Money Controls

Money Controls (UK) Limited
Coin House, New Coin Street, Royton, Oldham, Lancashire. OL2 6JZ
Tel: +44 (0) 161 678 0111 Fax: +44 (0) 161 633 8567

DES Encryption for Hoppers

ccTalk Protocol

Issue 2.1

28th January 2011

Revision History

Issue	Date	Comments
1.0	06-03-09	First draft.
1.1	16-07-09	New options for [command level] in ‘Request encryption support’ command
1.2	18-03-10	Addition of new section : Implementation Rules
2.0	05-10-10	General Release
2.1	28-01-11	Added paragraph on the ‘responsibility of the gaming machine manufacturer’ ccTalk revision level may be 4.6 or greater

1 Contents

2 Permissions

The security for hopper DES encryption lies in a public domain algorithm with the ‘key’ being the secret rather than the algorithm. For this reason this document is not as sensitive as those published for the ccTalk protocol layer encryption and not subject to the same restrictions. However, it is good practice to limit the number of copies circulated and to make sure that only the people who need to see it do so.

3 Overview

The current payout security key for ccTalk hoppers uses a proprietary Money Controls’ algorithm. This algorithm acts on a 64-bit block of data. It is proposed that this algorithm is changed to the industry-standard DES algorithm for better security. As this also acts on 64-bit data blocks the command sequence for dispensing coins is identical. The only necessary software changes are around key discovery and the change to the encryption algorithm itself. Additional support is provided for changing DES keys and verifying payout.

Each hopper will be manufactured with a unique DES key. Unless the DES key is known then no payout from the hopper will be possible. This DES key can be used for the lifetime of the peripheral taking into account that if it is cracked on one hopper it does not compromise another hopper. The time to crack a single DES key is estimated at just less than a day given vast processing capability on thousands of CPUs. In case this becomes economically attractive in future, a command has been provided that allows the game machine to rotate the DES key every e.g. 24 hours. This means a fraudster returning to a game machine the following day with a cracked key will find that it doesn't work; the key has moved on !

The single DES algorithm uses a 64-bit key (only 56 bits are actually used, the rest are parity) giving over 10,000,000,000,000,000 possibilities. It would take too long to manually enter a DES key into a game machine using simple up / down / left / right keys. Consideration also has to be given to servicing and spares which require a quick method of replacing a faulty hopper. Therefore we have introduced the concept of a 'trusted key exchange mode'. In this mode, which is a non-operating mode of the hopper, it will be possible for the game machine to request the security keys of the hopper. Once obtained, the keys need not be transferred again until the hopper is replaced. How trusted key exchange mode is entered into and the security aspects of that are discussed later.

Even though each hopper contains a unique DES key, they are all functionally identical and interchangeable for spares purposes. No key values need to be printed on the installation label.

As the DES algorithm is fairly lengthy in terms of processing cycles, and as this could impact on game performance, the DES calculation is only needed intermittently and not for any polling operations. On hoppers, the DES calculation only needs to be done for each dispense command of between 1 and 255 coins. Any performance penalty of introducing DES on older platforms should therefore be minimal.

For some additional security it is possible to combine the DES encryption algorithm with the ccTalk protocol layer encryption which uses a 6-digit key to encrypt each ccTalk command packet. This 6-digit key can also be rotated by the game machine every few seconds to guard against replay attacks. Protocol encryption has only previously been used on bank note validators (BNVs) and is often referred to as ***BNV encryption***. It is proposed to extend BNV encryption to hoppers in applications that require the additional security.

Peripherals that support DES encryption will return a ccTalk comms revision (header 004) of at least 4.6 and will have implemented the additional commands described in this document.

4 Payout Command Sequence

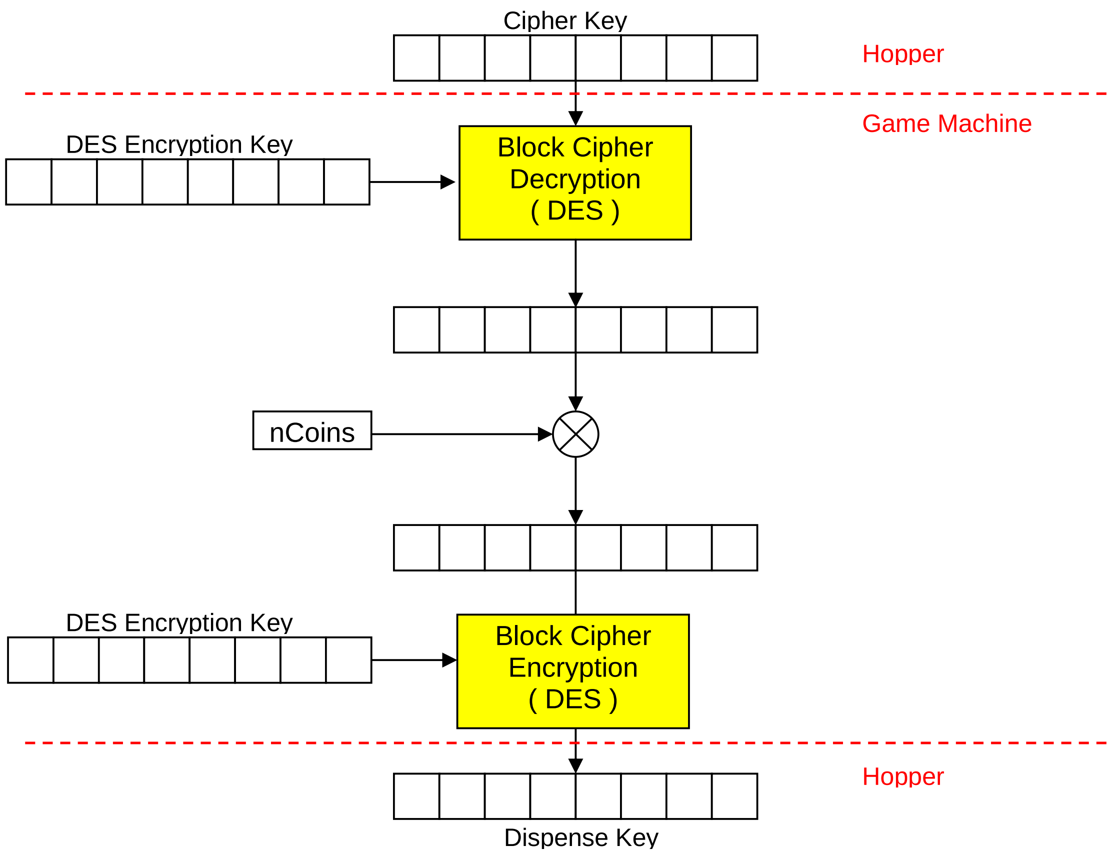
The ccTalk command sequence to dispense coins from a DES-encrypted hopper is no different to previous hoppers.

