Cryptographic Engine

HIGHLIGHTS

This section of the manual contains the following major topics:

Note: This family reference manual section is meant to serve as a complement to device data sheets. This document applies to all dsPIC33/PIC24 family devices. However, some features in this document will not apply to all devices.

Please consult the note at the beginning of the **"Cryptographic Engine"** chapter in the current device data sheet to check whether this document supports the device you are using.

Device data sheets and family reference manual sections are available for download from the Microchip Worldwide Web site at: http://www.microchip.com.

1.0 INTRODUCTION

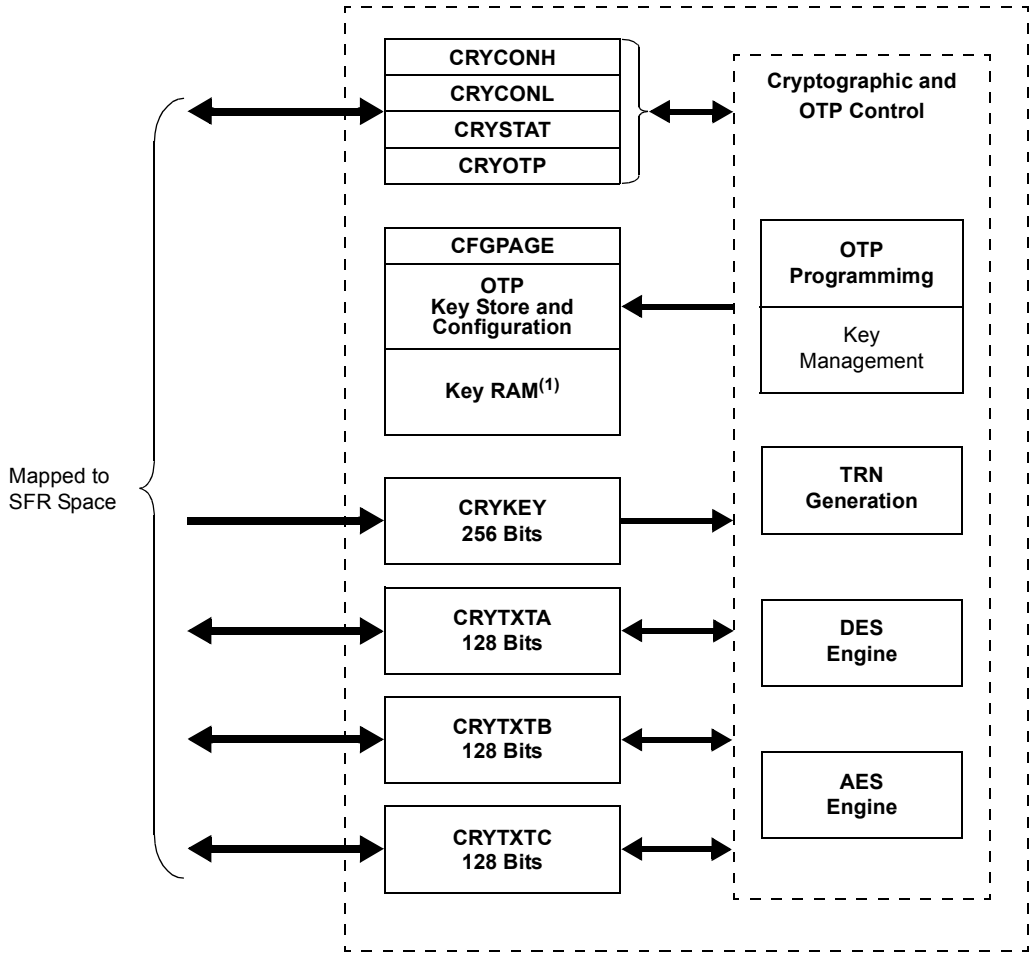
The Cryptographic Engine provides a new set of data security options for dsPIC33/PIC24 devices. Using its own free-standing state machines, the Cryptographic Engine can independently perform NIST-standard encryption and decryption of data, independently of the CPU. This eliminates the concerns of excessive CPU or program memory overhead that encryption and decryption would otherwise require, while enhancing the application's security. This also removes the need to develop an appropriate cryptographic code library for new applications.

Key features include:

- Memory-Mapped, 128-Bit and 256-Bit Memory Spaces for Encryption/Decryption Data
- Multiple Options for Key Storage, Selection and Management
- Support for Internal Context Saving
- Session Key Encryption and Loading
- Half-Duplex Operation
- DES and Triple DES (3DES) Encryption and Decryption (64-bit block size):
  - Supports 64-bit keys and 2-key or 3-key Triple DES
- AES Encryption and Decryption (128-bit block size):
  - Supports key sizes of 128, 192 or 256 bits
- Supports ECB, CBC, CFB, OFB and CTR modes for Both DES and AES Standards
- Programmatically Secure Key Storage:
  - 512-bit OTP array for key storage, not readable from other memory spaces
  - 32-bit configuration page
  - Simple, in-module programming interface
  - 512-bit Key RAM for secure temporary key storage with hardware anti-tamper options (select devices only)
  - Supports Key Encryption Key (KEK)
- Hardware Support for True Random Number Generation (TRNG)
- Support for Pseudorandom Number Generation (PRNG), NIST SP800-90 Compliant
- Hardware-Enabled Anti-Tamper Capabilities on Select Devices

A simplified block diagram of the Cryptographic Engine is shown in Figure 1-1.

Figure 1-1: Cryptographic Engine Block Diagram



Note 1: Implemented in select devices only. Refer to the specific device data sheet for details.

2.0 REGISTERS

2.1 Control Registers

The Cryptographic Engine uses four status and control registers:

- CRYCONH and CRYCONL
- CRYSTAT
- CRYOTP

The CRYCON registers (Register 2-1 and Register 2-2) set all of the parameters for encryption and decryption operations, including the secure management of cryptographic keys. The module is enabled and encryption/decryption operation is initiated from the CRYCONL register.

The CRYSTAT register (Register 2-3) indicates the status of the encryption/decryption operation. It also contains the module-level interrupt flags.

The CRYOTP register (Register 2-4) controls the programming of the Secure OTP Array. This is discussed in more detail in **Section 4.3.1 "Stored Keys (Secure OTP Array)"**.

Although not technically a register, Page 0 of the Secure OTP Array functions as if it were a register of OTP Configuration fuses. It is neither memory-mapped to the microcontroller's data space, nor is it directly programmable. The location and function of its control bits are shown in Register 2-5.

2.1.1 RESET STATES AND BEHAVIOR

Many of the bits with control functions behave in ways that might not otherwise be expected when compared to other dsPIC33/PIC24 peripherals. Most bits have additional Reset constraints beyond the normal system Reset. Other bits have values that may change during initialization; still others are locked out from changes during the module's operation.

