p1b1i47s0-ESTARFAQ.TXT.pdf

EXHIBIT
786

The EchoStar hacking FAQ by StuntGuy
Revision: 10151999

Contents:

CASE NO.
SA CV 03-950 DOC (JTLx)
ECHOSTAR SATELLITE CORP., et al.

vs.

NDS GROUP PLC, et al.

DEFENDANT'S EXHIBIT 786

DATE ______________ IDEN.

DATE ______________ EVID.

BY ______________________
Deputy Clerk

EXHIBIT
176

The EchoStar hacking FAQ by StuntGuy
Revision: 10151999

Contents:

0.1- Introduction/About me:

Okay, the first thing I'd like to say is that I'm not the one who figured all of this stuff out. While I did figure _some_ of it out, there's lots of information here that came from other sources.. their names will be included in section 0.3: Contributors, as will the names of others who have aided the cause. There will certainly by those who've provided information that will be presented here, and when it occurs to me (or when it's pointed out to me) that I've included information that came from an uncredited source, I'll certainly update the FAQ to include credit where credit is due.

Secondly, I'd just like to point out that the reason I'm doing this is just to get all of the information that's been scattered around in one place, and hopefully, to provide a target location for future findings. I'm not into hacking to deprive anyone of their due revenues (in fact, I pay for my E* subscription).. I'm in it for the entertainment. If the E* guys can write code that'll keep me and other hackers out of their cards, then I'm okay with that...but they'll have to do better than 512 bytes of data tables that do nothing but keep them from having to change an ROR into an ROL and count the number of '1' bits they're sending. Gimme a break, guys.

0.2- Where to find this FAQ:

The official release of this FAQ can always be found at

http://www.dishplex.com/eromcentral.shtml

0.3- Contributors:

The following is a list (in no particular order) of people who I think have made a significant contribution to the cause of EchoStar hacking in one form or another. Note that some of these guys aren't around anymore. One in particular was so discouraged by the bickering a technical discussion board that he left and hasn't been back since.

Swiss Cheese Productions/Mr. Bean/NIPPER: The source of the first bit of technical EchoStar info I found...got me on the road. Thanks, man.

Biatch. Keeps on pluggin' away at it. Nice disassembler. Worked great for XFILE

Stymie: PPVs in the enabler. Need I say more.

The_Crack: For the help with the FAQ. And I really appreciate someone who doesn't flame me when I do really stupid things.

xchi and pals: You know what you provided here.

The E_r0m guys: For providing a good environment in which to work, good information, and good sounding boards

Code. Though you haven't been quite as up-front about it as possible, you've been a good source of info, too. Thanks.

0.4- Detractors:

This is a message to those of you out there who can't seem to find anything better to do with your time than whine and flame those of us who're actually attempting to make some progress here. I'm not going to name names, but you all know who you are. The message is this: Go away. When it comes right down to it, there's only two reasons for you to be behaving the way you are:

1. You're working for EchoStar and you just want to create dissent among those of us who're working on hacking your system. If this is the case, your time would be much better spent working on tighter security And cleaner code. 16K of ROM space for this? A real programmer could do it in 8.

2. You're an asshole who thinks the world owes you something for nothing. If this is the case, then you're in for some serious disappointment. If you really do want to get some programming out of all of this, then just shut the hell up and let the rest of us do what we're good at

==

1.1- The EchoStar ATR

The EchoStar cards use a variant of the ISO-7816 protocol called the "T=1" or "asynchronous half-duplex block transfer" format. This format differs from the format used by DSS smartcards in that the DSS protocol (the "T=0" format) calls for the master device (the receiver or IRD) to send a 5-byte header block to the card. The card (or CAM) must then send back one of the bytes from the header (specifically, the second byte, which is the INS byte) to acknowledge receipt of the header. At this point, the IRD will either send the rest of the message to the CAM as one large packet, send the rest of the packet one byte at a time, awaiting an acknowledgment after each byte, or await the data return from the CAM.

The T=1 protocol, on the other hand, is defined such that any of 7 devices all connected to the same ISO-7816 bus may initiate a transmission to any of the other devices on the bus. In addition, the message will either be sent all at once (if it is short enough to fit in the destination device's receive buffer), or broken into smaller packets if the message is too long to be sent all at once

The protocol used by the EchoStar smartcards (and in fact, by any ISO-7816 compliant smart card) is requested by the card when it is reset by a master device. The card will send a sequence of data at a fixed baud rate (input clock frequency/372...It seems like an arbitrary number, but if the input clock frequency is 3.579545 MHz, the baud rate is 9622 bps...and although

3.579545 MHz seems like a strange number, it's actually pretty common: It's the frequency used by the colorburst crystal in NTSC television and set-top boxes). This data will include information about the data format the card wants to use, the baud rate at which it will want to communicate, and so on. To better understand the EchoStar cards, let's look at the ATR sent by a ROM version 2 EchoStar card:

```
3F FF 95 00  FF 91 81 71  64 47 00 44  4E 41 53 50
30 30 33 20  52 65 76 3x  3x 3x nn
```

If we break this ATR up into its constituent parts, we can decode it as follows:

```
3F ...                              Convention definition
|
|____________ Inverse convention (data is inverted)

FF 95 00 FF 91 ...                     Initial parm setup
| | | | |
| | | | |_ Td1=91 (Ta2 and Td2 will be sent, Protocol is async
| | | |          half duplex block format)
| | | |____ Tc1=FF (Guard time=257 bits)
| | |______ Tb1=00 (No Vpp)
| |_________ Ta1=95 (F=512, D=16; Bit period=(512/16) (32) clocks)
|____________ T0=FF (Ta1, Tb1, Tc1, and Td1 will be sent, 15
              historical characters will be sent)

81 71 . .                           Secondary parameters
| |
| |_________ Td2=71 (Ta3, Tb3, and Tc3 will be sent, protocol is async
|             half duplex block format)
|____________ Ta2=81 (Mode change not allowed, Protocol is async half
              duplex block format)

64 47 00  .                           T=1 specific parameters
| | |
| | |______ Tc3=00 (LRC (XOR-type) error checking to be used)
| |_________ Tb3=47 (Character wait time is 25 bit times. block wait time
|             is 634.9 mSec + 11 bit times) (1 bit time=7.111
|             uSec)
|____________ Ta3=64 (Receive block size=0x64 bytes (100 bytes decimal)

44 4E 41 53 50 30 30 33 20 52 65 76 3x 3x 3x ..   Historical bytes
|                    |
|____ ________________________________|
   |
   |______ ASCII text: "DNASP003 Revxxx". This is just an eye-catcher
         and/or ego boost for the Echo boys. It's the ROM version
         and EEPROM revision level of the firmware in the CAM.

nn
|____________ Checksum (all other bytes XORed together)
```

The data format (the T=1 format) is selected by bytes Td1, Td2, and Ta2. Note that all of them agree that the data format is asynchronous half-duplex block transfer. The baud rate is defined by byte Ta1. The upper nibble of this byte defines parameter F (frequency) as being 512, while the lower nibble defines parameter D (divisor) as being 16. The bit rate is found by dividing the card's input clock frequency by the quantity (F/D) (in the case of EchoStar cards, F/D = 512/16 = 32). If we check the clock being fed to the smartcard by an EchoStar IRD, we find that the master clock frequency, f, is either 4.5 MHz or 4.0 MHz, depending on the model IRD being used. Thus, the final baud rate for communication between an EchoStar IRD and smartcard is 4,500,000/32 (140,625) bps, or 4,000,000/32 (125,000) bps. Note that this bit rate is only necessarily 140,625 bps when the card is in an EchoStar IRD. If the card is in a programmer that feeds a 3.6864 MHz clock to the card, the final baud rate will be 115,200 bps.

The parameters such as "guard time", "character wait time", and "block wait time" apply only to messages being sent to the card from the IRD. These delays exist to allow the card enough time to move received data into its internal buffers and perform any necessary decryption on the received data before the next byte is received. It is assumed that the master device will not need such delays, since the master device most likely has a great deal more processing horsepower than the smartcard. In the case of the EchoStar smartcard, a delay of at least 25 bit times (178 microseconds) is required between bytes, and a delay of at least 635 milliseconds is required between whole blocks.

The Tc3 byte specifies that the card is going to use LRC (Longitudinal Redundancy Checking) as its means of error correction. This means that for any message sent, the final byte of the message will be the XOR-sum of all of the other bytes in the message. The other possibility for error checking (which is required by the ISO-7816 spec) is CRC checking, which would be selected if Tc3 was equal to 1.

The Ta3 byte specifies that the maximum message size that the card can accept it 0x64 (100 decimal) bytes. If the receiver wants to send a message that's longer than this, it has to break it up into smaller packets. In EchoStar ROM version 1 cards, Ta3 was 0x60 (96 decimal). One interesting thing to note, however, is that the first thing most IRDs I've seen do after they reset the smartcard is request that the smartcard shrink its buffer size to 0x58 (88 decimal) bytes.

