DYNAMIC ENGINEERING
435 Park Dr., Ben Lomond, Calif. 95005
831-336-8891 Fax 831-336-3840
http://www.dyneng.com
sales@dyneng.com
Est. 1988

IpCrypto

Driver Documentation

Win32 Driver Model

Revision A
Corresponding Hardware: Revision B
10-2001-0302
Corresponding Firmware: Revision E

IpCrypto
WDM Device Driver for the
IP-Crypto IP Module

Dynamic Engineering
435 Park Drive
Ben Lomond, CA 95005
831- 336-8891
831-336-3840 FAX

Manual Revision A. Revised February 17, 2004.

This document contains information of proprietary interest to Dynamic Engineering. It has been supplied in confidence and the recipient, by accepting this material, agrees that the subject matter will not be copied or reproduced, in whole or in part, nor its contents revealed in any manner or to any person except to meet the purpose for which it was delivered.

Dynamic Engineering has made every effort to ensure that this manual is accurate and complete. Still, the company reserves the right to make improvements or changes in the product described in this document at any time and without notice. Furthermore, Dynamic Engineering assumes no liability arising out of the application or use of the device described herein.

The electronic equipment described herein generates, uses, and can radiate radio frequency energy. Operation of this equipment in a residential area is likely to cause radio interference, in which case the user, at his own expense, will be required to take whatever measures may be required to correct the interference.

Dynamic Engineering's products are not authorized for use as critical components in life support devices or systems without the express written approval of the president of Dynamic Engineering.

This product has been designed to operate with IP Module carriers and compatible user-provided equipment. Connection of incompatible hardware is likely to cause serious damage.

Table of Contents

Introduction

The IpCrypto driver is a Win32 driver model(WDM) device driver for the IP-Crypto board from Dynamic Engineering. Each IP-Crypto board implements an interface to a KYK-13 communications security device. A separate Device Object controls each IP-Crypto board, and a separate handle references each Device Object. IO Control calls (IOCTLs) are used to configure the board and to transfer data to and from the IP device.

Note

This documentation will provide information about all calls made to the driver, and how the driver interacts with the hardware for each of these calls. For more detailed information on the hardware implementation, refer to the IP-Crypto user manual (also referred to as the hardware manual).

Driver Installation

Warning: The appropriate IP carrier driver must be installed before any IP modules can be detected by the system.

There are several files provided in each driver package. These files include IpCrypto.sys, IpDevice.inf, DDIpCrypto.h, IpCryptoGUID.h, IpCryptoDef.h, IPCTest.exe, and IPCTest source files.

Windows 2000 Installation

Copy IpDevice.inf and IpCrypto.sys to a floppy disk, or CD if preferred.

With the hardware installed, power-on the PCI host computer and wait for the *Found New Hardware Wizard* dialogue window to appear.

- Select *Next*.
- Select *Search for a suitable driver for my device.*
- Select *Next*.
- Insert the disk prepared above in the desired drive.
- Select the appropriate drive e.g. *Floppy disk drives.*
- Select *Next*.
- The wizard should find the IpDevice.inf file.
- Select *Next*.
- Select *Finish* to close the *Found New Hardware Wizard*.

Windows XP Installation

Copy IpDevice.inf to the WINDOWS\INF folder and copy IpCrypto.sys to a floppy disk, or CD if preferred. Right click on the IpDevice.inf file icon in the WINDOWS\INF folder and select *Install* from the pop-up menu. This will create a precompiled information file (.pnf) in the same directory.

With the hardware installed, power-on the PCI host computer and wait for the *Found New Hardware Wizard* dialogue window to appear. The IP-Crypto should be named in the dialogue box. Follow the steps below:

- Insert the disk prepared above in the appropriate drive.
- Select *Install from a list or specific location*
- Select *Next*
- Select *Don't search. I will choose the driver to install*
- Select *Next*
- Select *Show all devices* from the list
- Select *Next*
- Select *Dynamic Engineering* from the Manufacturer list
- Select *IP-Crypto Device* from the Model list
- Select *Next*
- Select *Yes* on the Update Driver Warning dialogue box.
- Enter the drive *e.g. A:*| in the *Files Needed* dialogue box.
- Select *OK*.
- Select *Finish* to close the *Found New Hardware Wizard*.