Specific behaviors are noted in the register description and throughout this chapter. Please also see **Section 6.0 "Effects of a Reset"** for specific information on Reset behavior.

2.2 Data Register Spaces

In addition to the control registers, there are four register spaces used for cryptographic data and key storage:

- CRYTXTA
- CRYTXTB
- CRYTXTC
- CRYKEY

Although mapped into the SFR space, all of these data spaces are actually implemented as 128-bit or 256-bit wide arrays, rather than groups of 16-bit wide Data registers. Reads and writes to and from these arrays are automatically handled as if they were any other register in the SFR space.

CRYTXTA through CRYTXTC are 128-bit wide spaces; they are used for writing data to, and reading from, the Cryptographic Engine. Additionally, they are also used for storing intermediate results of the encryption/decryption operation. None of these registers may be written to when the module is performing an operation (CRYGO = 1).

CRYTXTA and CRYTXTB normally serve as inputs to the encryption/decryption process. CRYTXTA usually contains the initial plaintext or ciphertext to be encrypted or decrypted. Depending on the mode of operation, CRYTXTB may contain the ciphertext output or intermediate cipher data. It may also function as a programmable length counter in certain operations.

CRYTXTC is primarily used to store the final output of an encryption/decryption operation. It is also used as the input register for data to be programmed to the Secure OTP Array.

CRYKEY is a 256-bit wide space, used to store cryptographic keys for the selected operation. It is writable from both the SFR space and the Secure OTP Array. Although mapped into the SFR space, it is a write-only memory area; any data placed here, regardless of its source, cannot be read back by any run-time operations. This feature helps to ensure the security of any key data.

Register 2-1: CRYCONH: Cryptographic Control Register High

U-0	R/W-0(1)	R/W-0(1)	R/W-0(1)	R/W-0(1)	R/W-0(1)	R/W-0(1)	R/W-0(1)
—	CTRSIZE6(2,3)	CTRSIZE5(2,3)	CTRSIZE4(2,3)	CTRSIZE3(2,3)	CTRSIZE2(2,3)	CTRSIZE1(2,3)	CTRSIZE0(2,3)
bit 15							bit 8

R/W-0(1)	R/W-0(1)	R/W-0(1)	R/S-0	R/W-0(1)	R/W-0(1)	R/W-0(1)	R/W-0(1)
SKEYSEL	KEYMOD1(2)	KEYMOD0(2)	KEYWIPE(4)	KEYSRC3(2)	KEYSRC2(2)	KEYSRC1(2)	KEYSRC0(2)
bit 7							bit 0

Legend:	S = Settable Only bit		
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

bit 15 **Unimplemented:** Read as '0'

bit 14-8 **CTRSIZE<6:0>:** Counter Size Select bits(1,2,3)

Counter is defined as CRYTXTB<n:0>, where n = CTRSIZE<6:0>. Counter increments after each operation and generates a rollover event when the counter rolls over from (2^{n-1} – 1) to 0.

1111111 = 128 bits (CRYTXTB<127:0>)
1111110 = 127 bits (CRYTXTB<126:0>)
•
•
0000010 = 3 bits (CRYTXTB<2:0>)
0000001 = 2 bits (CRYTXTB<1:0>)
0000000 = 1 bit (CRYTXTB<0>); rollover event occurs when CRYTXTB<0> toggles from '1' to '0'

bit 7 **SKEYSEL:** Session Key/TRN Storage Select bit(1)

1 = Next operation (Key Generation/Encryption/Loading/TRN storage) uses CRYKEY<255:128> as its destination
0 = Next operation uses CRYKEY<127:0> as its destination

bit 6-5 **KEYMOD<1:0>:** AES/DES Encryption/Decryption Key Mode/Key Length Select bits(1,2)

For DES Encryption/Decryption Operations (CPHRSEL = 0):
11 = 64-bit, 3-key 3DES
10 = Reserved
01 = 64-bit, standard 2-key 3DES
00 = 64-bit DES

For AES Encryption/Decryption Operations (CPHRSEL = 1):
11 = Reserved
10 = 256-bit AES
01 = 192-bit AES
00 = 128-bit AES

bit 4 **KEYWIPE:** Key RAM Erase Enable bit(4)

1 = Erases Key RAM and clears all write locks (bit cleared by hardware on next clock)
0 = No erase operation requested or previous erase has completed

bit 3-0 **KEYSRC<3:0>:** Cipher Key Source bits(1,2)

Refer to Table 4-1 and Table 4-2 for KEYSRC<3:0> values.

Note 1: These bits are reset on system Resets or whenever the CRYMD bit (in the PMDx register) is set.
2: Writes to these bit fields are locked out whenever an operation is in progress.(CRYGO bit is set).
3: Used only in CTR operations when CRYTXTB is being used as a counter; otherwise, these bits have no effect.
4: Not implemented on all devices; refer to the specific device data sheet for details.

Register 2-2: CRYCONL: Cryptographic Control Register Low

R/W-0	U-0	R/W-0	R/W-0(1)	R/W-0(1)	R/W-0(1)	U-0	R/W-0, HC(1)
CRYON	—	CRYSIDL	ROLLIE	DONEIE	FREEIE	—	CRYGO
bit 15							bit 8

R/W-0(1)	R/W-0(1)	R/W-0(1)	R/W-0(1)	R/W-0(1)	R/W-0(1)	R/W-0(1)	R/W-0(1)
OPMOD3(2)	OPMOD2(2)	OPMOD1(2)	OPMOD0(2)	CPHRSEL(2)	CPHRMOD2(2)	CPHRMOD1(2)	CPHRMOD0(2)
bit 7							bit 0

Legend:	HC = Hardware Clearable bit		
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

bit 15 **CRYON:** Cryptographic Engine Enable bit
1 = Module is enabled
0 = Module is disabled

bit 14 **Unimplemented:** Read as '0'

bit 13 **CRYSIDL:** Cryptographic Engine Stop in Idle Control bit
1 = Stops module operation in Idle mode
0 = Continues module operation in Idle mode

bit 12 **ROLLIE:** CRYTXTB Rollover Interrupt Enable bit(1)
1 = Generates an interrupt event when the counter portion of CRYTXTB rolls over to '0'
0 = Does not generate an interrupt event when the counter portion of CRYTXTB rolls over to '0'

bit 11 **DONEIE:** Operation Done Interrupt Enable bit(1)
1 = Generates an interrupt event when the current cryptographic operation completes
0 = Does not generate an interrupt event when the current cryptographic operation completes; software must poll the CRYGO or CRYBSY bit to determine when the current cryptographic operation has completed

bit 10 **FREEIE:** Input Text Interrupt Enable bit(1)
1 = Generates an interrupt event when the input text (plaintext or ciphertext) is consumed during the current cryptographic operation
0 = Does not generate an interrupt event when the input text is consumed