The historical bytes are really nothing more than superfluous identification bytes that are used by the master device to learn additional information about the smartcard. In this case, the card sends ASCII text indicating that it is a Dish Network Smartcard ("DNASP003"), and its current EEPROM code revision ("Revxxx"). As of this writing, the current smartcard EEPROM revision level is 369, so this text will probably read "Rev369".

Finally, the ATR is terminated by a checksum, which is the XOR-sum of all of the other bytes in the ATR.

1.2- EchoStar's packet structure, part 1: The ISO-specified portion

The T=1 protocol has a very specific format for the data packets. The first 3 bytes of the packet, as well as the last byte (or the last two bytes if CRC checking is used) of the packet are defined specifically as follows:

Byte 1: Node address byte (NAD)
Byte 2: Procedure control byte (PCB)
Byte 3: Length byte (LEN)
Last byte: Checksum (LRC)

Thus, an ISO-7816 compliant T=1 message looks like this:

NAD PCB LEN <information field> LRC

The NAD is used to route messages. The upper nibble of the NAD is defined as the target address, and the lower nibble is defined as the source address. In the EchoStar system, only two addresses are defined: Address 1 is the receiver, and address 2 is the smartcard. Although 4 bits appear to be available for addressing, in reality, only the lower 3 bits of each nibble are available for addresses. The upper bit of each nibble is reserved for Vpp control requests. Since the EchoStar system doesn't use these bits, they won't be discussed here.

The PCB is used to identify what type of data is being sent, whether it's part of a block that's been broken up due to buffer size restrictions, if it's a special type of control request, and so forth. There are 3 basic formats for the PCB, as follows:

A standard "instruction" block has a PCB that looks like this:

```
%0NC00000
|||
||L__________ "Chain" bit. If this bit is set, it means that this
||            packet requires at least one more packet before the
||            entire message is considered "sent".
|L____________ "sequence Number" bit. This bit should toggle between
|             messages. It's used to help ensure that packets were
|             not missed if the chain bit is set...if the smartcard
|             misses a single packet (or any odd number of packets)
|             from the master, it will know that data is corrupt.
L______________ "0" in bit 7 indicates "instruction block".
```

A standard response block from the smartcard will have a PCB like this:

```
%100N000E
LJ |  |L_ Errors detected either in LRC byte or in parity.
 | |  L___ Other errors occurred
 | L________ "sequence Number" bit. This bit should match the N
 |            bit for the message to which the smartcard is
 |            responding.
 L______________ "10" in bits 7-6 indicates "response block"
```

Other control requests all follow a common format as follows:

```
%11R000TT
LJ|    LJ
 ||     L__ Request type as follows.
 ||          00=Resync (complete reset)
 ||          01=IFS (Information Field Size)
 ||          10=Abort
 ||          11=WTX (Wait time)
 |L__________ Request/Response (0=request, 1=response)
 L______________ "11" in bits 7-6 indicates "control request"
```

For an IFS request, a single byte of data will be included that tells the target what size the source would like it to change its IFS to. Note that although it is theoretically possible to request that the card change its IFS to a value larger than the one specified in the ATR, the target would probably not respond favorably to such a request. A sample IFS request (taken from an

EchoStar log) might look like this:

```
21 C1 01 58 89    Receiver requests card change its IFS to 0x58 bytes
12 E1 01 58 AA    Card acknowledges request
         |
         |_______ Requested IFS
```

For a WTX request, a single byte of data will be included that tells the target what new value the source would like it to use as a block wait time when the target is sending data to the source. This value is a multiple of the Block Wait Time (BWT) specified by the source in its ATR.

1.3- EchoStar's packet structure, part 2: The IRD-to-CAM information field

For the most part, all EchoStar messages will have an information field with a common structure. Since the IRD and the CAM have free reign over the use of the information field of an information block, we have to make guesses as to the definition of certain fields. Fortunately, the data is repetitive enough that we can make some pretty good guesses.

For data being transferred from the IRD to the CAM, the information field (not including the checksum) will always look like this:

```
A0 CA 00 00 CL CN DL <data1..dataDL-1> RL
|_________| | | |          | |_ Expected length of response
   |        | | |          |____ Command data...total of CD bytes
   |        | | |                will be sent, including RL
   |        | | |_________________ Command Data length
   |        | |____________________ Command Number
   |        |_______________________ Command Length
   |___________________________________ Header: Always A0 CA 00 00
```

Note that there are two length bytes. Command Length (CL) and command Data Length (DL). In actual practice, DL should always be equal to CL minus 2, since CL counts the Command Number (CN) and the command Data Length byte (DL) in its total. Note also that although CL and DL define the number of bytes that will be present in the entire message, these numbers may be larger than the overall size of the receive buffer in the smartcard. These bytes are what the card uses to determine how many bytes the entire packet will contain after being put back together (the LEN byte always specifies the total number of bytes to be sent in a single burst).

Note also that the IRD indicates how much data it wants to see back from the CAM. The Response Length byte (RL) specifies how many bytes of data the IRD expects to see in the CAM's response, not including the 3-byte ISO-7816 header and the SW1/SW2/LRC trailer. Thus, if the IRD specifies a value of 03 for the RL, the CAM will send a total of 9 bytes back (3 bytes of ISO-7816 header, 1 byte of response type, 1 byte of data length, 1 byte of data, 2 bytes of SW1/SW2 info, and 1 byte of LRC).

1.4- EchoStar's packet structure, part 3: The CAM-to-IRD information field

When the CAM responds to a command from the IRD, its information field also has a rigidly defined structure. Although this structure is different from that of the IRD-to-CAM messages, it's not too tough to figure out. Messages sent from the CAM to the IRD look like this:

```
CC AL <data1..dataAL> SW1 SW2
```

```
||        ||____|_________ Standard ISO-7816 status word.
||        |            Usually 90 00.
||        |______________ Data being returned to IRD. This
||                     data field may have several data
||                     elements embedded within
||______________________________ Length of data field in bytes.
|                      This byte will always be equal
|                      to RL-2 (see section 1.3 for an
|                      explanation of the value of RL).
|_________________________________ Command response code. This is
                       related directly to the command
                       number for which this message is a
                       response. There's no apparent
                       set relationship between the
                       response code and the command number,
                       but each command number does
                       relate to a specific response code
                       (see section 2: Commands)
```

==

2.1- Known command list

Following is a list of all the commands I know about at this point. It's a real good bet there's others, and as I find them (or as they're pointed out to me), they'll be added here.

Cmd #	Function
$00	Entitlement Management Message (EMM)
$01	Unknown-possible PPV Entitlement Management Message
$02	Unknown-possible simulcrypt Entitlement Control Message
$03	Entitlement Control Message
$10	Unknown-existance unconfirmed, but probable
$11	Unknown-existance unconfirmed, but probable
$12	Serial Number Request
$13	Control Word Request (video decryption key request)
$14	unknown data request
$20	Data items available request
$21	Data item request
$30	Request for encryption of data to be sent in callback
$31	Request for data encrypted by previous command $30
$40	EEPROM data space available request
$41	PPV buy-related
$42	PPV buy-related
$50	Unknown-existance unconfirmed, but probable
$51	Unknown-existance unconfirmed, but probable
$52	Unknown-existance unconfirmed, but probable
$53	Unknown-existance unconfirmed, but probable
$54	Unknown-existance unconfirmed, but probable
$55	Unknown-Possibly pending purchase related
$56	Unknown-Possible pending purchase related
$60	Dump RAM from $80..$CF
$61	Write IRD info
$99	Unknown data request
$C0	CAM status request

$C1 Request for ID of updated data items

2.2- Command lengths, expected replies, and reply lengths

Each EchoStar command is a fixed length. In addition, for any command that the IRD sends to the CAM, there's a specific response that the IRD expects. In most cases, the IRD specifies the total length of data it expects back to the response as the last byte of the information field.

Cmd #	Length	Expected response	Response length
$00	$53	$80	$07
$01	$53	$81	$07
$02	unknown	$82	unknown (probably $07)
$03	$35/$3D/$45	$83	$07
$10	unknown	$90	unknown
$11	unknown	$91	unknown
$12	$08	$92	$08
$13	$09	$93	$1B
$14	$08	$94	$08
$20	$0C	$A0	$05
$21	$0D	$A1	variable
$30	$0B	$F0	$07
$31	$08	$F1	$54
$40	$08	$70	$04
$41	unknown	$71	unknown
$42	$0F	$72	$05
$50	unknown	$D0	unknown
$51	unknown	$D1	unknown
$52	unknown	$D2	unknown
$53	unknown	$D3	unknown
$54	unknown	$D4	unknown
$55	$0B	$D5	$08
$56	$0B	$D6	$08
$60	$08	$E0	$44
$61	$1C	$E1	$03
$99	$20	$99	$1C
$C0	$08	$B0	$08
$C1	$08	$B1	$06

Note that if an entry in the above table is listed as "unknown", that doesn't mean that nobody knows ..it just means that _I_ don't know yet, because I haven't logged that particular type of packet yet.