Clear any errors
|
Enable the hopper
|
Request a cipher key
|
Apply the encryption algorithm
|
Send a dispense coin command
|
Poll the hopper status
|
Test the hopper if an error occurs

The actual ccTalk commands that do this are…

Header 001, Reset device
|
Header 164, Enable hopper
|
Header 160, Request cipher key
|
< DES Decryption / Encryption >
|
Header 167, Dispense hopper coins
|
Header 166, Request hopper status
(Header 109, Request encrypted hopper status)
|
Header 163, Test hopper

5 DES Encryption Algorithm



The ‘Request cipher key’ command returns 64-bits of random data and this changes after each dispense operation so that a hopper replay attack cannot be performed

In the examples which follow, the entire ccTalk command packet is shown. The hopper is on address 3. The byte values between square brackets are shown in hexadecimal.

The examples shown do not use protocol level encryption so we use source addresses and 8-bit checksums rather than 16-bit CRC checksums.

TX is data from the game machine to the peripheral.
RX is data from the peripheral to the game machine.

```
Request cipher key
TX : [ 03 ][ 00 ][ 01 ][ A0 ][ 5C ]
RX : [ 01 ][ 08 ][ 03 ][ 00 ]
     [ B6 ][ CE ][ 58 ][ C5 ][ 0B ][ 77 ][ CD ][ 64 ][ A0 ]
```

TX Decode :
Destination address = 3
No. of data bytes = 0
Source address = 1
Command 0xA0 = 160 decimal = Request cipher key
8-bit checksum = 0x5C

RX Decode :
Destination address = 1
No. of data bytes = 8
Source address = 3
Cipher key = B6 CE 58 C5 0B 77 CD 64
8-bit checksum = 0xA0

Assume the DES key in our example is the 64-bit number 0xFEDCBA9876543210

The DES algorithm takes an array of 8 bytes which represent the key. The first byte in our example is the LSB byte or [10], followed by [32]… up to [FE].

Running this through the industry-standard DES decryption algorithm…

```
Cipher     [ B6 ] [ CE ] [ 58 ] [ C5 ] [ 0B ] [ 77 ] [ CD ] [ 64 ]
DES key    [ 10 ] [ 32 ] [ 54 ] [ 76 ] [ 98 ] [ BA ] [ DC ] [ FE ]
Decrypted  [ 01 ] [ 02 ] [ 03 ] [ 04 ] [ 05 ] [ 06 ] [ 07 ] [ 08 ]
```

At this point we need to know how many coins to dispense as this information forms part of the encrypted data.

Suppose we want to dispense 5 coins. We exclusive-OR each of the decrypted cipher key bytes with the dispense coin number.

```
     [ 01 ] [ 02 ] [ 03 ] [ 04 ] [ 05 ] [ 06 ] [ 07 ] [ 08 ]
XOR  [ 05 ] [ 05 ] [ 05 ] [ 05 ] [ 05 ] [ 05 ] [ 05 ] [ 05 ]
=    [ 04 ] [ 07 ] [ 06 ] [ 01 ] [ 00 ] [ 03 ] [ 02 ] [ 0D ]
```

Remember the coins to dispense can be any number form 1 (0x01) to 255 (0xFF) but it is the same exclusive-OR process as above.

We now run this through the industry-standard DES encryption algorithm…

```
From XOR   [ 04 ] [ 07 ] [ 06 ] [ 01 ] [ 00 ] [ 03 ] [ 02 ] [ 0D ]
DES key    [ 10 ] [ 32 ] [ 54 ] [ 76 ] [ 98 ] [ BA ] [ DC ] [ FE ]
Encrypted  [ 04 ] [ DD ] [ E7 ] [ A2 ] [ 97 ] [ 90 ] [ 31 ] [ 7E ]
```

So we now have 8 encrypted bytes which are sent to the hopper in the ‘Dispense hopper coins’ command. We also send the number of coins to pay as a 9th data byte in plaintext (unencrypted). Note that any illegal manipulation of the ‘coins to pay’ byte will fail because of the previous embedding process.

The format of the ‘Dispense hopper coins’ command is identical to previous hoppers.

```
Dispense hopper coins
TX : [ 03 ] [ 09 ] [ 01 ] [ A7 ]
     [ 04 ] [ DD ] [ E7 ] [ A2 ] [ 97 ] [ 90 ] [ 31 ] [ 7E ] [ 05 ] [ 07 ]
RX : [ 01 ] [ 01 ] [ 03 ] [ 00 ] [ 01 ] [ FA ]
```

TX Decode :
Destination address = 3
No. of data bytes = 9
Source address = 1
Command 0xA7 = 167 decimal = Dispense hopper coins
Encrypted data = 04 DD E7 A2 97 90 31 7E
No. of coins to pay = 05
8-bit checksum = 0x07

RX Decode :
Destination address = 1
No. of data bytes = 1
Source address = 3
Event counter = 1 (the hopper returns the new event counter for each new dispense attempt)
8-bit checksum = 0xFA

The hopper decodes this data, validates it, and dispenses 5 coins.