This process must be completed for each new device that is installed.

The DDIpCrypto.h file is the C header file that defines the Application Program Interface (API) to the driver. The IpCryptoGUID.h file is a C header file that defines the device interface identifier for the IpCrypto. These files are required at compile time by any application that wishes to interface with the IpCrypto driver. The IpCryptoDef.h file contains the relevant bit defines for the IP-Crypto registers. These files are not needed for driver installation.

The IPCTest.exe file is a sample Win32 console application that makes calls into the IpCrypto driver to test the driver calls without actually writing an application. It is not required during the driver installation. Open a command prompt console window and type *IPCTest –d0 -?* to display a list of commands (the IPCTest.exe file must be in the directory that the window is referencing). The commands are all of the form *IPCTest –dn –im* where *n* and *m* are the device number and driver ioctl number respectively. This application is intended to test the proper functioning of the driver calls, not for normal hardware operation.

Driver Startup

Once the driver has been installed it will start automatically when the system recognizes the hardware.

A handle can be opened to a specific board by using the CreateFile() function call and passing in the device name obtained from the system.

The interface to the device is identified using a globally unique identifier (GUID), which is defined in IpCryptoGUID.h.

Below is example code for opening a handle for device 0. The device number is underlined and italicized in the `SetupDiEnumDeviceInterfaces` call.

```
// The maximum length of the device name for
//  a given instance of an interface
#define MAX_DEVICE_NAME 256
// Handle to the device object
HANDLE                           hIpCrypto = INVALID_HANDLE_VALUE;
// Return status from command
LONG                             status;
// Handle to device interface information structure
HDEVINFO                         hDeviceInfo;
// The actual symbolic link name to use in the createfile
CHAR                             deviceName[MAX_DEVICE_NAME];
// Size of buffer required to get the symbolic link name
DWORD                            requiredSize;
// Interface data structures for this device
SP_DEVICE_INTERFACE_DATA         interfaceData;
PSP_DEVICE_INTERFACE_DETAIL_DATA pDeviceDetail;

hDeviceInfo = SetupDiGetClassDevs((LPGUID)&GUID_DEVINTERFACE_IPCRYPTO,
                                  NULL,
                                  NULL,
                                  DIGCF_PRESENT | DIGCF_DEVICEINTERFACE);

if(hDeviceInfo == INVALID_HANDLE_VALUE)
{
   printf("**Error: couldn't get class info, (%d)\n",
          GetLastError());
   exit(-1);
}
```

```
interfaceData.cbSize = sizeof(interfaceData);

// Find the interface for device 0
if(!SetupDiEnumDeviceInterfaces(hDeviceInfo,
                                NULL,
                                (LPGUID)&GUID_DEVINTERFACE_IPCRYPTO,
                                0,
                                &interfaceData))
{
   status = GetLastError();
   if(status == ERROR_NO_MORE_ITEMS)
   {
      printf("**Error: couldn't find device(no more items), (%d)\n", 0);
      SetupDiDestroyDeviceInfoList(hDeviceInfo);
      exit(-1);
   }
   else
   {
      printf("**Error: couldn't enum device, (%d)\n",
             status);
      SetupDiDestroyDeviceInfoList(hDeviceInfo);
      exit(-1);
   }
}

// Get the details data to obtain the symbolic link name
if(!SetupDiGetDeviceInterfaceDetail(hDeviceInfo,
                                    &interfaceData,
                                    NULL,
                                    0,
                                    &requiredSize,
                                    NULL))
{
   if(GetLastError() != ERROR_INSUFFICIENT_BUFFER)
   {
      printf("**Error: couldn't get interface detail, (%d)\n",
             GetLastError());
      SetupDiDestroyDeviceInfoList(hDeviceInfo);
      exit(-1);
   }
}

// Allocate a buffer to get detail
pDeviceDetail = (PSP_DEVICE_INTERFACE_DETAIL_DATA)malloc(requiredSize);
if(pDeviceDetail == NULL)
{
   printf("**Error: couldn't allocate interface detail\n");
   SetupDiDestroyDeviceInfoList(hDeviceInfo);
   exit(-1);
}

pDeviceDetail->cbSize = sizeof(SP_DEVICE_INTERFACE_DETAIL_DATA);

// Get the detail info
if(!SetupDiGetDeviceInterfaceDetail(hDeviceInfo,
                                    &interfaceData,
                                    pDeviceDetail,
                                    requiredSize,
                                    NULL,
                                    NULL))
{
   printf("**Error: couldn't get interface detail(2), (%d)\n",
          GetLastError());
   SetupDiDestroyDeviceInfoList(hDeviceInfo);
   free(pDeviceDetail);
   exit(-1);
}

// Save the name
lstrcpyn(deviceName,
         pDeviceDetail->DevicePath,
         MAX_DEVICE_NAME);

// Cleanup search
free(pDeviceDetail);
SetupDiDestroyDeviceInfoList(hDeviceInfo);

// Open driver
// Create the handle to the device
hIpCrypto = CreateFile(deviceName,
                      GENERIC_READ    | GENERIC_WRITE,
                      FILE_SHARE_READ | FILE_SHARE_WRITE,
                      NULL,
                      OPEN_EXISTING,
                      NULL,
                      NULL);

if(hIpCrypto == INVALID_HANDLE_VALUE)
{
   printf("**Error: couldn't open %s, (%d)\n", deviceName,
          GetLastError());
   exit(-1);
}
```