2.3- Command breakdown

Here, we'll be examining each command in detail, discussing their functions, data structures, etc.

2.3.1- Command $00/Response $80

This command is used to perform all manner of changes to the card: Updates to code in EEPROM, attacks against hacked cards, and so on. The data for this command is a 64-byte block which contains subcommands of its own. These subcommands can be anything from "change spending limit" to "execute embedded code from RAM" Note that although the data block is always 64 bytes long, not all of the bytes are necessarily used. The data block is padded to 64 bytes, probably with random data, prior to encryption. Immediately preceding the 64 byte data block is an 8-byte signature which is computed based on the unencrypted data to confirm that the data is authentic before the commands within are executed. In the DVB world, this type of message is referred to as an Entitlement Management Message or EMM. The 00 EMM is a public EMM.

Example of a $00 command and its response:

```
21 40 53 ; A0 CA 00 00          ;Standard header
      4D                        ;Command length
      00                        ;Command
      4B                        ;Command data length
      01 01 82                  ;Misc. data
      DC E9 1B 55 69 A0 D5 22   ,Signature
      16 48 08 B1 A4 84 8F 2B   ;Encrypted packet number 1
      3B DA AC 07 A2 31 B0 83   ;Encrypted packet number 2
      E2 27 5D 47 C8 27 D5 0E   ;Encrypted packet number 3
      7C 9C A6 51 E5 0D FE 18   ,Encrypted packet number 4
      54 04 80 26 49 FC A6 71   ;Encrypted packet number 5
      B2 F4 69 51 0C 22 B7 24   ;Encrypted packet number 6
      BF 7E 3F 64 D0 1A BC 24   ,Encrypted packet number 7
      F3 6E 88 6E 69 0F 0F 61   ;Encrypted packet number 8
      05                        ;Expected response length
      EB                        ;Checksum

12 40 07 , 80                   ;Standard header, response code
      03                        ;Response data length
      B1 01                     ;Decode succeeded
      04                        ;Sequence number
      90 00                     ;SW1/SW2
      F2                        ;Checksum
```

And a decrypted 00 packet (not the one above, though):

```
3F 01 01
60 0A
42 05 88 4E 25 22 BF 91 4A E4
42 B5 DF 6A 21 6A 46 A8 19 2D
20 00 33 00 00 00 0F 06 08 21 0C 21 0C 21
00

03 B5 4C 31 71 19 5A 9B
C0 C5 C9 32 88 03 8D B2
E6 8C 45 CA 20 A1 06 CD
```

As you can see, the first $28 bytes of decrypted data are EMM commands (see section 3, EMM commands), and the last $18 bytes are just random data that exists only to reduce the chances of two 00 packets looking even remotely alike.

2.3.2- Command $01/Response $81

Command 01 seems to be a card-specific version of message 00. Its primarily seen when purchasing a PPV event

Example of a $01 command and its response:

None available at this time

2.3.3- Command $02/Response $82

I have very little information on command $02 at this time. What I gather from what I've seen on the 'net is that this command may be used to transfer further decryption keys that are used in conjunction with the standard 03-type decryption. The data within this command seems to be encrypted, and the decrypted version of the data is XORed with the decrypted control words sent in the 03 commands before re-encrypting them and returning them to the IRD.

Example of a $02 command and its response.

None available at this time

2.3.4- Command $03/Response $83

This command is used to prime the card to return video decryption keys to the IRD. Contained within this command's encrypted packets are information pertaining to the program tier the user is attempting to view, the correct audio and video decryption keys for the channel, current date and time, and so forth. When the card receives a $13 command, it will re-encrypt the decryption keys using the IRD's 8-byte key and return them to the IRD if it (the card) believes that the program tier that the user is attempting to watch is one for which they are authorized.
In addition to information about the program that the user is attempting to watch, the 03 command contains information about the encryption method used, how many encrypted video keys are present, and so forth. For a detailed explanation of the control bytes, see the breakdown of the decrypted packets, below

Example of a $03 command, both encrypted and decrypted, and its response:

```
21 00 35 ; A0 CA 00 00          ;Standard header
      2F                        ;Instruction length
      03                        ;Command
      2D                        ;Command data length
      01 01 10                  ;Misc. data
      29                        ;Data field length
      05                        ;Key select byte (see below)
      12 D4 70 4D 34 FB 3C DF   ;Valid hash (signature)
      90 DD 23 D3 7B A9 79 DC   ;Encrypted packet 1
      CF DE 68 4E A4 43 0F 1B   ;Encrypted packet 2
      F5 E0 9B B3 30 58 FB 75   ;Encrypted packet 3
      4F 3E AB EB 4C 8F F0 6F   ;Encrypted packet 4
      05                        ;Expected response length
      29                        ;Checksum

12 00 07 ; 83                   ;Response code
      03                        ;Response data length
      B1 01                     ;Decode succeeded
```

```
03                    ;Sequence number
90 00                 ;SW1/SW2. Successful completion
B6                    ;Checksum
```

Here's what the data in the encrypted packets looks like after decryption:

```
10                       ;Control word control byte (see below)
80                       ;Simulcrypt control byte 1 (see below)
39 31 33 9D 31 32 35 98  ;Control word 1
80                       ;Simulcrypt control byte 2 (see below)
34 30 32 96 34 35 34 9D  ;Control word 2
24 01 0D 86 B3 54        ;Tier info
00                       ;End of standard data marker
55 55 55 55 55 55        ;Pad bytes to make encrypted data come
                         ;out to a multiple of 8 bytes
```

Key select byte: This byte is used to tell the $03 decode routine which public key (if any) was used to encrypt the data in the 03 command. If the low 3 bits of this byte are "101", then the CAM uses the standard E* DES-like decrypt on the data. If the low 3 bits are "111", the CAM doesn't perform any decryption on the data at all. If the low 3 bits are anything other than "101" or "111", the CAM discards the packet. Bit 4 of this byte indicates which public key was used. 0 for key 0, 1 for key 1. Bit 3 is a flag of some sort, but I don't yet understand its significance.

Control word control byte. If this byte is $10, then the CAM knows that two control words are present. If this byte is $11, then the CAM only tries to decrypt the first control word, but it returns it as the second return word in the next command 13.

Simulcrypt control byte: If bit 7 of this byte is clear, then before the CAM re-encrypts the control words for return to the IRD, it XORs them with a sequence of bytes that probably come from an 02 command. The 02 command seems like it probably contains 16 or 18 bytes of XOR data, and the starting position for the XOR is determined by the value of the simulcrypt control byte and the value of a control byte which is included with the XOR data itself.

Note that the above version of the 03 packet is the $35-byte length version. There's also a $3D-byte length version which contains additional information about the current time of day as well as the start time of the program being watched. An example of the $3D-length version would look like this:

```
21 00 3D ; A0 CA 00 00          ;Standard header
     37                         ;Instruction length
     03                         ;Command
     35                         ;Command data length
     01 01 10                   ;Misc. data
     31                         ;Data field length
     05                         ;Misc. data
     B8 2E A0 FF B4 F8 2C 2F    ;Valid hash (signature)
     3C 4F C7 CB 26 E1 FF 3E    ;Encrypted packet 1
     C0 4D F2 6F 37 C3 22 8D    ;Encrypted packet 2
     4B 42 B2 BC 99 15 3B D4    ;Encrypted packet 3
     5D 95 0D 0A 93 57 0F 7A    ;Encrypted packet 4
     69 50 D7 4E 2C B5 ED CB    ;Encrypted packet 5
     05                         ;Expected response length
     A6                         ;Checksum
```

```
12 00 07 ; 83           ;Response code
      03                ;Response data length
      81 01             ;Decode succeeded
      03                ;Sequence number
      90 00             ;SW1/SW2 Successful completion
      86                ;Checksum
```

And the decrypted data...

```
      10                ;Control word control byte
      80                ;SIMULCRYPT control byte 1
      33 36 34 8D 34 30 35 99  ;Control word 1
      80                ;SIMULCRYPT control byte 2
      36 31 32 99 30 33 39 8C  ;Control word 2
      20 00 1D 80 00 FF        ;Tier info
      00                ;End of standard data marker
      0D 8C 88 00       ;Program start time (30 June '98, 17:00:00 GMT)
      0D 8C 92 16       ;Current time (30 June '98, 18:15:38 GMT)
      00                ;End of time-of-day marker
      55 55 55 55 55    ;Pad bytes
```

Time-of-day info: The E* time of day seems to be encoded as follows: The first 2 bytes are the number of days since 1 January, 1989. The last 2 bytes are the number of "ticks" since midnight. It seems as though there are 2048 ticks in an hour, or roughly 1.75 seconds per tick. This information is just speculation on my part: If anyone knows for sure what the format of the time and date are, please let me know.