5.1 DES Technical Information

5.1.1 Parity

Although the DES key is 64 bits long, and passed as an array of 8 bytes, only 56 bits of this are actually used in the algorithm. 8 bits of the array are allocated to parity bits. The hopper ignores these parity bits and they can be set to anything. The parity bit here is bit 0 of each array byte.

The following keys are therefore identical i.e. produce the same result when run through the DES encryption algorithm.

0xFEDCBA9876543210
0xFFDDBB9977553311

In other words, each byte of the DES key array may be odd or even without consequence.

Note that when changing key with the ‘Switch encryption key’ command, the old key must match exactly i.e. parity bits are not ignored in the comparison.

5.1.2 IV

DES algorithms often refer to the IV or initialisation vector. This is only used when encrypting multiple blocks in one operation and block chaining is used. For hoppers we are only encrypting one or two blocks and are working in ECB mode (Electronic Code Book). For hoppers there is no IV so we do not define one. If you are unsure how it is being used then set it to all zeros (an array of 8 null bytes).

5.1.3 Symmetry

DES is a symmetrical encryption algorithm so that the key to encrypt a data block is the same key that decrypts the data block.

5.1.4 Weak Keys

It is important that when the game machine changes the DES key in the peripheral, weak keys are avoided as they are easier to crack.

DES has 4 documented weak keys and 6 semi-weak key pairs.

Weak keys :
0101010101010101
FEFEFEFEFEFEFEFE
E0E0E0E0F1F1F1F1
1F1F1F1F0E0E0E0E

Most DES key generator libraries will automatically avoid all weak and semi-weak keys.

5.2 Microsoft API

Cryptographic services are built into many high-level programming languages such as Microsoft Visual Basic.Net. Here is an example of encrypting data with DES.

```
Dim keyDES As New DESCryptoServiceProvider()
keyDES.Key = key64
keyDES.Mode = CipherMode.ECB
keyDES.Padding = PaddingMode.Zeros
Dim ms As MemoryStream = New MemoryStream
Dim xStream As CryptoStream = New CryptoStream(ms, keyDES.CreateEncryptor(),
CryptoStreamMode.Write)
Dim sw As New BinaryWriter(xStream)
sw.Write(dataBlock64)
sw.Close()
xStream.Close()
Dim encryptedData() As Byte = ms.ToArray()
ms.Close()
keyDES.Clear()
Return encryptedData
```

5.3 Further Information

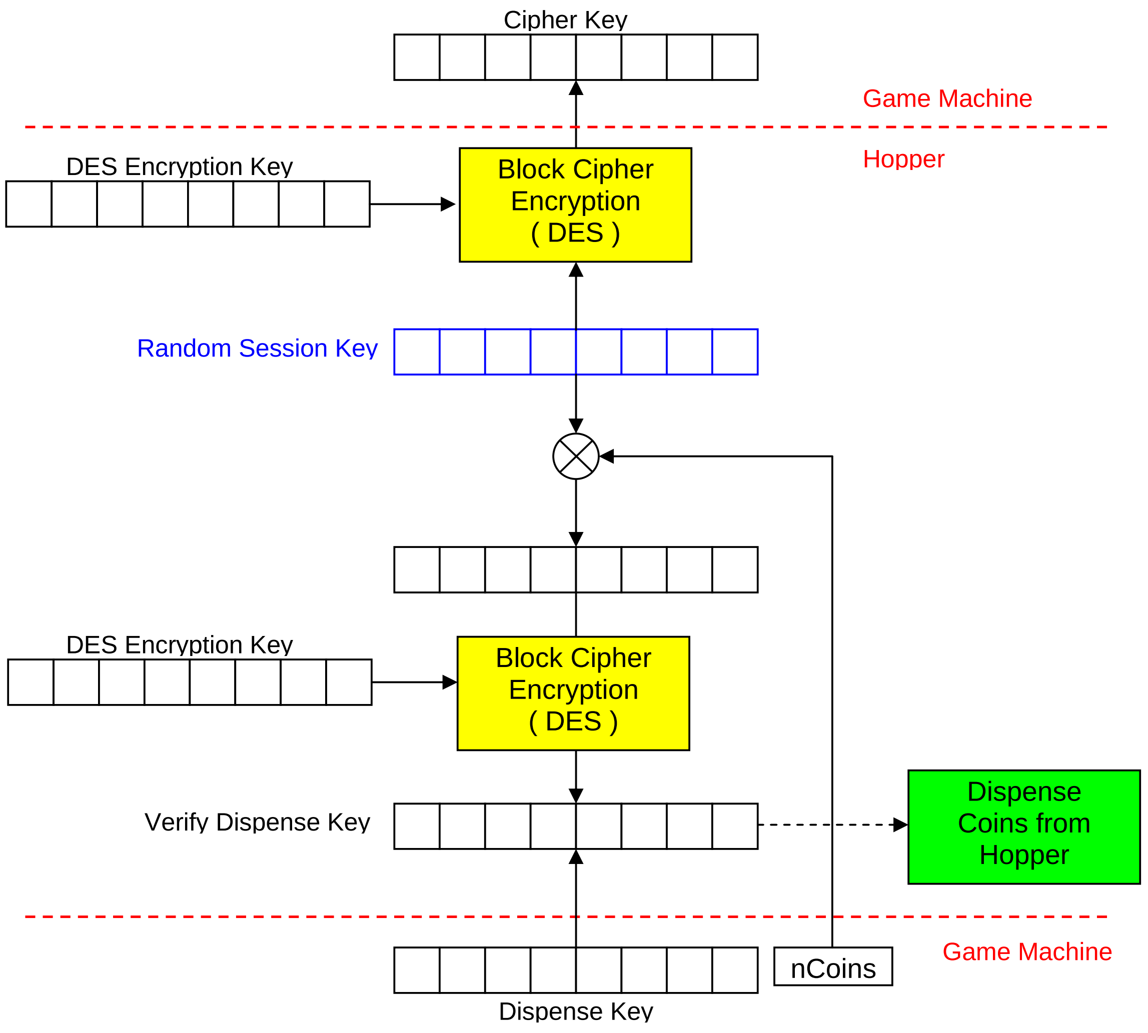
An excellent primer on DES encryption may be found on Wikipedia.
http://en.wikipedia.org/wiki/Data_Encryption_Standard
http://en.wikipedia.org/wiki/Block_cipher_modes_of_operation
http://en.wikipedia.org/wiki/Weak_key

The DES algorithm itself cannot be obtained from Money Controls but it has been widely published and is available in many programming languages and in libraries optimised for different processors.