IO Controls

The driver uses IO Control calls (IOCTLs) to configure the device and pass data in and out. IOCTLs refer to a single Device Object in the driver, which controls a single board. IOCTLs are called using the Win32 function DeviceIoControl(), and passing in the handle to the device opened with CreateFile(). IOCTLs generally have input parameters, output parameters, or both. Often a custom structure is used.

IOCTL_IPCRYPTO_GET_INFO

Function: Returns the current driver version.
Input: none
Output: DRIVER_IP_DEVICE_INFO structure
Notes: This call does not access the hardware, only driver parameters. See DDIpCrypto.h for the definition of DEVICE_IP_DEVICE_INFO.

IOCTL_IPCRYPTO_SET_IP_CONTROL

Function: Sets the configuration of the IP slot.
Input: ULONG
Output: none
Notes: Controls the IP clock speed and interrupt enables for the IP slot that the board occupies. See the bit definitions in the IpCryptoDef.h header file for more information.

IOCTL_IPCRYPTO_GET_IP_CONTROL

Function: Returns the configuration of the IP slot.
Input: none
Output: ULONG
Notes: Returns the slot configuration register value. See the bit definitions in the IpCryptoDef.h header file for more information.

IOCTL_IPCRYPTO_SET_INTERRUPT_ENABLES

Function: Selects which conditions will cause an interrupt.
Input: INTERRUPT_ENABLES structure
Output: none
Notes: The INTERRUPT_ENABLES structure has three BOOLEAN fields: ClearKey enables an interrupt to occur when the clear key process is performed; StateMachine enables an interrupt to occur when all eight words have been received from a transfer request; and InputData enables the input data interrupt condition specified in the SET_INT_EN, SET_EDGE_LEVEL, and SET_POLARITY calls.

IOCTL_IPCRYPTO_GET_INTERRUPT_ENABLES

Function: Returns the interrupt enable conditions.
Input: none
Output: INTERRUPT_ENABLES structure
Notes:

IOCTL_IPCRYPTO_ENABLE_DATA_OUTPUT

Function: Enables updating the parallel data output.
Input: none
Output: none
Notes: This call enables the final output latch for the output data lines, causing them to be driven off the board.