Further notes about the 03 packet: The two control words, the SIMULCRYPT control bytes, and the control word control byte must ALWAYS be located at the start of the encrypted data as shown above. The EchoStar CAM is hard-coded to expect the control word data and associated control bytes to be in those locations.

2.3.5- Command $10/Response $90

I have no information about this command at this time. In fact, I don't even have solid proof that the command exists, other than the fact that E* seems to group their commands together starting at MOD16 boundaries. Because a $12 command exists, I assume the existence of a $10 command.

2.3.6- Command $11/Response $91

I have no information about this command at this time. In fact, I don't even have solid proof that the command exists, other than the fact that E* seems to group their commands together starting at MOD16 boundaries. Because a $12 command exists, I assume the existence of a $11 command.

2.3.7- Command $12/Response $92

This command is used by the IRD to request the CAM's serial number. If you look at the underside of your CAM, you'll see a 12-digit, bar-coded number. This number is your CAM ID, and every CAM has a unique one. The first 10 digits of the number are significant, and the last 2 are a 2-digit check code. If you take the 10-digit serial number and convert it to hex, you'll have your card's hex CAM ID. For example, let's say your card's serial number is 0057386394. In hex, that would be 36BA59A. The $12 packet and its response for this CAM would look like this:

```
21 00 08 ; A0 CA 00 00          ;Standard header
       02                ;Instruction length
       12                ;Command
       00                ;Command data length
       06                ;Expected response length
       55                ;Checksum

12 00 08 , 92              ;Response code
       04                ;Response data length
       03 6B A5 9A         ;CAM ID: 036BA59A (00 5738 6394 xx)
       90 00              ;SW1/SW2. Successful completion
       48                ;Checksum
```

Note that the two-digit check code isn't included as part of the response. This check code is only used by the Dish Network customer service drone to ensure that you do (or at one time did), in fact, have physical possession of the card when you call to subscribe.

2.3 6- Command $13/Response $93

This command is used by the IRD to request the decryption keys for the channel to which the IRD is currently tuned. This command is a counterpart to command $03- The decryption keys are sent from the IRD to the CAM in an encrypted form that the IRD doesn't know how to decode along with information that tells the CAM which channel the IRD is tuned to. If the CAM decides that the user should be able to view the specified channel (ie , if there is a valid subscription tier in the CAM for the specified channel), then when the next $13 command is issued, the CAM will decrypt the data it was given in the $03 command, re-encrypt it using a key and method known to the IRD, and send the data back to the IRD, which will then decrypt the data and use it to decrypt the video and audio data streams. A typical $13 packet and it's associated response would look like this:

```
21 00 08 , A0 CA 00 00          ;Standard header
       03                ;Instruction length
       13                ;Command
       01                ;Command data length
       03                ;Command data
       19                ;Expected response length
       09                ;Checksum

12 00 08 , 93              ;Response code
       17                ;Response data
       81 01 06             ;Misc. data
       11                ;Video key 1 header
       08                ;Video key length
       11 22 33 44 55 66 77 88   ;Encrypted video key 1
       12                ;Video key 2 header
       08                ;Video key length
       11 22 33 44 55 66 77 88   ;Encrypted video key 2
       90 00              ,SW1/SW2. Successful completion
       CE                ;Checksum
```

Note: If the card is using the SIMULCRYPT extension, once the box has

finished decoding the 13 command, it will need to XOR the video keys with the same data the CAM did, if it expects to recover the raw data that was present in the 03 command. Note also that regardless of whether the 03 command had a $10 or $11 control byte (indicating the presence of two or one control words, respectively), the 13 command will ALWAYS have two control words in it, and will always be formatted as above.
Also note: If the IRD is requesting keys for a channel that the CAM thinks it's not authorized for, the CAM will return all 00s for the encrypted video keys.

2.3.7- Command $14/Response $94

I don't know much about this command, other than the fact that it's always the same in virgin cards. An example of a virgin $14 command is:

```
21 00 08 ; A0 CA 00 00       ;Standard header
      02                     ;Instruction length
      14                     ;Command
      00                     ;Command data length
      06                     ;Expected response length
      C7                     ;Checksum

12 00 08 ; 94                ;Response code
      04                     ;Response data length
      0F 4C 54 60            ;Data
      90 00                  ;SW1/SW2: Successful completion
      0D                     ;Checksum
```

2.3.8- Command $20/Response $A0

This command is used by the IRD to ask the CAM whether it has any data of a specific type stored inside it. Basically, the $20 command is a poll for "number of data items to report". If the CAM responds with a non-zero value, the IRD will then proceed with appropriate $21 commands to get the data from the CAM. An example of the $20 command would look like this

```
21 00 0C ; A0 CA 00 00       ;Standard header
      06                     ;Instruction length
      20                     ;Command
      04                     ;Command data length
      01                     ;Data type being queried
                             ;(see section 4, data types)
      02                     ;Second data field length
      FF FF                  ;Misc. data
      03                     ;Expected response length
      25                     ;Checksum

12 00 05 ; A0                ;Response code
      01                     ;Response data length
      01                     ;Number of data items to return
      90 00                  ;SW1/SW2. Successful completion
      67                     ;Checksum
```

2.3.9- Command $21/Response $A1

This command is used by the IRD to actually request data from the CAM.

The IRD should only send this command if it receives a non-zero value from the CAM in response to the $20 command for the requested data type. Each data type has its own response length and structure. An example of a 21 command is as follows:

```
21 00 0D , A0 CA 00 00      ;Standard header
       07                   ;Instruction length
       21                   ;Command
       05                   ;Command data length
       01                   ;Data type being requested
                            ;(see section 4, data types)
       03                   ;Second data field length
       FF FF                ;Misc. data
       00                   ;Element number being requested
       20                   ;Expected response length
       47                   ;Checksum
```

Note that the IRD already knows how long the response to its query should be. This is because most data types are fixed-length. For data types that are NOT fixed-length, the data will always have a fixed-length field to start, and one or more variable-length fields to follow. An example of a series of 21 commands that retrieve a variable-length data item is as follows:

```
21 40 0D , A0 CA 00 00      ;Standard header
       07                   ;Instruction length
       21                   ;Command
       05                   ;Command data length
       11                   ;Data type
       03                   ;Second data field length
       FF FF                ;Misc. data
       00                   ;Element number being requested
       04                   ;Expected response length
       33                   ;Checksum

12 40 06 ; A1               ;Response code
       02                   ;Response data length
       01                   ;Subtype of data in this element
       11                   ;Length of this element
       90 00                ;SW1/SW2 Successful completion
       69                   ;Checksum


21 00 0D , A0 CA 00 00      ;Standard header
       07                   ;Instruction length
       21                   ;Command
       05                   ;Command data length
       11                   ;Data type
       03                   ;Second data field length
       FF FF                ;Misc. data
       00                   ;Element number being requested
       15                   ;Expected response length (note: Equals
                            ;length reported by CAM above plus 4, to
                            ;account for
       64                   ;Checksum

12 00 17 ; A1               ;Response code
```

```
        13              ;Response data length
        01              ;Data type to follow. date/time
        0F 0E 35 28        ;Date/time: 21 July. 1999, 06:38:40 GMT
        00              ;End of date/time
        44 65 65 70 20 49 6D
        70 61 63 74 90        ;"Deep Impact" in ASCII
        00              ;End of PPV title
        90 00            ;SW1/SW2: Successful completion
        0C              ;Checksum (note no SW1/SW2)


21 40 0D ; A0 CA 00 00        ;Standard header
        07              ;Instruction length
        21              ;Command
        05              ;Command data length
        11              ;Data type
        03              ;Second data field length
        FF FF            ;Misc. data
        01              ;Element number being requested
        04              ;Expected response length
        32              ;Checksum

12 40 06 ; A1            ;Response code
        02              ;Response data length
        02              ;Subtype of data in this element
        05              ;Length of this element
        90 00            ;SW1/SW2: Successful completion
        60              ;Checksum


21 00 0D ; A0 CA 00 00        ;Standard header
        07              ;Instruction length
        21              ;Command
        05              ;Command data length
        11              ;Data type
        03              ;Second data field length
        FF FF            ;Misc. data
        01              ;Element number being requested
        09              ;Expected response length
        7F              ;Checksum

12 00 08 ; A1            ;Response code
        07              ;Response data length
        02              ;Data type to follow. Channel
        05              ;Length of channel ID
        50 50 56 31 31        ;"PPV11" in ASCII
        90 00            ,SW1/SW2. Successful completion
        7E              ;Checksum
```