bit 9 **Unimplemented:** Read as '0'

bit 8 **CRYGO:** Cryptographic Engine Start bit(1)
1 = Starts the operation specified by OPMOD<3:0> (cleared automatically when operation is done)
0 = Stops the current operation (when cleared by software); also indicates the current operation has completed (when cleared by hardware)

bit 7-4 **OPMOD<3:0>:** Operating Mode Selection bits(1,2)
1111 = Loads the Session Key (decrypts Session Key in CRYTXTA/CRYTXTB using the Key Encryption Key and writes to CRYKEY)
1110 = Encrypts Session Key (encrypts Session Key in CRYKEY using the Key Encryption Key and writes to CRYTXTA/CRYTXTB)
1011 = Generates a True Random Number (TRN) and stores the result in CRYKEY<255:128> or <127:0>, as determined by SKEYSEL (CRYCONH<7>)
1010 = Generates a True Random Number (TRN) and stores the result in CRTYXTA
1001
• • • = Reserved
0011
0010 = AES Decryption Key Expansion
0001 = Decryption
0000 = Encryption

Note 1: These bits are reset on system Resets or whenever the CRYMD bit is set.
2: Writes to these bit fields are locked out whenever an operation is in progress (CRYGO bit is set).

Register 2-2: CRYCONL: Cryptographic Control Register Low (Continued)

bit 3 **CPHRSEL:** Cipher Engine Select bit(1,2)
- 1 = AES engine
- 0 = DES engine

bit 2-0 **CPHRMOD<2:0>:** Cipher Mode bits(1,2)
- 11x = Reserved
- 101 = Reserved
- 100 = Counter (CTR) mode
- 011 = Output Feedback (OFB) mode
- 010 = Cipher Feedback (CFB) mode
- 001 = Cipher Block Chaining (CBC) mode
- 000 = Electronic Codebook (ECB) mode

Note 1: These bits are reset on system Resets or whenever the CRYMD bit is set.

2: Writes to these bit fields are locked out whenever an operation is in progress (CRYGO bit is set).

Register 2-3: CRYSTAT: Cryptographic Status Register

U-0	U-0	U-0	U-0	U-0	U-0	U-0	U-0
—	—	—	—	—	—	—	—
bit 15							bit 8

R/HSC-x(1)	R/HSC-0(1)	R/C-0, HS(2)	R/C-0, HS(2)	U-0	R/HSC-0(1)	R/HSC-x(1)	R/HSC-x(1)
CRYBSY(3)	TXTABSY	CRYABRT	ROLLOVR	—	MODFAIL(4)	KEYFAIL(3,4)	PGMFAIL(3,4)
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
HS = Hardware Settable bit	C = Clearable bit	HSC = Hardware Settable/Clearable bit	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Reset state conditional

bit 15-8 **Unimplemented:** Read as '0'

bit 7 **CRYBSY:** Cryptographic Engine Busy Status bit(1,3)
- 1 = A cryptographic operation is in progress
- 0 = No cryptographic operation is in progress

bit 6 **TXTABSY:** CRYTXTA Busy Status bit(1)
- 1 = The CRYTXTA register is busy and may not be written to
- 0 = The CRYTXTA register is free and may be written to

bit 5 **CRYABRT:** Cryptographic Operation Aborted Status bit(2)
- 1 = Last operation was aborted by clearing the CRYGO bit in software (additional operations cannot be performed until the bit is cleared)
- 0 = Last operation completed normally (CRYGO cleared in hardware)

bit 4 **ROLLOVR:** Counter Rollover Status bit(2)
- 1 = The CRYTXTB counter rolled over on the last CTR mode operation
- 0 = Rollover event did not occur

bit 3 **Unimplemented:** Read as '0'

bit 2 **MODFAIL:** Mode Configuration Fail Flag bit(1,4)
- 1 = Currently selected Operating and Cipher mode configuration is invalid, the PGM bit cannot be set until a valid mode is selected (automatically cleared by hardware with any valid configuration)
- 0 = Currently selected Operating and Cipher mode configuration is valid

bit 1 **KEYFAIL:** Key Configuration Fail Status bit(1,3,4)

See Table 4-1 and Table 4-2 for invalid key configurations.
- 1 = Currently selected key and mode configurations are invalid, the PGM bit cannot be set until a valid mode is selected (automatically cleared by hardware with any valid configuration)
- 0 = Currently selected configurations are valid

bit 0 **PGMFAIL:** Key Storage/Configuration Program Configuration Fail Flag bit(1,3,4)
- 1 = The page indicated by KEYPG<3:0> is reserved or locked; the PGM bit cannot be set and no programming operation can be started
- 0 = The page indicated by KEYPG<3:0> is available for programming

Note 1: These bits are reset on system Resets or whenever the CRYMD bit is set.

2: These bits are reset on system Resets when the CRYMD bit is set or when CRYGO is cleared.

3: These bits are automatically set during all OTP read operations, including the initial read at POR. Once the read has completed, the bit assumes the proper state that reflects the current configuration.

4: These bits are functional even when the module is disabled (CRYON = 0). This allows mode configurations to be validated for compatibility before enabling the module.

Register 2-4: CRYOTP: Cryptographic OTP Page Program Control Register

U-0	U-0	U-0	U-0	U-0	U-0	U-0	R/S-0, HC(1,2)
—	—	—	—	—	—	—	KEYPSEL
bit 15							bit 8

R-x, HSC(2)	R/W-0(2)	R/S-1, HC(3)	R/W-0(2)	R/W-0(2)	R/W-0(2)	R/W-0(2)	R/S-0, HC(4)
PGMTST	OTPIE	CRYREAD(5,6)	KEYPG3	KEYPG2	KEYPG1	KEYPG0	CRYWR(5,6)
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
S = Settable Only bit	HC = Hardware Clearable bit	HSC = Hardware Settable/Clearable bit	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

bit 15-9 **Unimplemented:** Read as '0'

bit 8 **KEYPSEL:** Key Storage Programming Select bit(1,2)
1 = Programming operations write to Key RAM
0 = Programming operations write to the Secure OTP Array

bit 7 **PGMTST:** Key Storage/Configuration Program Test bit(2)
This bit mirrors the state of the TSTPGM bit and is used to test the programming of the Secure OTP Array after programming.
1 = TSTPGM (CFGPAGE<30>) is programmed ('1')
0 = TSTPGM is not programmed ('0')

bit 6 **OTPIE:** Key Storage/Configuration Program Interrupt Enable bit(2)
1 = Generates an interrupt when the current program or read operation completes
0 = Does not generate an interrupt when the current program or read operation completes; software must poll the TSTPGM, CRYREAD or CRYBSY bit to determine when the current programming operation is complete

bit 5 **CRYREAD:** Cryptographic Key Storage/Configuration Read bit(3,5,6)
1 = Read operation is in progress (when CRYGO = 1; automatically cleared by hardware when complete)
0 = Read operation has completed

bit 4-1 **KEYPG<3:0>:** Key Storage/Configuration Program Page Select bits(2)
1111
• • • = Reserved
1001
1000 = OTP Page 8
0111 = OTP Page 7
0110 = OTP Page 6
0101 = OTP Page 5
0100 = OTP Page 4
0011 = OTP Page 3
0010 = OTP Page 2
0001 = OTP Page 1
0000 = Configuration Page (CFGPAGE); OTP Page 0

bit 0 **CRYWR:** Cryptographic Key Storage/Configuration Program bit(4,5,6)
1 = Programs the Key Storage/Configuration bits with the value found in CRYTXTC<63:0>
0 = Program operation has completed