Most microcontrollers today should be able to encrypt a DES block in well under 10ms.

6 Inside the Hopper

The hopper dispense algorithm only needs to be implemented in the forward direction (encryption not decryption) and this helps to improve performance where the key schedule is pre-calculated on small microcontrollers.



The random session key is generated by the hopper for each new dispense command to prevent simple replay attacks. It can mix in data from random or pseudo-random sources and run old keys through the DES engine to increase entropy.

Measures should be taken to ensure the session key is not the same every time a hopper is powered-up or reset.

7 Trusted Key Exchange Mode

A new command has been introduced into issue 4.6 of the ccTalk generic specification. This command allows a game machine to identify the level of encryption that exists on a peripheral and in certain conditions actually read the encryption keys. This command works unencrypted regardless of whether BNV protocol encryption is in use; previously only encrypted commands could be sent to an encrypted peripheral preventing any self-configuration of the gaming board interface.

The new command is as follows…

Header 111 : Request encryption support

```
Transmitted data : [ AA ] [ 55 ] [ 00 ] [ 00 ] [ 55 ] [ AA ]
Received data :    [ protocol level ] [ command level ]
                   [ protocol key size ]
                   [ command key size ] [ command block size ]
                   [ trusted mode ]
                   [ BNV2 | BNV1 ] [ BNV4 | BNV3 ] [ BNV6 | BNV5 ]
                   [ DES1 ] [ DES2 ] [ DES3 ] [ DES4 ]
                   [ DES5 ] [ DES6 ] [ DES7 ] [ DES8 ]
```

The 6 data bytes transmitted (0xAA, 0x55, 0x00, 0x00, 0x55, 0xAA) are a validation signature for this command. They never change but must be transmitted as shown.

The receive data consists of the following information.

[protocol level]
0 – no encryption
1 – ccTalk Serial Protocol Encryption Standard 1.2

This refers to the protocol layer encryption and is nothing to do with the DES key on hoppers. The '1.2' refers to the latest implementation of this algorithm at the time of publication of this document. It has been the standard encryption algorithm on ccTalk bill validators for many years.

Documentation on the ccTalk protocol layer encryption can be obtained from Money Controls on request but it is currently strictly controlled and subject to NDA. Manufacturers of hopper peripherals do not have to support protocol layer encryption and can just use DES instead. The high security level obtained with DES alone should be perfectly adequate.

[command level]
0 – no encryption
11 – Serial Hopper Encryption Standard CMF1-1 (SCH2 L1 encryption)
12 – Serial Hopper Encryption Standard CMF1-2 (SCH3 L2 encryption)
13 – Serial Hopper Encryption Standard CMF1-3 (SCH3E L3 encryption)
21 – Serial Hopper Encryption Standard CMF2-1 (Combi encryption)
101 – DES Encryption
102 – AES Encryption (future support only)
103 – Triple DES Encryption (future support only)

On hoppers this refers to the security available on the hopper dispense command. One of the options is DES encryption as put forward by this document (level = 101).

[protocol key size]

This is the size in bits of the protocol layer encryption key. The BNV algorithm uses a 6 decimal digit encryption key 000000 to 999999. It is encoded in BCD format in 3 bytes or 24 bits. But although we will return ‘24’ by convention the actual key space is a lot less than this due to the decimal sub-group and a slight key reduction on 3 byte packets due to the way it has been implemented.

[command key size]

This is the size in bits of the command level encryption key. For single DES we will return ‘64’. The value of 0 will be reserved for 256 bits if required in future encryption algorithms.

[command block size]

Block sizes of 64 bit and 128 bits are common for symmetrical encryption algorithms. On hoppers with DES the block size is ‘64’. This relates directly to the length of the cipher key returned by ccTalk command header 160, ‘Request cipher key’.

[trusted mode]
0 – normal operating mode
255 – trusted key exchange mode

The game machine should treat any value other than 255 as normal operating mode. In trusted key exchange mode the protocol level BNV key and the DES encryption key will follow. ***In normal operating mode the keys will still be returned as data bytes but they will all be zero***.

This is typical packet exchange for a hopper to show the format of the keys.

```
Request encryption support
TX :  [ 03 ] [ 06 ] [ 01 ] [ 6F ]
      [ AA ] [ 55 ] [ 00 ] [ 00 ] [ 55 ] [ AA ]
      [ 89 ]
RX :  [ 01 ] [ 11 ] [ 03 ] [ 00 ]
      [ 00 ] [ 65 ] [ 18 ] [ 40 ] [ 40 ] [ FF ]
      [ 00 ] [ 00 ] [ 00 ]
      [ 10 ] [ 32 ] [ 54 ] [ 76 ] [ 98 ] [ BA ] [ DC ] [ FE ]
      [ B7 ]
```

Note that this command is always sent unencrypted with 8-bit checksums, even if the peripheral uses protocol level encryption.

TX Decode :
Destination address = 3
No. of data bytes = 6
Source address = 1
Command 0x6F = 111 decimal = Request encryption support
8-bit checksum = 0x89

RX Decode :
Destination address = 1
No. of data bytes = 17
Source address = 3
Protocol level = 0x00 = no encryption
Command level = 0x65 = 101 decimal = DES Encryption
Protocol key size = 0x18 = 24 decimal (bits)
Command key size = 0x40 = 64 decimal (bits)
Command block size = 0x40 = 64 decimal (bits)
Trusted mode = 0xFF = 255 decimal = yes
BNV key = 0x000000
DES key = 0xFEDCBA9876543210
8-bit checksum = 0xB7

Note that ccTalk message packets are little-endian – we return LSB bytes first and MSB bytes last. So the first DES key byte is returned first, in this case [10], and the last DES key byte last, in this case [FE]. The order it is written in or displayed is fairly arbitrary. The important point is that when you call the DES library, the first byte of the key array will be 0x10.