2.3 12- Command $30/Response $F0

This command seems to be a request for callback information. The IRD sends this command to the CAM, then waits until the CAM has a response ready. When the response is ready, the IRD requests the encoded callback data using the S31 command. Unfortunately, I don't have any idea as to what the data that's passed to the CAM in the 30 command represents The data is censored here because it may somehow represent card-specific data. An example of the S30 command looks like this:

```
21 00 0B ; A0 CA 00 00      ;Standard header
      05                    ;Instruction length
      30                    ;Command
      03                    ;Command data length
      xx xx xx              ,Misc. data
      05                    ;Expected response length
      xx                    ;Checksum

12 40 07 , F0               ;Response code
      03                    ;Response data length
      B1 01                 ;Packet OK
      01                    ;Misc. data
      90 00                 ;SW1/SW2. Successful completion
      87                    ;Checksum
```

2.3.13- Command $31/Response SF1

This command requests the return of the encoded callback data whose preparation was requested by a $30 command. A total of $48 bytes of data are returned, the first 8 of which are a valid hash value. I'm relatively certain that this information is encrypted using the same algorithm as 00 and 01 commands, though I don't know what key is used. The data in the example below is censored because it may contain card-specific data. An example of the S31 command is as follows:

```
21 00 08 ; A0 CA 00 00      ;Standard header
      02                    ;Instruction length
      31                    ;Command
      00                    ;Command data length
      52                    ;Expected response length
      22                    ;Checksum

12 40 54 ; F1               ;Response code
      50                    ;Response data length
      B1 01                 ;Misc. data
      01                    ;Misc. data
      04                    ;Length of status bytes
      4B xx xx xx           ;Status of encode
      xx xx xx xx xx xx xx xx  ;Valid hash
      xx xx xx xx xx xx xx xx  ;Encrypted packet 1
      xx xx xx xx xx xx xx xx  ,Encrypted packet 2
      xx xx xx xx xx xx xx xx  ,Encrypted packet 3
      xx xx xx xx xx xx xx xx  ;Encrypted packet 4
      xx xx xx xx xx xx xx xx  ;Encrypted packet 5
      xx xx xx xx xx xx xx xx  ;Encrypted packet 6
      xx xx xx xx xx xx xx xx  ,Encrypted packet 7
      xx xx xx xx xx xx xx xx  ;Encrypted packet 8
      90 00                 ;SW1/SW2: Successful completion
      xx                    ,Checksum
```

2.3.14- Command $40/Response S70

This command is a request for the CAM to tell us how much EEPROM space is available for data storage. The CAM figures this out by starting at the beginning of its data area, going through every data item it can find, adding the length of each as it finds it, and subtracting the final result from the total amount of EEPROM space known to be available. The result is returned in the $70 response. An example of the $40 command might look like this:

```
21 40 08 , A0 CA 00 00     ;Standard header
      02                   ;Instruction length
      40                   ;Command
      00                   ;Command data length
      04                   ;Expected response length
      45                   ;Checksum

12 40 06 , 70              ;Response code
      02                   ;Response data length
      0A 52                ;EEPROM space available: $A52 bytes
      90 00                ;SW1/SW2 Successful completion
      EE                   ,Checksum
```

Note that the returned value does not necessarily reflect the amount of contiguous data space available in the card's EEPROM ..the data storage mechanism used in the E* CAM is sort of a crude flash file system kind of thing, so when a data item is removed, the space it once occupied is considered to be "available", even though that space may be wedged between two other active data items This approach makes the E* CAMs more flexible than the DSS CAMs, but also means that the amount of code required to support the data storage system is much larger.

2.3.15- Command $41/Response $71

This command is part of the PPV buy mechanism. I'm not entirely sure what the included data does, but it appears to me as though this is the command that triggers the creation of a type 11 (PPV purchase) data item.

```
21 00 08 ; A0 CA 00 00     ;Standard header
      05                   ;Instruction length
      41                   ;Command
      03                   ;Command data length (note: I believe
                           ;that if this value is >$7F, no data
                           ;item is created
      50 50 56             ;Command data "PPV" in ASCII
      03                   ;Expected response length
      52                   ;Checksum

12 40 05 ; 71              ;Response code
      01                   ;Response data length
      14                   ;Resposne data
      90 00                ;SW1/SW2 Successful completion
      A3                   ;Checksum
```

2.3.16- Command $42/Response $72

This command is also part of the PPV buy mechanism. As with the $41 command, I'm not entirely sure what this command does, but I do know that when the CAM receives it, it does a search for any data item that has data matching the received command data, as follows:

Assume that 7 parameter bytes are passed to the routine, numbered P1 through P7. Those bytes are arranged in RAM as follows:

P1 P2 xx P4 P5 P6 P7

Where xx is a don't-care byte. It appears as though the CAM then performs a search for any data item whose first two bytes match P1 and P2, whose 3rd byte is anything, and whose next 3 bytes match P4..P6. If the search finds a match, the CAM performs some sort of operation on the located data (I don't know what yet, though). An example of a 42 command is as follows:

```
21 40 0F ; A0 CA 00 00    ;Standard header
     09                   ;Instruction length
     42                   ;Command
     07                   ;Command data length
     01                   ;Parameter byte P1
     01                   ;Parameter byte P2
     00                   ;Parameter byte P3
     7F                   ;Parameter byte P4
     08                   ;Parameter byte P5
     14                   ;Parameter byte P6
     01                   ;Parameter byte P7
     03                   ;Expected response length
     29                   ;Checksum

12 00 05 ; 72             ;Response code
     01                   ;Response data length
     14                   ;PPV item number?
     90 00                ;SW1/SW2: Successful completion
     E0                   ;Checksum
```

2.3.17- Command $50/Response $D0

I have no information about this command at this time. In fact, I don't even have solid proof that the command exists, other than the fact that E* seems to group their commands together starting at MOD16 boundaries. Because $55 and $56 commands exist, I assume the existence of a $50 command.

2.3.18- Command $51/Response $D1

I have no information about this command at this time. In fact, I don't even have solid proof that the command exists, other than the fact that E* seems to group their commands together starting at MOD16 boundaries. Because $55 and $56 commands exist, I assume the existence of a $51 command.

2.3.19- Command $52/Response $D2

I have no information about this command at this time. In fact, I don't even have solid proof that the command exists, other than the fact that E* seems to group their commands together starting at MOD16 boundaries. Because $55 and $56 commands exist, I assume the existence of a $52 command.

2.3.20- Command $53/Response $D3

I have no information about this command at this time. In fact, I don't even have solid proof that the command exists, other than the fact that E* seems to group their commands together starting at MOD16 boundaries. Because $55 and $56 commands exist, I assume the existence of a $53 command.

2.3.21- Command $54/Response $D4

I have no information about this command at this time. In fact, I don't even have solid proof that the command exists, other than the fact that E* seems to group their commands together starting at MOD16 boundaries. Because $55 and $56 commands exist, I assume the existence of a $54 command.

2.3.22- Command $55/Response $D5

The $55 command is sort of a mystery to me. I know that it has something to do with type $10 data (see section 4, data types), but I don't know for sure what that data is. Because it's variable-length, though, I'm guessing that type $10 data is some sort of "pending purchase" information. What I do know is that when the CAM receives a $55 command, it searches for a type $10 data item whose first two bytes match the first two parameter bytes, and whose fourth byte matches the third parameter byte. If a match is found, the entry is updated. An example of a $55 command is as follows:

```
21 00 08 ; A0 CA 00 00        ;Standard header
      05                      ;Instruction length
      55                      ;Command
      03                      ;Command data length
      11                      ;Parameter byte P1
      22                      ;Parameter byte P2
      33                      ;Parameter byte P3
      06                      ;Expected response length
      0B                      ;Checksum

12 40 08 ; D5                 ;Response code
      04                      ;Response data length
      11                      ;Parameter byte P1
      22                      ;Parameter byte P2
      33                      ;Parameter byte P3
      00                      ;00=match found, FF=no match
      90 00                   ;SW1/SW2: Successful completion
      FA                      ;Checksum
```

2.3.23- Command $56/Response $D6

Like the $55 command, the $56 command is currently beyond my understanding. Basically, it functions exactly the same as a 55 command, but different data in the located data item is modified if a match is found

```
21 00 08 ; A0 CA 00 00        ;Standard header
      05                      ;Instruction length
      56                      ;Command
      03                      ;Command data length
```

```
        11                    ;Parameter byte P1
        22                    ,Parameter byte P2
        33                    ;Parameter byte P3
        06                    ;Expected response length
        08                    ;Checksum

12 00 08 ; D6                     ;Response code
        04                    ;Response data length
        11                    ,Parameter byte P1
        22                    ,Parameter byte P2
        33                    ;Parameter byte P3
        00                    ;00=match found, FF=no match
        90 00                  ;SW1/SW2 Successful completion
        BA                    ;Checksum
```