Note 1: Not implemented in all devices; refer to the specific device data sheet for details.
2: These bits are reset on system Resets or whenever the CRYMD bit is set.
3: This bit is reset on system Resets only.
4: These bits are reset on system Resets when the CRYMD bit is set or when CRYGO is cleared.
5: Set this bit only when CRYON = 1 and CRYGO = 0. Do not set both CRYREAD and CRYWR at any given time.
6: Do not clear CRYON or these bits while they are set; always allow the hardware operation to complete and clear the bit automatically.

Register 2-5: CFGPAGE: Cryptographic Secure Array Configuration Register (OTP Page 0)

r-x	R/P-x	R/P-x	R/P-x	R/P-x	R/P-x	R/P-x	R/P-x
—	TSTPGM(1)	KEYSZRAM1(2)	KEYSZRAM0(2)	KEY7TYPE1	KEY7TYPE0	KEY5TYPE1	KEY5TYPE0
bit 31							bit 24

R/P-x	R/P-x	R/P-x	R/P-x	R/P-x	R/P-x	R/P-x	R/P-x
KEY3TYPE1	KEY3TYPE0	KEY1TYPE1	KEY1TYPE0	SKEYEN	LKYSRC7	LKYSRC6	LKYSRC5
bit 23							bit 16

R/P-x	R/P-x	R/P-x	R/P-x	R/P-x	R/P-x	R/P-x	R/P-x
LKYSRC4	LKYSRC3	LKYSRC2	LKYSRC1	LKYSRC0	SRCLCK	WRLOCK8	WRLOCK7
bit 15							bit 8

R/P-x	R/P-x	R/P-x	R/P-x	R/P-x	R/P-x	R/P-x	R/P-x
WRLOCK6	WRLOCK5	WRLOCK74	WRLOCK3	WRLOCK2	WRLOCK1	WRLOCK0	SWKYDIS
bit 7							bit 0

Legend:	r = Reserved bit		
R = Readable bit	P = Program Once bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

bit 31 **Reserved:** Do not modify

bit 30 **TSTPGM:** Customer Program Test bit(1)
- 1 = CFGPAGE has been programmed
- 0 = CFGPAGE has not been programmed

bit 29-28 **KEYSZRAM<1:0>:** Key Type Selection bits (Key RAM Pages)(2)
- 11 = 192/256-bit AES operations only
- 10 = 128-bit AES operations only
- 01 = DES3 operations only
- 00 = DES/DES2 operations only

bit 27-26 **KEY7TYPE<1:0>:** Key Type Selection bits (OTP Pages 7 and 8)
- 11 = 192/256-bit AES operations only
- 10 = 128-bit AES operations only
- 01 = DES3 operations only
- 00 = DES/DES2 operations only

bit 25-24 **KEY5TYPE<1:0>:** Key Type Selection bits (OTP Pages 5 and 6)
- 11 = 192/256-bit AES operations only
- 10 = 128-bit AES operations only
- 01 = DES3 operations only
- 00 = DES/DES2 operations only

bit 23-22 **KEY3TYPE<1:0>:** Key Type Selection bits (OTP Pages 3 and 4)
- 11 = 192/256-bit AES operations only
- 10 = 128-bit AES operations only
- 01 = DES3 operations only
- 00 = DES/DES2 operations only

Note 1: This bit's state is mirrored by the PGMTST bit (CRYOTP<7>).

2: Implemented only in devices that implement Key RAM. Refer to the specific device data sheet for details.

Register 2-5: CFGPAGE: Cryptographic Secure Array Configuration Register (OTP Page 0) (Continued)

bit 21-20 **KEY1TYPE<1:0>:** Key Type Selection bits (OTP Pages 1 and 2)
11 = 192/256-bit AES operations only
10 = 128-bit AES operations only
01 = DES3 operations only
00 = DES/DES2 operations only

bit 19 **SKEYEN:** Session Key Enable bit
1 = Stored Key #1 may be used only as a Key Encryption Key
0 = Stored Key #1 may be used for any operation

bit 18-11 **LKYSRC<7:0>:** Locked Key Source Configuration bits
If SRCLCK = 1:
1xxxxxxx = Key source is as if KEYSRC<3:0> = 1111
01xxxxxx = Key source is as if KEYSRC<3:0> = 0111
001xxxxx = Key source is as if KEYSRC<3:0> = 0110
0001xxxx = Key source is as if KEYSRC<3:0> = 0101
00001xxx = Key source is as if KEYSRC<3:0> = 0100
000001xx = Key source is as if KEYSRC<3:0> = 0011
0000001x = Key source is as if KEYSRC<3:0> = 0010
00000001 = Key source is as if KEYSRC<3:0> = 0001
00000000 = Key source is as if KEYSRC<3:0> = 0000
If SRCLCK = 0:
These bits are ignored.

bit 10 **SRCLCK:** Key Source Lock bit
1 = The key source is determined by the LKYSRC<7:0> bits (software key selection is disabled)
0 = The key source is determined by the KEYSRC<3:0> (CRYCONH<3:0>) bits (locked key selection is disabled)

bit 9-1 **WRLOCK<8:0>:** Write Lock Page Enable bits
For OTP Pages 0 (CFGPAGE) through 8:
1 = OTP page is permanently locked and may not be programmed
0 = OTP page is unlocked and may be programmed

bit 0 **SWKYDIS:** Software Key Disable bit
1 = Software key (CRYKEY register) is disabled; when KEYSRC<3:0> = 0000; the KEYFAIL status bit will be set and no Encryption/Decryption/Session Key operations can be started until KEYSRC<3:0> are changed to a value other than '0000'
0 = Software key (CRYKEY register) can be used as a key source when KEYSRC<3:0> = 0000

Note 1: This bit's state is mirrored by the PGMTST bit (CRYOTP<7>).
2: Implemented only in devices that implement Key RAM. Refer to the specific device data sheet for details.