When not in trusted mode the following information is returned.

```
TX :  [ 03 ] [ 06 ] [ 01 ] [ 6F ]
      [ AA ] [ 55 ] [ 00 ] [ 00 ] [ 55 ] [ AA ]
      [ 89 ]
RX :  [ 01 ] [ 11 ] [ 03 ] [ 00 ]
      [ 00 ] [ 65 ] [ 18 ] [ 40 ] [ 40 ] [ 00 ]
      [ 00 ] [ 00 ] [ 00 ]
      [ 00 ] [ 00 ] [ 00 ] [ 00 ] [ 00 ] [ 00 ] [ 00 ] [ 00 ]
      [ EE ]
```

The [00] after the [40] indicates this is normal operating mode, not trusted mode.

The BNV keys and DES keys are still returned (17 data bytes in the packet), but the values are now cleared to zero to maintain secrecy.

If protocol level encryption is being used in trusted mode then the return data may look like this…

```
TX :  [ 03 ] [ 06 ] [ 01 ] [ 6F ]
      [ AA ] [ 55 ] [ 00 ] [ 00 ] [ 55 ] [ AA ]
      [ 89 ]
RX :  [ 01 ] [ 11 ] [ 03 ] [ 00 ]
      [ 01 ] [ 65 ] [ 18 ] [ 40 ] [ 40 ] [ FF ]
      [ 21 ] [ 43 ] [ 65 ]
      [ 10 ] [ 32 ] [ 54 ] [ 76 ] [ 98 ] [ BA ] [ DC ] [ FE ]
      [ ED ]
```

Protocol level = 0x01 = ccTalk Serial Protocol Encryption Standard 1.2

BNV key = 0x654321, usually written on labels as 123456

7.1 Entering Trusted Key Exchange Mode

For security it is essential that trusted key exchange mode can only be entered with direct physical access to the peripheral. If it is possible to switch to this mode using a serial command then there is the possibility that security could be defeated across the entire range of DES hoppers. Therefore trusted mode assumes the game cabinet has been opened up and physical contact can be made with the peripheral. This would occur anyway during installation of new hoppers and the servicing of old ones. Trusted key security then becomes the physical security of the game cabinet itself with regards to locks, hinges, steel covers and ventilation holes etc.

This document does not specify how trusted mode is entered; this can be left to the manufacturers of the peripheral concerned. However, it does warrant careful design consideration so as not to jeopardise the benefits of DES encryption on hoppers.

Some examples of how this mode can be engaged are given below.

7.1.1 DIP Switch

A DIP switch on the product could be used to enter trusted mode. Special attention should be paid to prevent a wire being inserted through an access point in the game cabinet and pushed against the switch. Perhaps the DIP switch is underneath the peripheral or behind a protective cover.

The DIP switch should only be read at power-up or after a software reset. If the DIP switch is left in the wrong position then any attempt to dispense coins should be refused.

7.1.2 PCB Jumper

A 2-pin PCB jumper could be used to enter trusted mode. A jumper is normally left attached, shorting the 2 terminals.

If the jumper is missing at power-up or after a software reset then trusted mode is entered. Any attempt to dispense coins will be refused if the jumper is missing. The jumper should not be in an exposed position on the peripheral to prevent it being lifted off remotely through 'keyhole surgery'.

7.1.3 Level Sensors

If the hopper has a level sensor then a simple lockout can be used to ensure that trusted mode can only be entered if the hopper is empty or nearly empty. This should not be used in isolation but can be combined with another method for better security. It is far less tempting to crack the keys on an empty hopper and be able to pay out nothing than a full one.

8 Switching DES Key

Header 110 : Switch encryption key

```
Transmitted data : [ old1 ] [ new1 ] [ old2 ] [ new2 ] [ old3 ] [ new3 ]
                   [ old4 ] [ new4 ]
                   [ old5 ] [ new5 ] [ old6 ] [ new6 ] [ old7 ] [ new7 ]
                   [ old8 ] [ new8 ]
Received data :    ACK
```

The old key is interleaved with the new key and sent to the peripheral. Before it is sent, both 8 byte blocks of transmit data are DES encrypted with the old key. The peripheral decrypts the data with the old key and if the old key matches then an immediate switch is made to the new key and an ACK returned. If the old key is incorrect then there is no reply. An ACK reply takes at least **100ms** to make guessing key values using this command totally impractical.

8.1 Key Verification

It is possible when a game starts to verify that the correct DES key is being used. This is useful after a hopper is replaced as it may be some time before a payout is attempted.

The 'Switch encryption key' command is used and the new key is made equal to the old key. The peripheral does not store any new data; it just confirms that the old key is correct.

```
Transmitted data : [ old1 ] [ old1 ] [ old2 ] [ old2 ]
                   [ old3 ] [ old3 ] [ old4 ] [ old4 ]
                   [ old5 ] [ old5 ] [ old6 ] [ old6 ]
                   [ old7 ] [ old7 ] [ old8 ] [ old8 ]
Received data :    ACK
```

For example, this is the verification of our default 0xFEDCBA9876543210 key.

```
TX :  [ 03 ] [ 10 ] [ 01 ] [ 6E ]
      [ B0 ] [ 2D ] [ F1 ] [ 0E ] [ 88 ] [ 6C ] [ FC ] [ FF ]
      [ 9B ] [ EC ] [ 58 ] [ F1 ] [ 89 ] [ 6C ] [ A6 ] [ 9C ]
      [ AC ]
RX :  [ 01 ] [ 00 ] [ 03 ] [ 00 ] [ FC ] = ACK
```

9 Payout Verification

After a dispense command, the hopper is polled with header 166, ‘Request hopper status’. To protect against illegal manipulation of the data bus, such as changing a hopper dispense ACK to a NAK in order to get a retry, there is a DES-protected hopper status command as well. This can be used after polling is completed or after a NAK was received from a ‘Dispense hopper coins’ command to verify what was actually paid out by the hopper. The reply combines data from the ‘Request hopper status’, ‘Test hopper’ and ‘Request payout high / low status’ commands with data in the same format.