IOCTL_IPCRYPTO_DISABLE_DATA_OUTPUT

Function: Disables updating the parallel data output.
Input: none
Output: none
Notes:

IOCTL_IPCRYPTO_INIT_XFER

Function: Initiate a KYK-13 transfer request.
Input: none
Output: none
Notes: Starts the state machine to initiate a transfer request and load eight 16-bit serial words.

IOCTL_IPCRYPTO_CLEAR_KEY

Function: Clear the stored key data pattern.
Input: none
Output: none
Notes: The received key values are successively written over with three values that were previously stored in the three clear pattern registers.

IOCTL_IPCRYPTO_ABORT_SM

Function: Abort the current state machine operation.
Input: none
Output: none
Notes:

IOCTL_IPCRYPTO_READ_KYK

Function: Returns a 16-bit segment of the key received.
Input: none
Output: none
Notes:

IOCTL_IPCRYPTO_SET_CLEAR_DATA

Function: Writes clear data patterns to the three clear registers.
Input: CLEAR_PATTERNS structure
Output: none
Notes: The CLEAR_PATTERNS structure has three 16-bit fields that contain the three data patterns used in the clear key process.

IOCTL_IPCRYPTO_GET_CLEAR_DATA

Function: Returns the contents of the three clear registers.
Input: none
Output: CLEAR_PATTERNS structure
Notes:

IOCTL_IPCRYPTO_SET_OUT_DATA

Function: Writes values to the two output data registers.
Input: OUTPUT_DATA structure
Output: none
Notes: Only output data lines 1..23 are controlled by this call. Output data line 0 is used for transfer request from the crypto state machine to the KYK-13. The OUTPUT_DATA structure has an additional BOOLEAN field called AutoSync. If this is set to TRUE, the updating of the output lines is disabled until both data registers are written and then re-enabled to propagate all bits at the same time. The state of the output enable bit remains the same as it was before this call was made.

IOCTL_IPCRYPTO_GET_OUT_DATA

Function: Returns the combined value from the output data registers.
Input: none
Output: ULONG
Notes: This call returns all 24 output data bits even though bit 0 is never driven off the board.

IOCTL_IPCRYPTO_SET_INT_EN

Function: Writes values to the two interrupt enable registers.
Input: ULONG
Output: none
Notes: This call defines the mask of the 24 input lines that will be enabled to cause an interrupt when the specified conditions are met (1 = enabled, 0 = disabled).

IOCTL_IPCRYPTO_GET_INT_EN

Function: Returns the combined value from the interrupt enable registers.
Input: none
Output: ULONG
Notes:

IOCTL_IPCRYPTO_SET_EDGE_LEVEL

Function: Writes values to the two edge/level registers.
Input: ULONG
Output: none
Notes: Determines whether the interrupt for each of the input lines will respond to a static logic level or a transition between levels (1 = edge, 0 = level).

IOCTL_IPCRYPTO_GET_EDGE_LEVEL

Function: Returns the combined value from the edge/level registers.
Input: none
Output: ULONG
Notes:

IOCTL_IPCRYPTO_SET_POLARITY

Function: Writes values to the two polarity registers.
Input: ULONG
Output: none
Notes: Determines the polarity of the level or edge to which the interrupt for each of the input lines will respond (1 = inverted, 0 = non-inverted).

IOCTL_IPCRYPTO_GET_POLARITY

Function: Returns the combined value from the polarity registers.
Input: none
Output: ULONG
Notes:

IOCTL_IPCRYPTO_READ_DIRECT

Function: Reads the direct input data.
Input: none
Output: ULONG
Notes: This call reads the raw real-time input data from the 24 input lines and returns the combined value.

IOCTL_IPCRYPTO_READ_FILTERED

Function: Reads the filtered input data registers.
Input: none
Output: ULONG
Notes: This call reads the contents of the interrupt latches after the enable mask, edge/level, and polarity bits have been applied. A one means that the specified conditions for that bit have been met. Reading these registers clears the latched bits.

IOCTL_IPCRYPTO_GET_INT_STAT

Function: Returns the status bits in the INT_STAT register and clears the latched bits.
Input: none
Output: USHORT
Notes: See the INTSTAT_ defines in IpCryptoDef.h for more information on specific bit values.