3.0 THEORY OF OPERATION

It is not possible to present even a brief introduction to cryptography and cryptographic techniques within this chapter. Instead, this section provides a brief overview of the specific encryption and decryption methods implemented by the Cryptographic Engine and their many variations. The internal details of the AES and DES Encryption standards are not discussed.

For users requiring a more complete background or additional information, please see the additional reference material listed in **Section 8.0 "Selected References"**.

3.1 Symmetric Ciphers

The simplest type of modern cipher to understand is the **Symmetric Cipher**. Such a cipher combines the plaintext message with a **Secret Key** (also sometimes referred to as a **Private Key**) to produce the encrypted ciphertext. Decrypting the ciphertext requires knowledge of the Secret Key; hence, the cipher is symmetric. Technically speaking, a Symmetric Cipher algorithm does not have to require the same Secret Key for both encryption and decryption, but it should be trivial to generate one key from the other if different keys are used.

Symmetric Ciphers can be further broken down into **Block Ciphers**, discussed in **Section 3.2 "Block Ciphers"**, and **Stream Ciphers**, discussed in **Section 3.3 "Stream Ciphers"**.

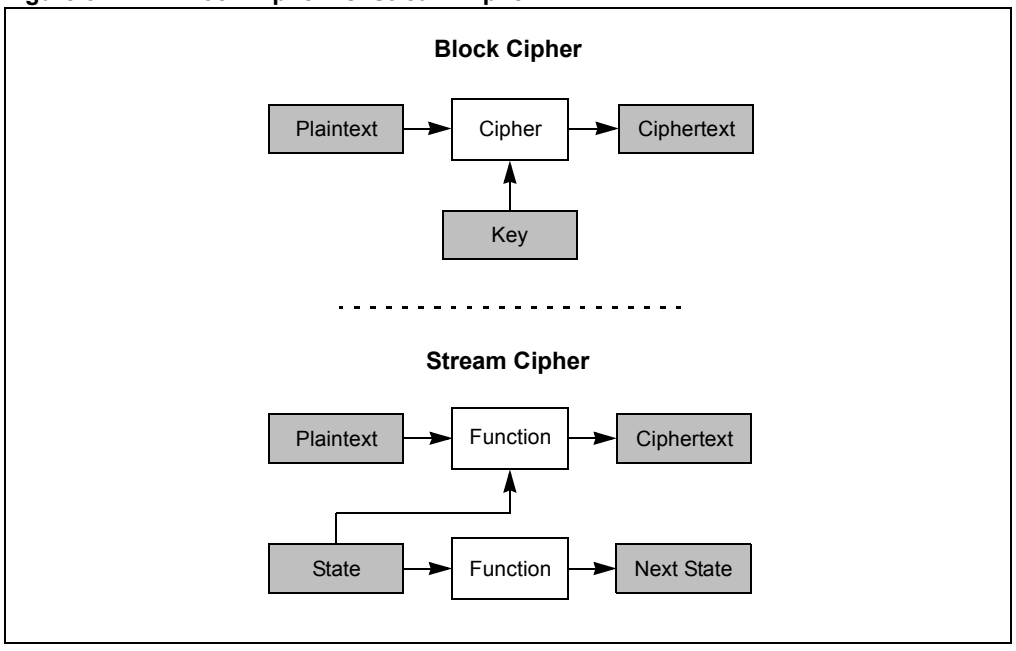
3.1.1 BLOCK CIPHERS VERSUS STREAM CIPHERS

While both Block and Stream Ciphers (also known as State Ciphers) create a ciphertext message from a plaintext message, there are several key differences between the two. Block Ciphers apply the same cipher function to each plaintext block, as shown in Figure 3-1. Each piece of plaintext (i.e., a "block") is relatively large, as compared to its counterpart in a Stream Cipher. Creating a ciphertext message only requires the plaintext message, a key and a cipher function.

Stream Ciphers, on the other hand, require the plaintext message, the current stream "state", a function to create the next "state", and a function to create the ciphertext message from the plaintext message and the current state. The size of the pieces of plaintext being transformed are typically small, even as small as one bit, and hence the term, "Stream Cipher".

Historically, Stream Ciphers have been mathematically simpler, and therefore, faster and easier to implement in hardware. The well-known ARCFOUR (RC4) cipher is an example of a Stream Cipher. The security of Stream Ciphers relies on making one or both of the functions cryptographically secure.

Figure 3-1: Block Cipher vs. Stream Cipher



3.2 Block Ciphers

Block Ciphers are ciphers that work on a plaintext block of a fixed length to produce a ciphertext block of the same length. Given a particular key, there is a 1-to-1 correspondence between the plaintext block and the ciphertext block. For this reason, Block Ciphers are sometimes referred to as **Codebooks**, as it would be theoretically possible to implement the cipher as an extremely large look-up table for each key value. This is, of course, impractical for modern Block Cipher algorithms, and so they are implemented as sequences of mathematical operations, well-suited to implementation on a computer. Well-known Block Ciphers include **Data Encryption Standard (DES)**, **Advanced Encryption Standard (AES)** and **Blowfish**.

3.2.1 ELECTRONIC CODEBOOK MODE

The simple usage of a Block Cipher that we have discussed so far is known as an Electronic Codebook (ECB) mode, and is shown in Figure 3-2 and Figure 3-3. The formal notation to describe the operations is shown in Equation 3-1.

Equation 3-1: ECB Encryption and Decryption

Encryption:

$C_i = E_K(P_i)$

(Ciphertext Block i is produced by encrypting Plaintext Block i using Key K).

Decryption:

