    ReLength,PEK,16,
MODE_DECRYPT);
    }
        }
        memcpy(ReBuffer, SendBuffer, ReLength);
    }
```

4.7 FtDidEnterBackground(private interface)

Void FtDidEnterBackground(unsigned int bDidEnter)

Description:

Enter application to background, won't close session

Parameter:

bDidEnter [IN] True (when you set to 1, it will support background running)

4.8 ReaderInterfaceDelegate(private interface)

4.8.1 readerInterfaceDidChange(private interface)

-(void)readerInterfaceDidChange:(BOOL)attached

Description:

To monitor the reader status's delegation

Parameter:

attached [OUT] TRUE means the reader has inserted， FALSE means the reader has plug-out

Example: Please refer in isCardAttached

4.8.1 cardInterfaceDidDetach(private interface)

-(void)cardInterfaceDidDetach:(BOOL)attached

Describe:

The delegation is to monitor the card status

Parameter:

Attached [OUT] Ture means the card has inserted, false means the card has removed out

Example:

Please refer in isCardAttached

4.9 @interface ReaderInterface(private interface)

4.9.1 setDelegate(private interface)

-(void) setDelegate:(id<ReaderInterfaceDelegate>)delegate;

Description:

Set delegate:

Example:

Please refer in isCardAttached

4.9.2 isReaderAttached(private interface)

-(BOOL) isReaderAttached

Descryption:

To get reader status (absent or attach)

Return:

The ture means the reader has inserted, false means reader plug out

Example: Please refer in isCardAttached

4.9.3 isCardAttached(private interface)

-(BOOL) isCardAttached

Descryption:

To check the card slot status

Return:

The ture means the card inserted, the false means the card plug out

Example

```
@interface mainViewController:UIViewController<ReaderInterfaceDelegate>
{
    ReaderInterface *reader;
}
@end

@implementation mainViewController
- (id)initWithNibName:(NSString *)nibNameOrNil bundle:(NSBundle *)nibBundleOrNil
{
    self = [super initWithNibName:nibNameOrNil bundle:nibBundleOrNil];
    if (self) {
        // Custom initialization
```

```
    }
    return self;
}

- (void)viewDidLoad
{
    /*
     *To check the card status ReaderInterfaceDelegate
     */
SCARDCONTEXT cardContext;
    reader =[[ReaderInterface alloc] init];
[reader setDelegate:self];
SCardEstablishContext(SCARD_SCOPE_SYSTEM, NULL, NULL,
                              &cardContext);
[self test];
    [super viewDidLoad];
    // Do any additional setup after loading the view from its nib.
}

-(void)test
{
    BOOL ReaderStatus;
    BOOL    CardStatus;
    ReaderStatus = isReaderAttached;
if (ReaderStatus) {
        NSLog(@"\nreader is attached>>>");
    }else{
        NSLog(@"\nreader is disattached>>>");
    }
```

```
    CardStatus = isCardAttached;
if (CardStatus) {
        NSLog(@"\ncard is attached>>>");
    }else{
        NSLog(@"\ncard is disattached>>>");
    }
}

- (void)didReceiveMemoryWarning
{
    [super didReceiveMemoryWarning];
    // Dispose of any resources that can be recreated.
}

#pragma ReaderInterfaceDelegate
-(void)readerInterfaceDidChange:(BOOL)attached
{
    if (attached) {
        NSLog(@"\nreader is attached>>>");
    }else{
        NSLog(@"\nreader is disattached>>>");
    }
}
-(void)cardInterfaceDidDetach:(BOOL)attached
{
    if (attached) {
        NSLog(@"\ncard is attached>>>");
    }else{
        NSLog(@"\ncard is disattached>>>");
```

```
        }
    }
```

5.0 Support background mode

If you application support background mode, please do below operation:

1. Add background support string in your Info.plist

```
<key>UIBackgroundModes</key>
    <array>
        <string>App communicates with an accessory</string>
    </array>
```

2. When you enter to background, please do release context.

```
extern  SCARDCONTEXT gContxtHandle;
    //If your application support background running, please do release context when enter to background
    FtDidEnterBackground(1);
    SCardReleaseContext(gContxtHandle);

       }
```

3. Also when enter to foreground, please do create context also.

```
- (void)applicationDidBecomeActive:(UIApplication *)application {
    /*
     Restart any tasks that were paused (or not yet started) while the application was inactive. If the application was previously in the background, optionally refresh the user interface.
     */
    //When back to foreground, do create context
      SCardEstablishContext(SCARD_SCOPE_SYSTEM,NULL,NULL,&gContxtHandle);

}
```

5.1 Support bR301 also

If you application also want to support iR301 and bR301, please do below small change on your project.