Header 109 : Request encrypted hopper status

```
Transmitted data : [ challenge 1 ] [ challenge 2 ] [ challenge 3 ]
Received data :    [ CRC checksum LSB ] [ challenge 1 ] [ event counter ]
                   [ payout coins remaining ] [ last payout : coins paid ]
                   [ last payout : coins unpaid ] [ random1 ]
                   [ challenge 2 ]

                   [ challenge 3 ]
                   [ hopper status register 1 ] [ hopper status register 2 ]
                   [ hopper status register 3 ] [ level status ]
                   [ random 2 ] [ random 3 ]
                   [ CRC checksum MSB ]
```

For security, 3 challenge bytes are sent out by the game machine as plaintext. These are included in the reply from the hopper along with 3 random or undefined bytes. The hopper adds a CRC-16 checksum to the data using the same algorithm as the ccTalk packet structure when protocol level encryption is enabled. The CRC is calculated for the 14 bytes in between. The reply blocks are then DES encrypted by the hopper.

For example…

```
Request encrypted hopper status
TX :  [ 03 ] [ 03 ] [ 01 ] [ 6D ] [ EC ] [ AE ] [ BF ] [ 33 ]
RX :  [ 01 ] [ 10 ] [ 03 ] [ 00 ]
      [ E8 ] [ F8 ] [ 2E ] [ B4 ] [ 74 ] [ 7A ] [ 0E ] [ 35 ]
      [ 03 ] [ 4A ] [ 94 ] [ 10 ] [ 17 ] [ F4 ] [ B1 ] [ 9A ]
      [ B2 ]
```

TX Decode :
Destination address = 3
No. of data bytes = 3
Source address = 1
Command 0x6D = 109 decimal = Request encrypted hopper status
Challenge data = 0xEC 0xAE 0xBF
8-bit checksum = 0x33

RX Decode :
Destination address = 1
No. of data bytes = 16
Source address = 3
Packet 1 = **0xE8F82EB4747A0E35**
Packet 2 = **0x034A941017F4B19A**
8-bit checksum = 0xB2

Assume the DES key in our example is the 64-bit number 0xFEDCBA9876543210

Running this through the industry-standard DES decryption algorithm…

Block 1
```
Encrypted  [ E8 ] [ F8 ] [ 2E ] [ B4 ] [ 74 ] [ 7A ] [ 0E ] [ 35 ]
DES key    [ 10 ] [ 32 ] [ 54 ] [ 76 ] [ 98 ] [ BA ] [ DC ] [ FE ]
Decrypted  [ 9E ] [ EC ] [ 01 ] [ 00 ] [ 05 ] [ 00 ] [ 00 ] [ AE ]
```

Block 2
```
Encrypted  [ 03 ] [ 4A ] [ 94 ] [ 10 ] [ 17 ] [ F4 ] [ B1 ] [ 9A ]
DES key    [ 10 ] [ 32 ] [ 54 ] [ 76 ] [ 98 ] [ BA ] [ DC ] [ FE ]
Decrypted  [ BF ] [ 00 ] [ 00 ] [ 10 ] [ 00 ] [ AE ] [ 88 ] [ 6E ]
```

```
Red   : CRC bytes
Green : Challenge bytes
Blue  : Random / Undefined bytes
```

Event counter = 1
Payout coins remaining = 0
Last payout : coins paid = 5
Last payout : coins unpaid = 0
Hopper status register 1 = 0x00
Hopper status register 2 = 0x00
Hopper status register 3 = 0x10
Level status = 0x00
CRC-16 checksum = 0x6E9E

When the game machine receives a reply to this command it should…

- Verify the ccTalk packet address and packet checksum as usual
- Decrypt both blocks with the current DES key
- Verify the CRC-16 checksum
- Verify the challenge bytes
- Verify the payout status (event counter & coins paid)

10 Game Machine Initialisation

When the game machine starts-up, it will need to check each ccTalk peripheral on the expected address, taking into account that a service engineer could have replaced the peripheral with a new one and that would mean the DES key is no longer valid. It can do this with the 'Request encryption support' command.

If there is no reply to the 'Request encryption support' command then the possibilities are…