IOCTL_IPCRYPTO_REGISTER_EVENT

Function: Registers an event to be signaled when an interrupt occurs.
Input: Handle to Event object
Output: none
Notes: The caller creates an event with CreateEvent() and supplies the handle returned from that call as the input to this IOCTL. The driver then obtains a system pointer to the event and signals the event when an interrupt is serviced. The user interrupt service routine waits on this event, allowing it to respond to the interrupt. In order to unregister the event, set the event handle to NULL while making this call.

IOCTL_IPCRYPTO_ENABLE_INTERRUPT

Function: Enables the master interrupt.
Input: none
Output: none
Notes: Sets the IP slot control register interrupt 0 enable bit. This IOCTL is used in the user interrupt processing function to re-enable the interrupts after they were disabled in the driver interrupt service routine. This allows that function to enable the interrupts without knowing the particulars of the other configuration bits.

IOCTL_IPCRYPTO_DISABLE_INTERRUPT

Function: Disables the master interrupt.
Input: none
Output: none
Notes: Clears the IP slot control register interrupt 0 enable bit, leaving all other bit values in that register unchanged. This IOCTL is used when interrupt processing is no longer desired.

IOCTL_IPCRYPTO_FORCE_INTERRUPT

Function: Causes a system interrupt to occur.
Input: none
Output: none
Notes: Causes an interrupt to be asserted on the IP bus if the master interrupt is enabled. This IOCTL is used for development, to test interrupt processing.

IOCTL_IPCRYPTO_SET_VECTOR

Function: Sets the value of the interrupt vector.
Input: UCHAR
Output: none
Notes: This value is driven onto the low byte of the data bus in response to an INT_SEL strobe, which is used in vectored interrupt cycles. This value will be read in the interrupt service routine and stored for future reference.

IOCTL_IPCRYPTO_GET_VECTOR

Function: Returns the current interrupt vector value.
Input: none
Output: UCHAR
Notes:

IOCTL_IPCRYPTO_GET_ISR_STATUS

Function: Returns the interrupt status and vector read in the last ISR.
Input: none
Output: INT_STAT structure
Notes: The status contains the contents of the INT_STAT register read in the ISR. Also if bit 12 is set, it indicates that a bus error occurred for this IP slot. See IpCryptoDef.h for more information on specific bit values.

Warranty and Repair

Dynamic Engineering warrants this product to be free from defects under normal use and service and in its original, unmodified condition, for a period of one year from the time of purchase. If the product is found to be defective within the terms of this warranty, Dynamic Engineering's sole responsibility shall be to repair, or at Dynamic Engineering's sole option to replace, the defective product.

Dynamic Engineering's warranty of and liability for defective products is limited to that set forth herein. Dynamic Engineering disclaims and excludes all other product warranties and product liability, expressed or implied, including but not limited to any implied warranties of merchandisability or fitness for a particular purpose or use, liability for negligence in manufacture or shipment of product, liability for injury to persons or property, or for any incidental or consequential damages.

Dynamic Engineering's products are not authorized for use as critical components in life support devices or systems without the express written approval of the president of Dynamic Engineering.

Service Policy

Before returning a product for repair, verify as well as possible that the driver is at fault. The driver has gone through extensive testing and in most cases it will be "cockpit error" rather than an error with the driver. When you are sure or at least willing to pay to have someone help then call the Customer Service Department and arrange to speak with an engineer. We will work with you to determine the cause of the issue. If the issue is one of a defective driver we will correct the problem and provide an updated module(s) to you [no cost]. If the issue is of the customer's making [anything that is not the driver] the engineering time will be invoiced to the customer. Pre-approval may be required in some cases depending on the customer's invoicing policy.

Out of Warranty Repairs

Out of warranty support will be billed. The current minimum repair charge is $125. An open PO will be required.

For Service Contact:

Customer Service Department
Dynamic Engineering
435 Park Dr.
Ben Lomond, CA 95005
831-336-8891
831-336-3840 fax
support@dyneng.com