$P_i = D_K(C_i)$

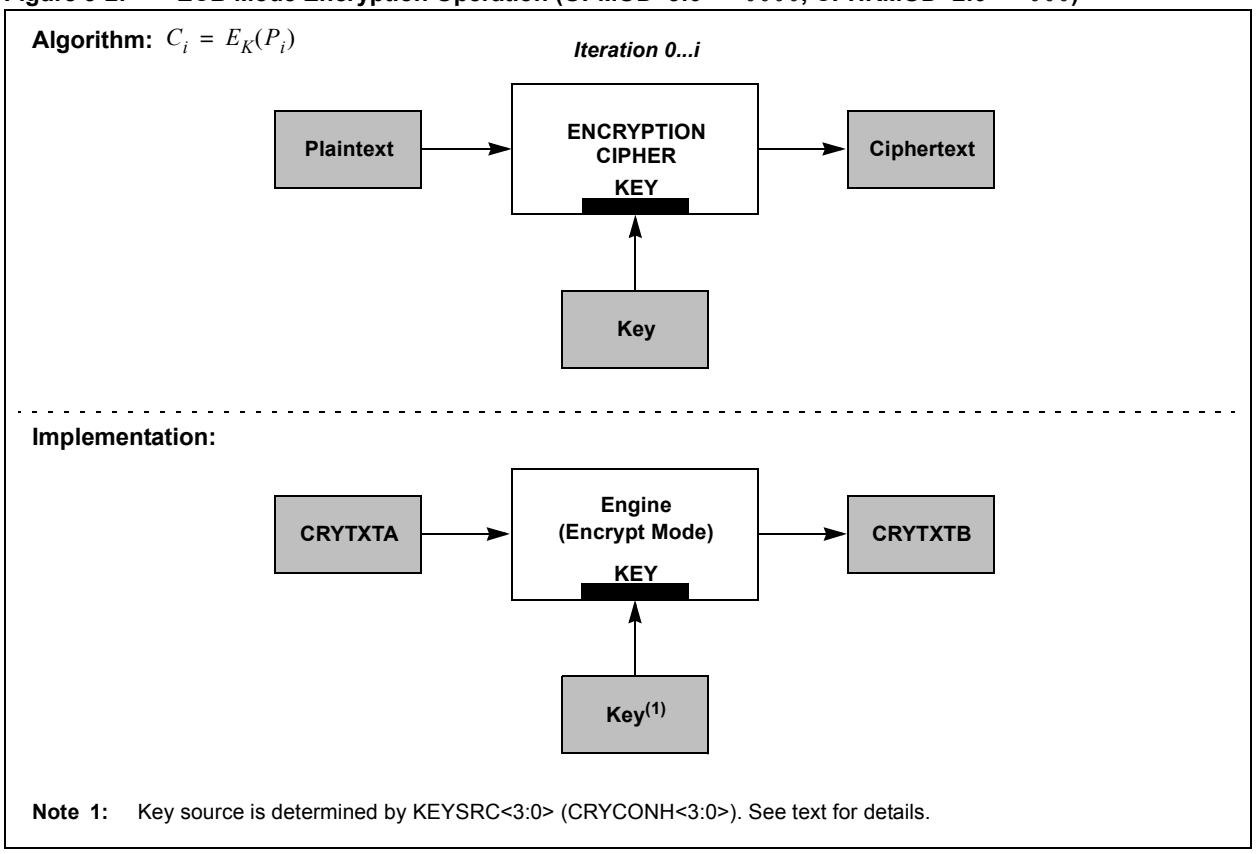
(Plaintext Block i is produced by decrypting Ciphertext Block i using Key K).

There are several problems with simply using a Block Cipher in ECB mode to encrypt data. First, since each plaintext block encrypts to the same ciphertext block every time, it is possible to associate the ciphertext block with an event without ever knowing the plaintext block. When one wishes to trigger the event, they can simply resend the ciphertext block; a process known as a **Replay Attack**.

In addition, most Block Cipher algorithms in ECB mode do nothing to scramble repetitive data, making the plaintext block somewhat reversible from the ciphertext block. Encrypting any sort of a simple graphic image using ECB mode would result in an encrypted graphic image that would probably still be recognizable.

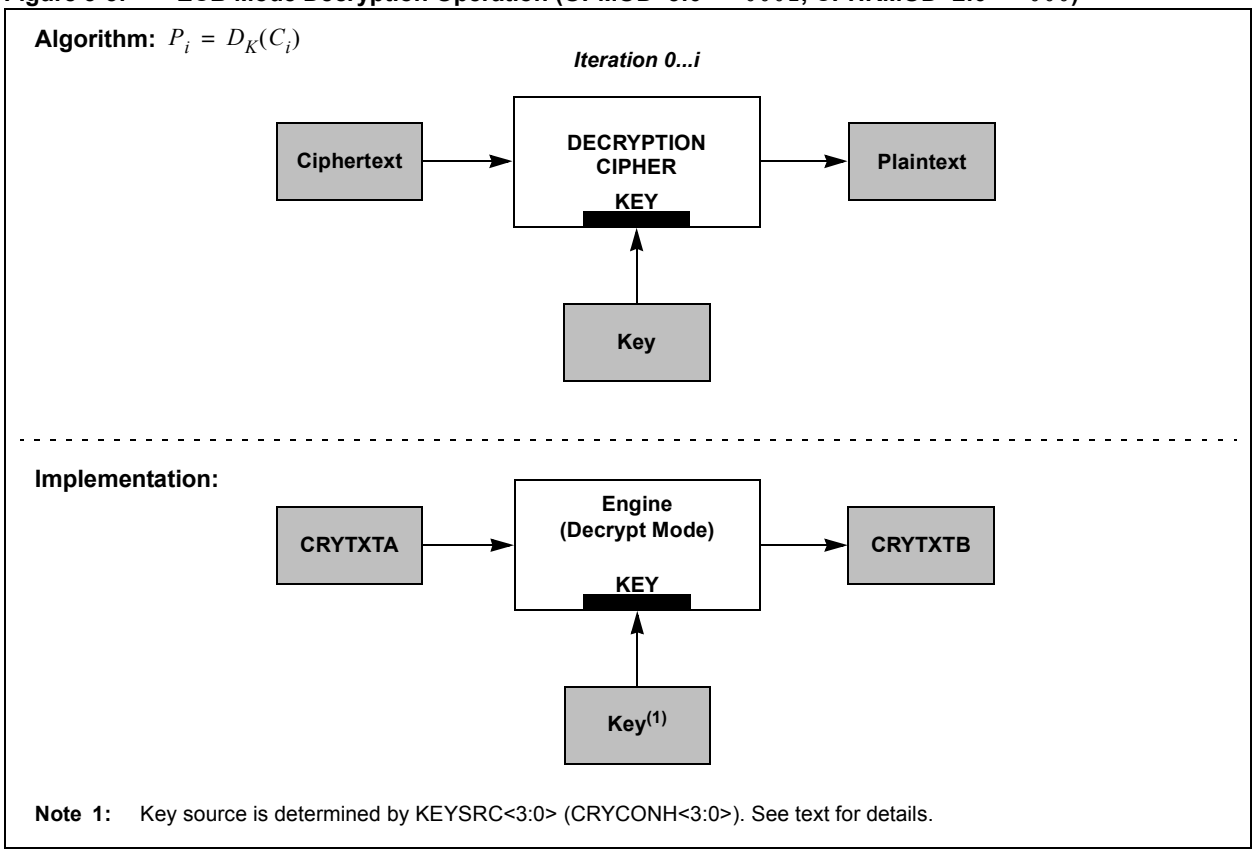
Therefore, Block Ciphers are generally used in one of the various Block Cipher modes described in the following sections.

Figure 3-2: ECB Mode Encryption Operation (OPMOD<3:0> = 0000, CPHRMOD<2:0> = 000)



Note 1: Key source is determined by KEYSRC<3:0> (CRYCONH<3:0>). See text for details.