2.3.24- Command $60/Response $E0

The $60 command is quite interesting. If the CAM is in a specific state (which is indicated by one of the bits in a $C0 command response), if the CAM receives a $60 command, it will return the $40 bytes of RAM from $80..$BF. This is interesting because inside the CAM, that's where the decrypted EMM data is stored. Unfortunately, at this point, I don't know how to cause the CAM to be in a mode where it's receptive to the $60 command. Note that if the CAM return data for a $C0 command indicates that it will accept a $60 command, the $60 command MUST be the next command other than $C0 that is sent to the card. All other commands cause the bit which makes the CAM dump the RAM area to be cleared. An example of the $60 command is as follows.

```
21 00 08 ; A0 CA 00 00            ;Standard header
        02                    ,Instruction length
        60                    ;Command
        00                    ;Command data length
        42                    ;Expected response length
        63                    ,Checksum

12 40 44 ; E0                     ;Response code
        40                    ;Response data length
        xx xx xx xx xx xx xx xx       ;Response data
        xx xx xx xx xx xx xx xx
        xx xx xx xx xx xx xx xx
        xx xx xx xx xx xx xx xx
        xx xx xx xx xx xx xx xx
        xx xx xx xx xx xx xx xx
        xx xx xx xx xx xx xx xx
        xx xx xx xx xx xx xx xx
        90 00                  ;SW1/SW2 Successful completion
        26                    ;Checksum
```

2.3.25- Command $61/Response $E1

The $61 command is used to write IRD-specific information to the CAM. It can be used to write the serial number of the IRD as well as 16 bytes of miscellaneous data that E* uses to store the version numbers of the bootstrap and firmware in the IRD. The format of the 61 command is as follows:

```
21 40 1C . A0 CA 00 00        ;Standard header
      16                      ;Instruction length
      61                      ;Command
      14                      ;Command data length
      33 22 11 00             ;IRD serial number
      aa aa aa aa aa aa aa aa ;IRD firmware info in ASCII
      aa aa aa aa aa aa aa aa
      03                      ;Expected response length
      xx                      ;Checksum

12 40 05 ; E1                 ;Response code
      01                      ;Response data length
      00                      ;Response data (always 00)
      90 00                   ;SW1/SW2: Successful completion
      27                      ;Checksum
```

2.3.26- Command $99/Response $99

The $99 command is very odd. It's the only command I know of that has a response code that's the same as the command itself. Also, it seems to have at least two different formats. There is speculation that the output of a $99 command is related to the data contained in the $00 command most recently decoded by the card, but I can't confirm that. Because I know so little about this command, I'm not going to provide an example of it yet.

2.3.27- Command $C0/Response $B0

This command returns 4 bytes of CAM status bits. Some are completion codes for the last command sent, some are to signal that the CAM has finished processing the last command (as in the $03), some indicate that the CAM has been updated by an EMM ($00) and the IRD should poll for the new info. A detailed breakdown of the bits I understand follows the example:

```
21 00 08 , A0 CA 00 00        ;Standard header
      02                      ;Instruction length
      C0                      ;Command
      00                      ;Command data length
      06                      ;Expected response length
      B7                      ;Checksum

12 00 08 , B0                 ;Response code
      04                      ;Response data length
      08                      ;Response byte 1
      00                      ;CAM status flags 1
      00                      ;CAM status flags 2
      16                      ;CAM status flags 3
      90 00                   ;SW1/SW2: Successful completion
      20                      ;Checksum
```

Bit-by-bit breakdown of CAM status flags bytes:

Bit #	Flags 1	Flags 2	Flags 3
0	CAM suggests $C1 cmd		Cmd $03 in progress

1	CAM suggests $C0 cmd	Cmd 00/01 received	Cmd $13 data ready
2		Cmd 00/01 decrypt bad	
3		Cmd $30 in progress	
4		Cmd $31 data ready	
5		Cmd $60 allowed	Cmd $02 in progress
6			
7			

2.3.28- Command $C1/Response $B1

This command queries the CAM as to the existence of data items that have changed since they were last polled. The return data is a bit mapped field, with each bit representing whether or not a particular type of data has changed since the last check. When the CAM first powers up, it will respond to this command with $CF FF in the return so that the IRD will request all available data An example of the $C1 command is as follows:

```
21 00 08 ; A0 CA 00 00          ;Standard header
       02                ;Instruction length
       C1                ;Command
       00                ;Command data length
       04                ;Expected response length
       84                ;Checksum

12 00 06 , B1                   ;Response code
       02                ;Response data length
       CF                ;Response data byte 1 (see below)
       FF                ;Response data byte 2 (see below)
       90 00             ;SW1/SW2: Successful completion
       07                ;Checksum
```

Format for response data bytes:

Bit #	Data byte 1	Data byte 2
0	Data type $11 has changed	Data type $08 has changed
1	Data type $10 has changed	Data type $07 has changed
2	n/a	Data type $06 has changed
3	n/a	Data type $05 has changed
4	Data type $0C has changed	Data type $04 has changed
5	Data type $0B has changed	Data type $03 has changed
6	Data type $0A has changed	Data type $02 has changed
7	Data type $09 has changed	Data type $01 has changed

==

3.1- EMM command list

As mentioned in section 2.3.1, the EMM (Entitlement Management Messages or ,in the EchoStar system include an encrypted 64-byte block of data which, after decryption, is interpreted as a list of EMM commands. In this section, each EMM command will be addressed individually, and their functions and formats will be explained (where known). Note that although I think I know what all of the valid EMM commands are, I don't know what they all do. Also note that because of the differences in the 288-01 and 288-02 cards' ROM Images, there are at least two EMM commands which will work on one of the cards, but not the other (see section 5, inside EchoStar cards).

EMM CMD	Function
$00	End of EMM sequence
$10	?
$11	PPV write (not verified)
$12	?
$13	?
$14	?
$15	?
$20	?
$21	?
$22	?
$23	Unknown, seems to relate to spending limit
$24	?
$25	?
$26	?
$2F	?
$30	?
$31	?
$32	?
$40	Activation-related (not verified)
$41	?
$42	Control word key change
$43	?
$44	IRD info change
$45	?
$50	?
$51	?
$52	?
$53	?
$54	?
$60	?
$61	?
$62	?
$63	?
$64	?
$80	?
$81	?
$82	?
$83	?
$84	?
$85	?
$86	?
$F0	Execute code from RAM (288-01 cards)
$F3	Execute code from RAM (288-02 cards)

3.2.1- EMM command $00

This EMM command is used to let the CAM know that no more EMM commands are contained in the 64-byte block. This is necessary because the encrypted data block is padded to 64 bytes with random data, so a mechanism needs to be defined to prevent the random data from being interpreted as commands This command does not include any parameters.

3.2.2- EMM command $10

I don't know what this EMM command does, but it definitely exists on 288-01 cards.

3.2.3- EMM command $11

I've been told that this command is used to write PPV events to the card, but I don't know anything at all about its format, nor can I verify that it does, in fact, relate to PPV buys in any way.

3.2.4- EMM command $12

I don't know what this EMM command does, but it definitely exists on 288-01 cards.

3.2.5- EMM command $13

I don't know what this EMM command does, but it definitely exists on 288-01 cards

3.2.6- EMM command $14

I don't know what this EMM command does, but it definitely exists on 288-01 cards.

3.2.7- EMM command $15

I don't know what this EMM command does, but it definitely exists on 288-01 cards.

3.2.8- EMM command $20

I know nothing of this command other than it seems to have a $10 byte data field associated with it.

3.2.9- EMM command $21

I don't know what this EMM command does, but it definitely exists on 288-01 cards.

3.2.10- EMM command $22

I don't know what this EMM command does, but it definitely exists on 288-01 cards.

3.2.11- EMM command $23

I don't yet know what this command does, but it does do a lookup on type $0C data, which I believe is related to the spending limit

3.2.12- EMM command $24

I don't know what this EMM command does, but it definitely exists on 288-01 cards.

3.2.13- EMM command $25

I don't know what this EMM command does. but it definitely exists on 288-01 cards.

3.2.14- EMM command $26

I don't know what this EMM command does, but it definitely exists on 288-01 cards.

3.2.15- EMM command $2F

I don't know what this EMM command does, but it definitely exists on 288-01 cards.

3.2.16- EMM command $30

I don't know what this EMM command does, but it definitely exists on 288-01 cards.

3.2.17- EMM command $31

I don't know what this EMM command does, but it definitely exists on 288-01 cards

3.2.18- EMM command $32

I don't know what this EMM command does, but it definitely exists on 288-01 cards.

3.2.19- EMM command $40

I've been told that this command is related to activation (and presumably, adding of type $08 data), but I don't know anything at all about its format, nor can I verify that it does, in fact, relate to activation in any way

3.2.20- EMM command $41

I don't know what this EMM command does, but it definitely exists on 288-01 cards.

3.2.21- EMM command $42

This command is used to change the public keys that are used to decrypt the $03 command data. It takes two data fields: One byte to tell it which key to change, and 8 bytes of new key data. The $42 command looks like this:

```
42                          ;EMM command
05                          ;Key ID byte (see below)
11 22 33 44 55 66 77 88     ;New key
```

Key ID byte: This byte is used to select which key is updated by the $42 command, as follows: $04=verify key, $05=key 0, $85=key 1.