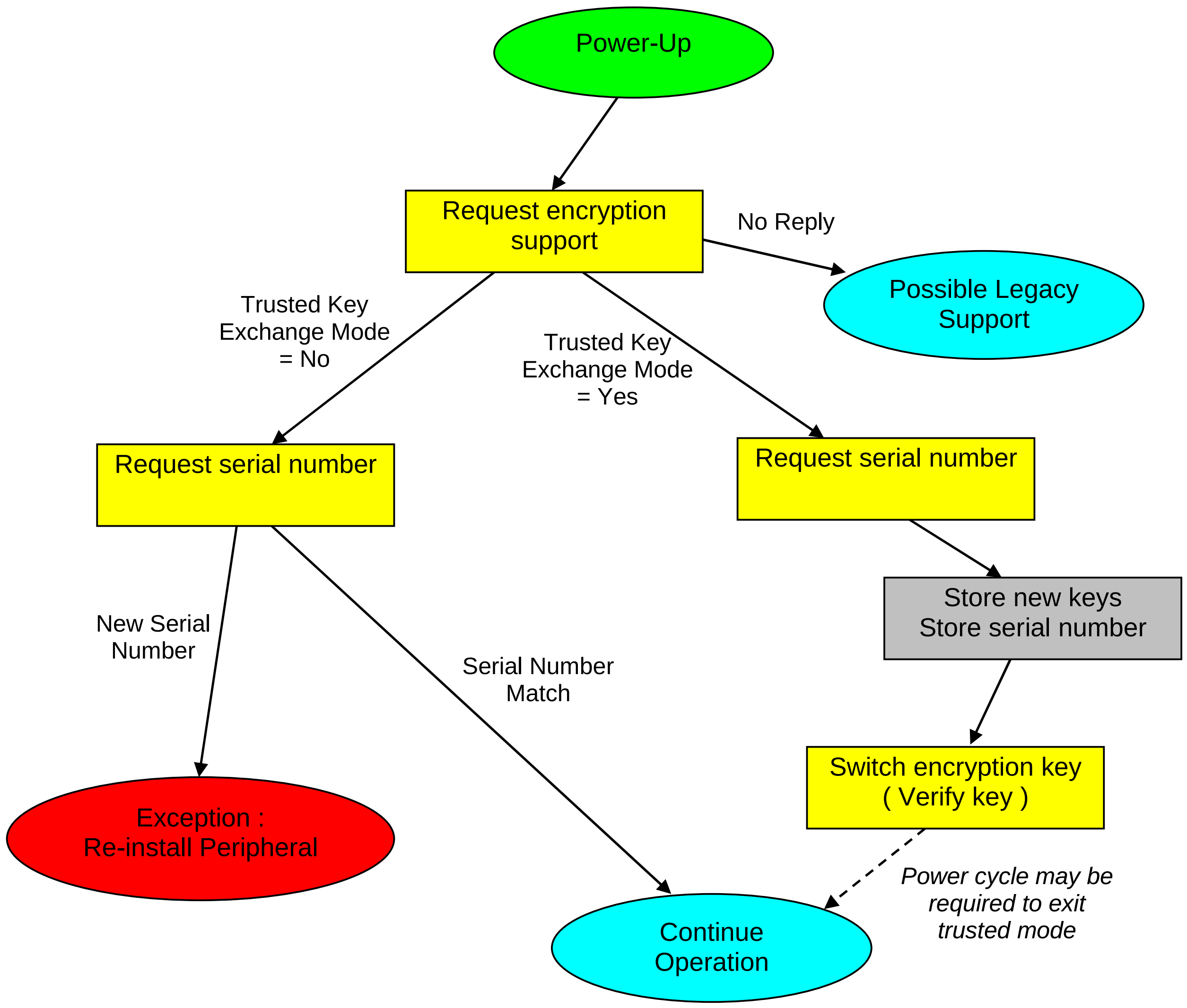
- The peripheral is missing or unpowered
- The peripheral is on a different address to that expected
- This is a 'legacy' peripheral with no support for command header 111

If a legacy peripheral then other initialisation procedures should be used.

If the encryption support reply shows ‘trusted key exchange mode’ then new values of BNV and DES keys should be stored as appropriate. The key can then be verified with the ‘Switch encryption key command’ before it is actually required; perhaps hours later.

The peripheral may exit trusted key exchange mode automatically on reading the keys or it may have to be power cycled / reset; this will vary according to the peripheral and the exact operating requirements should be explained in the product manual. In trusted mode the peripheral is non-functional for obvious security reasons.

If the encryption support reply does not show ‘trusted key exchange mode’ then it must be assumed the keys are the same as when the peripheral was powered off. In this case the serial number should be checked to confirm the peripheral has not changed. If it has then the peripheral must be re-installed in trusted mode and the game should throw an exception.


Power-Up
Request encryption support
No Reply
Possible Legacy Support
Trusted Key Exchange Mode = No
Trusted Key Exchange Mode = Yes
Request serial number
Request serial number
Store new keys
Store serial number
New Serial Number
Serial Number Match
Switch encryption key (Verify key)
Exception : Re-install Peripheral
Power cycle may be required to exit trusted mode
Continue Operation

11 Combined Protocol Level and Command Level Encryption

It is possible to combine protocol level encryption using the BNV key and command level encryption using the DES key. The BNV key wraps the entire message packet and within that the DES key protects payload data. The combined encryption strength is not much better than DES but for some applications the masking of command headers and checksums may provide some additional complexity to simple bus attacks.

There are some complications to be aware of. For instance with BNV and DES encryption enabled, the peripheral address may have been changed and stored to a non-standard value. How can we discover the address ? If BNV encryption is enabled then no ccTalk commands can be transmitted without knowing the BNV key, including header 253, ‘Address poll’, which can be used to discover a peripheral address. To obtain the BNV key we need to send the ‘Request encryption support’ command in trusted mode. Fortunately we can send this command with the broadcast address and assuming no other peripherals are connected to the bus at this time the BNV key will be obtained. We can then use the ‘Address poll’ command to find the address or quickly poll the ccTalk address space (2 to 255) with header 254, ‘Simple poll’, until the peripheral replies with an ACK. The ‘Address change’ command on header 251 can then be used to put the address back.

12 Miscellaneous Security Issues

For hoppers using DES encryption, some of the security features available on older ccTalk hoppers are redundant.

12.1 Header 161, ‘Pump RNG’

There is no need to ‘pump’ the random number generator as DES encryption provides a vast key space.

It is recommended that DES hoppers do not use this command.

12.2 Header 219, ‘Enter new PIN number’

PIN number protection only provides limited security against hoppers being used in unauthorised game machines. The transfer of the PIN number during machine start-up is also susceptible to bus snooping.

It is recommended that DES hoppers do not use this command.

12.3 Header 218, ‘Enter PIN number’

It is recommended that DES hoppers do not use this command. See above.

12.4 Header 137, ‘Switch encryption code’

If BNV encryption is being used alongside DES encryption then this command can be used to rotate the 6-digit key as we do on bill validators every couple of seconds.

The use of this command is optional.

12.5 Header 136 , ‘Store encryption code’

If BNV encryption is being used alongside DES encryption then this command can be used to store the latest 6-digit key inside the hopper so that it will become the new starting value at power-up.