Figure 3-3: ECB Mode Decryption Operation (OPMOD<3:0> = 0001, CPHRMOD<2:0> = 000)



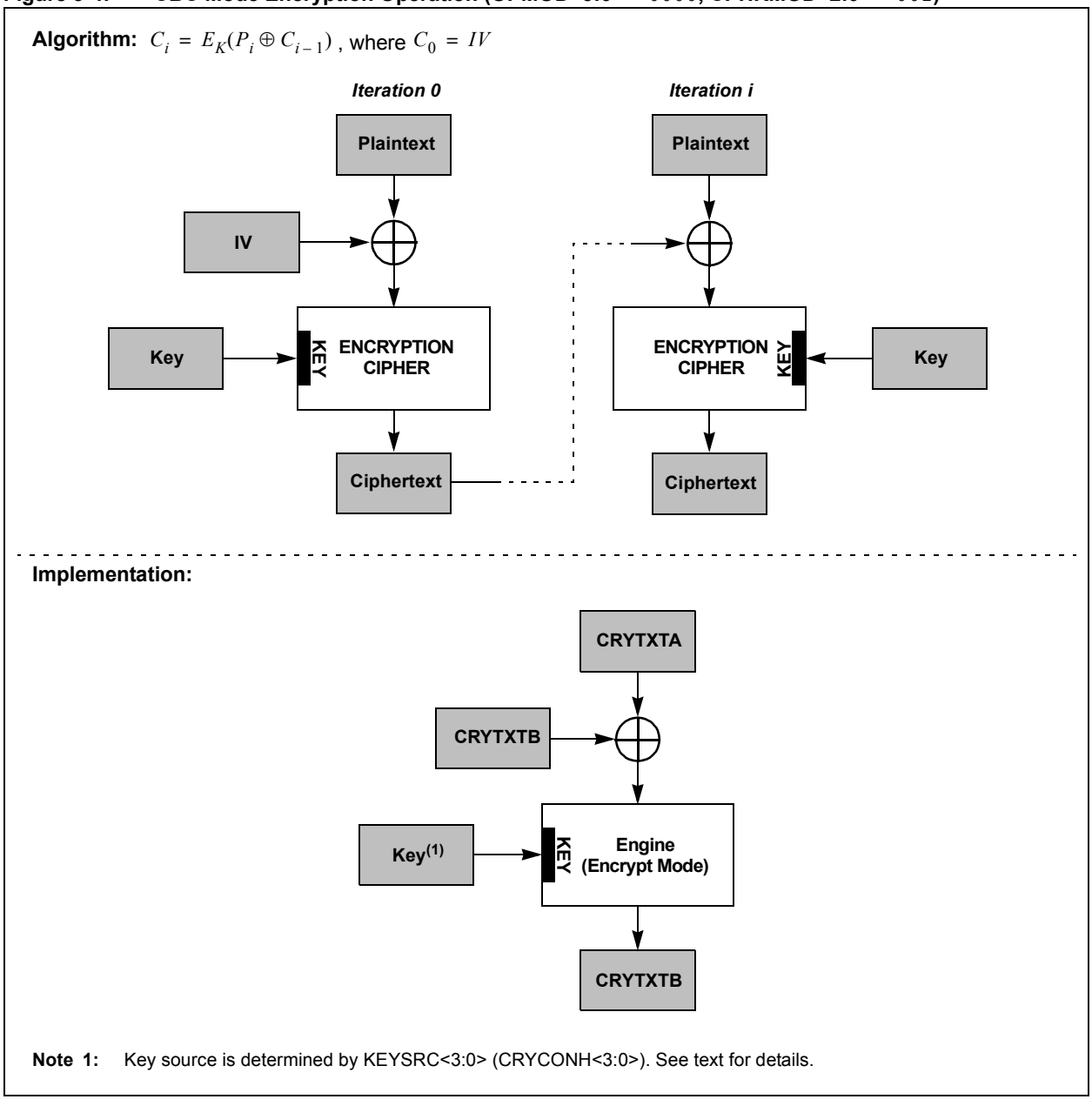
Note 1: Key source is determined by KEYSRC<3:0> (CRYCONH<3:0>). See text for details.

3.2.2 CIPHER BLOCK CHAINING MODE

In **Cipher Block Chaining (CBC)** mode, pictured in Figure 3-4 and Figure 3-5, each plaintext block is XORed with the results of the previous block encryption operation before being encrypted. In this way, all plaintext blocks depend on the previous block, making it more difficult to remove, add or change individual blocks without detection. In addition, the encryption for the first block is performed using the XOR of the plaintext block and an **Initial Value (IV)**. This IV can be changed for each message, making it more resistant to Replay Attacks.

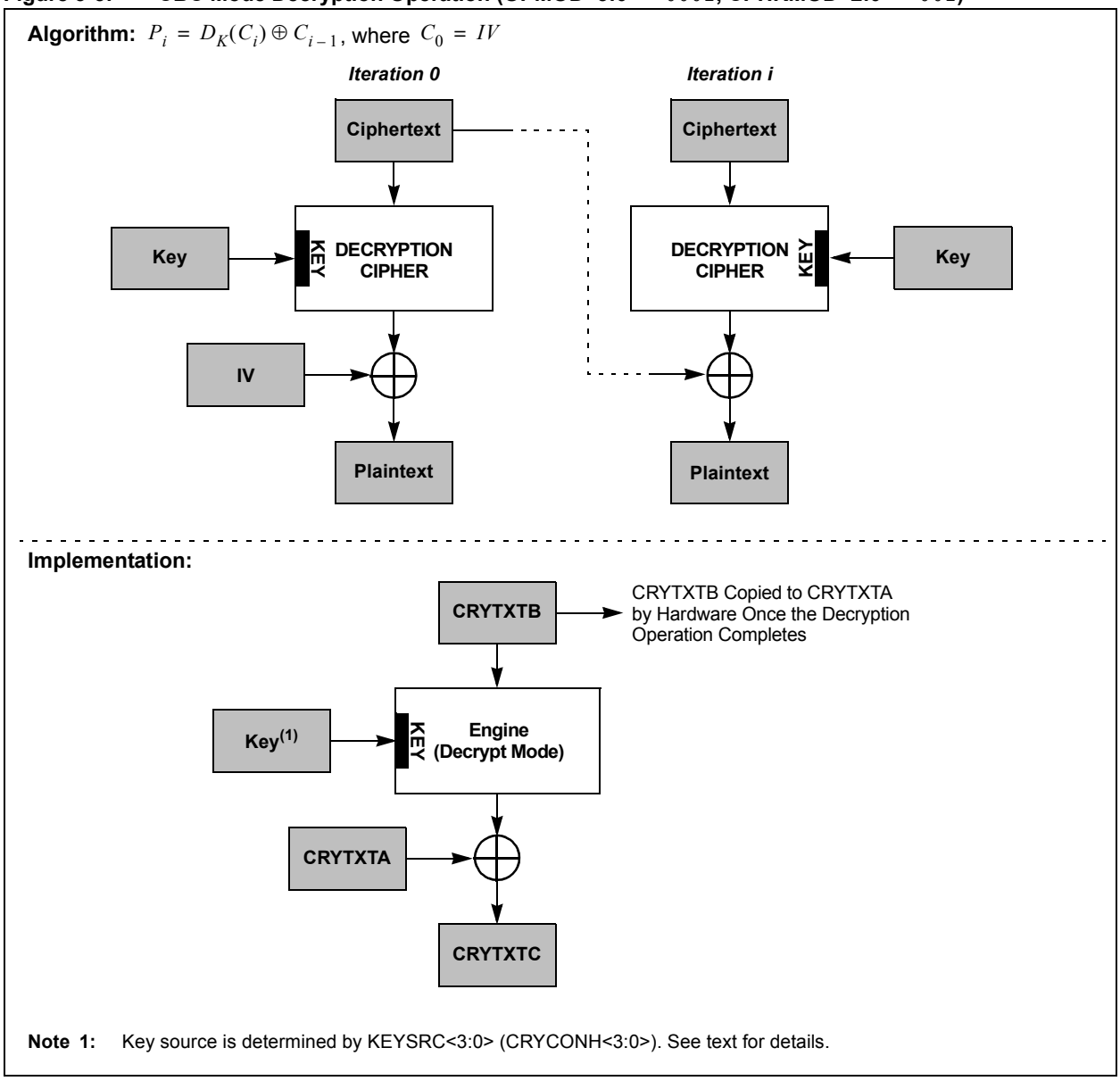
The major drawback to this mode of operation is that the encryption process must be done sequentially, and cannot, therefore, take advantage of computer parallelism to speed up operation. However, for decryption, each plaintext block may be decrypted from exactly two ciphertext blocks, and is therefore, better suited to parallel processing.