3.2.22- EMM command $43

I don't know what this EMM command does, but it definitely exists on 288-01 cards.

3.2.23- EMM command $44

This command is used to write IRD info into the CAM. It's like the $61 command, but this command is more selective and also allows access to the IRD's box key. This is the key that's used to encrypt the data for the $13 command. The IRD's box key is stored in the $01 data type, in a field that's not dumped by the 21-01 command. This command includes $1D bytes of data and looks like this:

```
44                         ;EMM command
nn                         ;Control byte (see below)
01 02 03 04 05 06 07 08    ;New IRD data ($1C bytes)
09 0A 0B 0C 0D 0E 0F 10
11 12 13 14 15 16 17 18
19 1A 1B 1C
```

Control byte: it seems as though this byte is used to control which 4-byte blocks of the $1C byte data block are actually written to the IRD. The breakdown of the bits appears to me to be as follows:

Bit #	Data block to be written
0	Bytes 15..18 and 19..1C (Suspect IRD key)
1	Bytes 01..04 (Suspect IRD serial #)
2	Bytes 05..08 (Suspect bytes 1..4 of IRD version info)
3	Bytes 09..0C (Suspect bytes 5..8 of IRD version info)
4	Bytes 0D..10 (Suspect bytes 9..12 of IRD version info)
5	Bytes 11..14 (Suspect bytes 13..16 of IRD version info)
6	Bytes 15..18 (Suspect first four bytes of IRD key)
7	Bytes 19..1C (Suspect last four bytes of IRD key)

3.2.24- EMM command $45

I don't know what this EMM command does, but it definitely exists on 288-01 cards.

3.2.25- EMM command $50

I don't know what this EMM command does, but it definitely exists on 288-01 cards.

3.2.26- EMM command $51

I don't know what this EMM command does, but it definitely exists on 288-01 cards.

3.2.27- EMM command $52

I don't know what this EMM command does, but it definitely exists on 288-01 cards.

3.2.28- EMM command $53

I don't know what this EMM command does, but it definitely exists on 288-01 cards.

3.2.29- EMM command $54

I don't know what this EMM command does, but it definitely exists on 288-01 cards.

3.2.30- EMM command $60

I don't know what this EMM command does, but it definitely exists on 288-01 cards.

3.2.31- EMM command $61

I don't know what this EMM command does, but it definitely exists on 288-01 cards.

3.2.32- EMM command $62

I don't know what this EMM command does, but it definitely exists on 288-01 cards.

3.2.33- EMM command $63

I don't know what this EMM command does, but it definitely exists on 288-01 cards.

3.2.34- EMM command $64

I don't know what this EMM command does, but it definitely exists on 288-01 cards.

3.2.35- EMM command $80

I don't know what this EMM command does, but it definitely exists on 288-01 cards.

3.2.36- EMM command $81

I don't know what this EMM command does, but it definitely exists on 288-01 cards.

3.2.37- EMM command $82

I don't know what this EMM command does, but it definitely exists on 288-01 cards

3.2.38- EMM command $83

I don't know what this EMM command does, but it definitely exists on 288-01 cards.

3.2.39- EMM command $84

I don't know what this EMM command does, but it definitely exists on 288-01 cards.

3.2.40- EMM command $85

I don't know what this EMM command does, but it definitely exists on 288-01 cards

3.2.41- EMM command $86

I don't know what this EMM command does, but it definitely exists on 288-01 cards.

3.2.42- EMM command $F0 (288-01)/EMM command $F1 (288-02)

This is the most powerful EMM command of all. It allows EchoStar/Nagra to actually upload a code fragment to the card, and have that code fragment executed from RAM. It is the existence of this command that makes a stable, widely available EchoStar 3M card unlikely: If this command is left enabled in the card, E*/Nagra can use it to detect the foreign 3M code and attack the card. If this command is disabled, E*/Nagra can use it to make changes to the public keys for command $03, so even though the card wants to work, it won't be able to. This is a variable-length command, so it will likely be the last command in an EMM message. The format is as follows:

```
288-01 card: F0                      ;EMM command
     <16CF54B object code here>      ;Code to be executed
     <JMP to EMM cleanup routine>    ;Last instruction is likely either
                                     ; a JMP to the routine that deals
                                     ; with EMM end-of-processing, or
                                     ; a JMP to $4000 to reset the card

288-02 card: F3                      ;EMM command
     <16CF54B object code here>      ;Code to be executed
     <JMP to EMM cleanup routine>    ;Last instruction is likely either
                                     ; a JMP to the routine that deals
                                     ; with EMM end-of-processing, or
                                     ; a JMP to $4000 to reset the card
```

The reason the 288-01 and 288-02 cards have different EMM commands for the same function is because the 288-01 cards and 288-02 cards have different ROM code inside them. The code is very similar (see section 5, inside the EchoStar card), but because it's not identical, a mechanism is needed to ensure that code fragments that are calling routines in the 288-01 ROM won't be executed on a 288-02 card, and vice-versa. When the code is executed, the first instruction is always at address $81 in RAM.

==

4.1- Data type list

The EchoStar cards manage subscription, PPV, and encryption information using what is essentially a crude flash file system. Their data engine can deal with 12 known fixed-length data types and 2 known variable-length data types. These data items are all stored in a single, large area of the EEPROM which is given over entirely to this purpose. As new data items are added, they may be added to the end of existing data or stuck in a hole that was created by the elimination of another data entry. Commands $20 and $21 are used to determine the existence and values, respectively, of the data items stored in the card's EEPROM. In this section, I'll be describing the functions and structures of the data items that I know about.

There are at least 14 types of data that can be polled from the card, each having a unique data structure. Note that not all data for a given data type is necessarily available to a 21-01 command. Also, it appears as though the first two bytes of data for fixed-length data types are overhead and not actually returned in a 21-01 command. Known data types are as follows:

```
Type Len Avail Data description
---- --- ----- -----------------------------------------
$01  $28 $1E   Married IRD info. Includes such information as the
               married IRD's serial number, subscriber's ZIP code,
               (for sports blackouts), subscriber's time zone, IRD
               box key, etc. Note: Contains hidden info (box key)

$02  $06 $04   unknown
$03  $0E ??    unknown
$04  $0C ??    unknown
$05  $0E ??    unknown
$06  $39 $26   Public services info. Includes blackout information
               and public decryption keys for $03 commands (on 288-01
               cards, anyway). Note: Contains hidden info (decrypt
               keys)
$07  $7A ??    unknown
$08  $1E $1C   Valid channel services (enables channels in the program
               guide, allows the IRD to decide on its own if a channel
               is subscribed, and if not, to display the "this channel
               is not subscribed" dialog)
$09  $27 ??    unknown
$0A  $2A ??    unknown
$0B  $24 $22   Valid PPV services (enables the IRD to decide on its own
               whether or not to display the "this PPV has not been
               purchased" dialog)
$0C  $13 $11   unknown, suspect spending limit info
$10  var var   unknown, suspect pending purchase info
$11  var var   Purchased PPV info.
```

Details on each data type are included below.

4.2.1- Data type $01

This is information on the IRD that the CAM is married to. It includes such information as the married IRD's serial number, the ZIP code and time zone where the subscriber is located, information on the IRD's software revision level, and the IRD box keys. The structure of the $01 data type is as follows:

```
00             ;Misc. data
01             ;Misc. data
00             ;Misc. data
01             ;IRD status byte (see below)
00             ;Misc. data
01 38 F0       ;ZIP code in HEX (80112)
E4             ;Time zone (see below)
00             ;Misc. data
33 22 11 00    ;IRD # (00112233 = 00 0112 2867 xx)
```

```
31 30 30 42 42 54 45 41  ;IRD bootstrap revision "110BBTEA"
35 32 30 50 31 31 44 4E  ;IRD firmware revision. "520P11DN"
11 22 33 44 55 66 77 88  ;IRD box key (hidden)
```

IRD status byte If bit 7 of this byte is set, the IRD is not activated (ie., subscribed)

Time zone encoding: The time zone is expressed as an 8-bit signed integer which represents the number of 15-minute ticks need to be added or subtracted from GMT Thus, 00 is GMT, FF is GMT minus 15 minutes, etc. The following table details CONUS time zones:

Time Zone	Offset (hours)	Offset (ticks)	Time zone byte
PST	GMT-8	GMT-32	E0
MST	GMT-7	GMT-28	E4
CST	GMT-6	GMT-24	E8
EST	GMT-5	GMT-20	EC

4.2.2- Data type $02

I don't know what kind of information this command is requesting from the CAM. I do know, however, that the information appears to be the same for both virgin and subscribed cards. Here's what the data looks like:

```
01 01 00 00          ;Response data
```

4.2.3- Data type $03

I have no information about this data type at this time, other than its length is $0E bytes.