If the BNV key is lost at any point then it can be recovered through trusted key exchange mode.

The use of this command is optional.

13 Example Communication Dump

```
Hopper is on address 3

TX Header = Request encryption support
TX Packet = 03 06 01 6F AA 55 00 00 55 AA 89
RX Packet = 01 11 03 00 00 65 18 40 40 00 00 00 00 00 00 00 00 00 00 00 00 EE
Trusted key exchange mode is off
BNV Key not used

In the examples below,
DES Key = 0x93FC6183BD93C526

desKey[0] = 0x26
desKey[1] = 0xC5
desKey[2] = 0x93
desKey[3] = 0xBD
desKey[4] = 0x83
desKey[5] = 0x61
desKey[6] = 0xFC
desKey[7] = 0x93

TX Header = Request serial number
TX Packet = 03 00 01 F2 0A
RX Packet = 01 03 03 00 EA 03 00 0C
Serial number = 1002

TX Header = Reset device
TX Packet = 03 00 01 01 FB
RX Packet = 01 00 03 00 FC

TX Header = Enable hopper
TX Packet = 03 01 01 A4 A5 B2
RX Packet = 01 00 03 00 FC

TX Header = Request hopper status
TX Packet = 03 00 01 A6 56
RX Packet = 01 04 03 00 00 00 05 00 F3
Event counter = 0
Payout coins remaining = 0
Last payout : coins paid = 5
Last payout : coins unpaid = 0

TX Header = Request cipher key
TX Packet = 03 00 01 A0 5C
RX Packet = 01 08 03 00 CB 15 D1 61 BE 17 37 41 95
Cipher   CB15D161BE173741
DES Key  26C593BD8361FC93
Decrypt  0E646B17BE50A227
XOR      0505050505050505
Encrypt  0B616E12BB55A722
Result   AE09D75526155745
```

```
TX Header = Dispense hopper coins
TX Packet = 03 09 01 A7 AE 09 D7 55 26 15 57 45 05 8D
RX Packet = 01 01 03 00 01 FA
Request 5 coins are paid
ACK returned from hopper ( event counter = 1 )

TX Header = Request hopper status
TX Packet = 03 00 01 A6 56
RX Packet = 01 04 03 00 01 05 00 00 F2
Event counter = 1
Payout coins remaining = 5
Last payout : coins paid = 0
Last payout : coins unpaid = 0

TX Header = Request hopper status
TX Packet = 03 00 01 A6 56
RX Packet = 01 04 03 00 01 04 01 00 F2
RX Time   = 15 ms
Event counter = 1
Payout coins remaining = 4
Last payout : coins paid = 1
Last payout : coins unpaid = 0

TX Header = Request hopper status
TX Packet = 03 00 01 A6 56
RX Packet = 01 04 03 00 01 03 02 00 F2
Event counter = 1
Payout coins remaining = 3
Last payout : coins paid = 2
Last payout : coins unpaid = 0

TX Header = Request hopper status
TX Packet = 03 00 01 A6 56
RX Packet = 01 04 03 00 01 02 03 00 F2
Event counter = 1
Payout coins remaining = 2
Last payout : coins paid = 3
Last payout : coins unpaid = 0

TX Header = Request hopper status
TX Packet = 03 00 01 A6 56
RX Packet = 01 04 03 00 01 01 04 00 F2
Event counter = 1
Payout coins remaining = 1
Last payout : coins paid = 4
Last payout : coins unpaid = 0

TX Header = Request hopper status
TX Packet = 03 00 01 A6 56
RX Packet = 01 04 03 00 01 00 05 00 F2
Event counter = 1
Payout coins remaining = 0
Last payout : coins paid = 5
Last payout : coins unpaid = 0
```

```
TX Header = Request encrypted hopper status
TX Packet = 03 03 01 6D 6D 3E BD 24
RX Packet = 01 10 03 00 72 66 DC 92 E4 40 68 5B 43 17 89 25 23 E7 CE 1A C5
DES Decrypted Block 1 = F26D01000500003E
DES Decrypted Block 2 = BD00001000AE8DB2

Event counter = 1
Payout coins remaining = 0
Last payout : coins paid = 5
Last payout : coins unpaid = 0
Hopper status register  = 0x100000
Level status = 0x00
Verified, 5 coins were paid out of the 5 requested
```

14 Implementation Rules

The following rules must be adhered to meet the ccTalk DES standard.

14.1 Gaming Machine Manufacturers

For maximum security it is recommended that the gaming machine changes the DES key to a new, random value every 24 hours. However, key changes MUST not be done more often than once every 8 hours to simplify the storage requirements on the peripheral. A peripheral product lifetime of 10 years would then require about 11K write cycles.

It is the responsibility of the gaming machine manufacturer to ensure that trusted mode on the peripheral is secured by a mechanical lock or other equivalent means, that once the DES keys have been read they are not stored in a way that is easy to access, that subsequent key changes are to random values which do not follow any kind of pattern or repeat cycle, and that weak or fixed test keys are avoided. Any deviation from this will reduce the security of DES encryption and could lead to a successful attack strategy.

14.2 Peripheral Manufacturers

If trusted mode is entered on a switch then the peripheral should be inhibited from dispensing coins until the switch is restored to its normal operating state.

‘Exiting trusted mode’ means restoring the physical switch position to its normal operating state but there is also a software lockout to prevent the key being read multiple times. See next.

On entering trusted mode, only 1 read of the DES key using header 111, ‘Request encryption support’, is allowed. Another read will report normal operating mode. If the machine needs another go at reading the keys then trusted mode has to be physically exited and re-entered.

It is not a requirement to have header 111 sent by the gaming machine before the peripheral can dispense coins.

For security, measures MUST be taken to ensure the session key is not the same every time a hopper is powered-up or reset. This would open up the hopper to replay attacks.

Peripherals implementing DES must report a ccTalk revision level of at least 4.6 with header 4, ‘Request comms revision’. The reply data = [xxx] [004] [006].