4.2.4- Data type $04

I have no information about this data type at this time, other than its length is $0C bytes.

4.2.5- Data type $05

I have no information about this data type at this time, other than its length is $0E bytes.

4.2.6- Data type $06

This data type contains information about programming in general. Included in this is information about blackouts (which seems to be a bit mapped field), as well as (on 288-01 cards, anyway) the public keys used to decrypt the $03 commands (even though they're not returned for a 21-06 command). This data has the following structure:

```
00                    ;Element number (more than 1 21-06 item
                      ; is available)
00 00 00              ;Misc. data
31 3D F3              ;Data header, it seems. These 3 bytes
                      ; always seem to be the same
FF FF FF FF FF FF FF FF   ;Blackout info
```

```
FF FF FC FF FF 00 FF FF   ;Blackout info
00 00 00 00 00 00 00 00   ;Blackout info
00 00 00 00 00 00 00      ;Blackout info
00 00                     ;Misc. data
00 00 00 00 00 00 00 00   ;Key 0 (in element 0 only)
11 11 11 11 11 11 11 11   ;Key 1 (in element 0 only)
```

4.2.7- Data type $07

I have no information about this data type at this time. other than its length is $7A bytes.

4.2.8- Data type $08

This data type contains information relating to the standard channel services which the is allowed to receive and/or show in its EPG. The data in this structure doesn't actually control access to the channels...instead, it's used by the IRD to decide which channels are shown in the EPG, whether they're shown in red, and whether it should immediately display the "this is a channel to which you have not subscribed" dialog box when you change channels. In a typical subscribed card, this data type will have many entries. An example that would allow viewing of all available channels in the program guide is:

```
01 01 00 00            ;Misc. data
00 00                  ;Misc. data
00 00                  ;Misc. data
00 00 00 00 00 00      ;Misc. data
4C 21 4C 21            ;Misc. data
00 00                  ;Min. tier authorized by this packet
7F FF                  ;Max tier authorized by this packet
80 00                  ;Misc. data
FF 00                  ;Misc. data
FF 00                  ;Misc. data
```

Note that presenting this data to the IRD is not enough to receive programming: the IRD will still depend on the CAM to provide the video decryption keys, and if the CAM knows that a channel for which the IRD is requesting information isn't subscribed, it won't return the proper keys.

4.2.9- Data type $09

I have no information about this data type at this time, other than its length is $27 bytes.

4.2.10- Data type $0A

I have no information about this data type at this time, other than its length is $2A bytes.

4.2.11- Data type $0B

This data type is much like the $08 data type, except that it relates to PPV events rather than normal channels. I don't really understand the structure of this data type, though...it appears to possibly contain a bit-mapped data indicating which PPV channels are allowed, but it's tough to say. A type $0B data item that authorizes all PPVs would look like this:

```
01 01 00 00        ;Misc. data
7F FF              ;Misc. data
7F FF              ;Misc. data
7F FF              ;Misc. data
7F FF              ;Misc. data
7F FF              ;Misc. data
FF 00              ;Misc. data
00 7F              ;Misc. data
00 FF              ;Misc. data
00 00              ;Misc. data
7F FF              ;Misc. data
7F FF              ;Misc. data
7F FF              ;Misc. data
7F FF              ;Misc. data
7F FF              ;Misc. data
7F FF              ;Misc. data
```

As you can see, I have no idea at this point what each of the values in the data response represents. The organization that I've presented here is entirely supposition on my part.

Note that presenting this packet to the IRD is not enough to receive programming: the IRD will still depend on the CAM to provide the video and audio decryption keys, and if the CAM knows that a PPV for which the IRD is requesting information isn't authorized, it won't return the proper keys

4.2.12- Data type $0C

This data type seems to be something to do with the subscriber's spending limit. The returned data differs between virgin (unsubscribed) and subscribed cards. The data from a virgin card looks like this:

```
01 01              ;Misc data
00 01              ;Misc. data
12 34              ;Misc. data
00 00 00 00 00 00  ;Misc. data
AA 00 00 BB        ;Misc data
```

4.2.13- Data type $10

I have no information about this data type at this time I suspect that it has something to do with pending PPV purchases. This data type is variable-length.

4 2.14- Data type $11

This data type contains information about purchased PPV events. The individual elements are variable-length, and seem to be able to contain anything from the date and time the event was purchased to the name of the event to the channel on which the event was broadcast. There's no set pattern I can see to the data in this data type, other than the event name seems to be set between 00 bytes.

==

5.1- The MCU core

The microcontroller in the EchoStar smartcards is an ST Micro (SGS Thomson at the time they were first released) ST16CF54B. It's basically a 6805, but with a couple of additional features...it has a cryptoprocessor built in which allows high-speed (well, relatively so) modular multiplication and exponentiation, and it has one additional instruction: TSA (transfer stack to A: Opcode $9E). The cryptoprocessor allows it to do some kinds of math pretty quickly, and the TSA instruction allows the firmware to easily figure out the value of the stack pointer. The card has 8K of library ROM that came from SGS, 16K of user ROM which was written by Nagra, 4K of EEPROM space that can be used to hold additional code or data, and 512 bytes of RAM. If you want to understand how the firmware works, I strongly suggest learning about the 6805 (because information on it is plentiful, whereas information on the 16CF54B is sparse at best). Here's a little bit of information about the internal registers of the 16CF54B:

Addr	Description
$00	I/O register
$01	Security register
$03	EEPROM control register
$04	Test status register
$06	Random number generator high byte
$07	Random number generator low byte
$0A	Cryptoprocessor I/O byte 1
$0B	Cryptoprocessor I/O byte 2
$0C	Cryptoprocessor control byte 1
$0D	Cryptoprocessor control byte 2

By default, the stack occupies the RAM space from $40 through $7F (the RSP instruction resets the stack pointer to $7F). If the stack overflows, it will wrap around, destroying the least-recently-used data on the stack. However, there's no rule that says that the lower end of the stack can't be used to store variables: The firmware just has to be written so that JSRs and interrupts never nest deeply enough to destroy the variables that are stored in stack space
The processor's reset and interrupt vectors are as follows

Addr	Description
$4000	Reset vector. Execution starts here on reset.
$4008	SWI vector. SWI interrupt begins execution here.
$4010	NMI vector. Security interrupt begins execution here.
$4018	INT interrupt vector Maskable interrupt begins execution here

Note that ST Micro has "recommended code" that they like to see placed at the vector locations. If you get hold of the SECA ROM dump that's floating around the 'net, you'll see code that's pretty much straight out of the ST Micro manual at $4000..$401F.

The processor's memory map:

Addr	Description
$0000-$001F	Peripheral registers
$0020-$003F	General-purpose RAM
$0040-$007F	Stack

$0080-$021F General-purpose RAM
$2000-$3FFF ST Micro library ROM
$4000-$7FFF User ROM (Echostar main code)
$E000-$EFFF EEPROM

5.2- 288-01 vs. 288-02

There are two known versions of the EchoStar smartcard. They can be identified by a number printed on the back of the card (the side with the contacts) in fine print. If you hold the card so that the contacts are at the top, near the bottom right corner of the card, you should see either 288-01 or 288-02.
The 288-01 cards were the first release. Their code was quite buggy, and had a large amount of patched code in the EEPROM (about $900 bytes). Most of the EchoStar ROM dumps you'll see on the 'net came from these cards.
The 288-02 cards have basically the same code, but with all of the fixes that were in the 288-01 EEPROM area integrated back into the ROM.
Although the code in the two cards does the same thing (and in many cases is identical), because of the differences in lengths of some of the routines, the ROMs are not identical. This is the reason for separate EMM "execute code" commands.

5.3- Other cards based on the same code

Because EchoStar provides the receivers, CAMs, antennae, and so forth for lots of satellite services worldwide, there are lots of cards that are based on the same code as the EchoStar cards. Like the EchoStar cards, these cards have a 288-xx number printed on them, and the code within is based on the EchoStar 288-02 cards
These cards are used by ExpressVu, SkyVista, and several other services. Check your cards...you never know what you might find.

===

6.1- Change log

Here's where you'll find a list of changes that have been made to this FAQ since its creation. Changes may include addition of new data, correction of errant data, or deletion of errant data.

Release date	Changes
10/15/99	Initial release.

6.2- Coming soon

Here's where you'll find a list of things that I'll be adding to the FAQ as I get the time and/or come across the info.

-Discussion of the $03 encryption algorithm
-Discussion of the $00/$01 encryption algorithm
-Discussion of the $02 encryption algorithm
-Discussion of the $30/$31 packets
-Discussion of the $99 packets
-Glossary of EchoStar/DVB terms

6.3- Contacting me

I realize that I'm not the ultimate authority on the EchoStar system, and that there's a lot of information that I don't yet have. If you have any corrections to the information in this FAQ, or if you have information you'd like to add to it, email it to me at:

stuntguy@dishplex.com

Let me know how you want to be credited when you send the information.

-S