Figure 3-4: CBC Mode Encryption Operation (OPMOD<3:0> = 0000, CPHRMOD<2:0> = 001)



Note 1: Key source is determined by KEYSRC<3:0> (CRYCONH<3:0>). See text for details.

Figure 3-5: CBC Mode Decryption Operation (OPMOD<3:0> = 0001, CPHRMOD<2:0> = 001)



Note 1: Key source is determined by KEYSRC<3:0> (CRYCONH<3:0>). See text for details.

3.2.3 CIPHER FEEDBACK MODE

In **Cipher Feedback (CFB)** mode (Figure 3-6 and Figure 3-7), each ciphertext block is produced by XORing the plaintext block with the encrypted version of the previous ciphertext block. As with CBC mode, an IV is provided as an initial seed for the start of the message encryption process.

Like CBC mode, CFB mode cannot be parallelized for encryption, but can be parallelized for decryption. However, CFB mode has two distinct advantages over CBC mode: low latency and resource reuse.

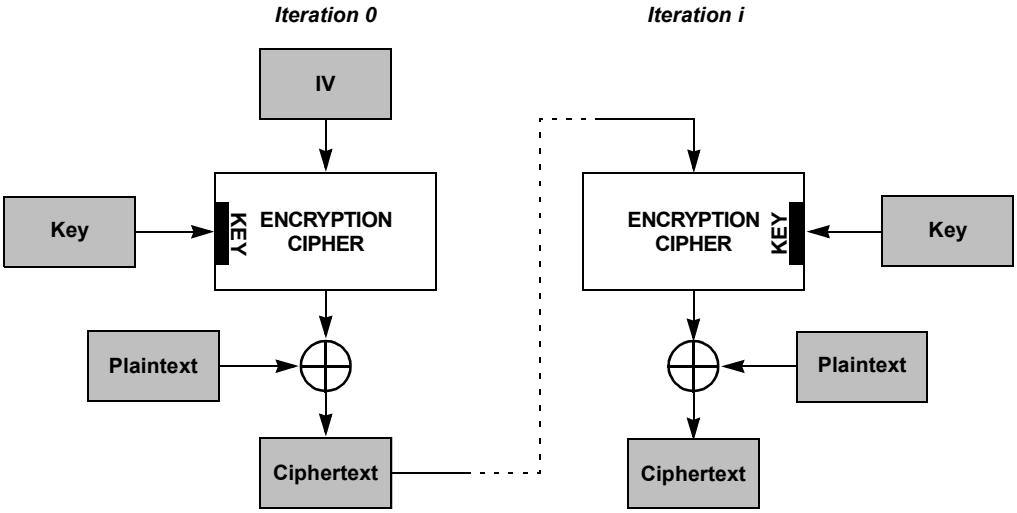
As with any of the Block Cipher mode encryption operations, the majority of the computing timing is spent in the actual encryption cipher, rather than the XOR operation. In CFB mode, only the previous ciphertext is operated on by the encryption cipher. This means that the encryption operation can be executed once the previous plaintext has been processed, but before the current plaintext is available. In this manner, the amount of time between when the current plaintext is available and when the corresponding ciphertext is computed (i.e., the latency of the encryption operation) is minimized. This can have a great impact in any real-time applications that require encryption. A similar argument applies for the decryption process.

CFB mode can also run with different feedback lengths. The most common CFB modes for AES are CFB1, CFB8 and CFB128. For DES/TDES, the most common modes are CFB1, CFB8 and CFB64. This module implements CFB128 for AES and CFB64 for DES/TDES.

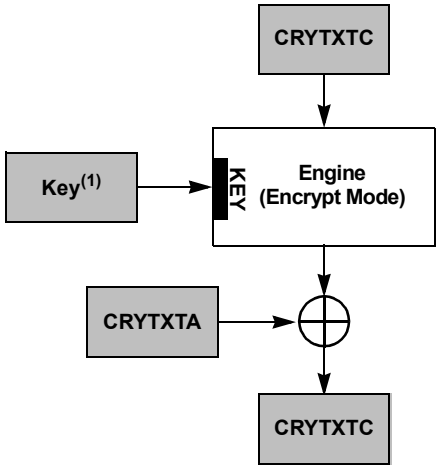
The CFB Decryption operation uses the same encryption cipher, rather than a separate decryption cipher. When implemented in software, this has the advantage of code reusability and code size. When implemented in hardware, this has the advantage of reducing the size of the hardware, assuming half-duplex operation is being used (i.e., cannot encrypt and decrypt simultaneously).

Figure 3-6: CFB Mode Encryption Operation (OPMOD<3:0> = 0000, CPHRMOD<2:0> = 010)

Algorithm: $C_i = E_K(C_{i-1}) \oplus P_i$, where $C_0 = IV$

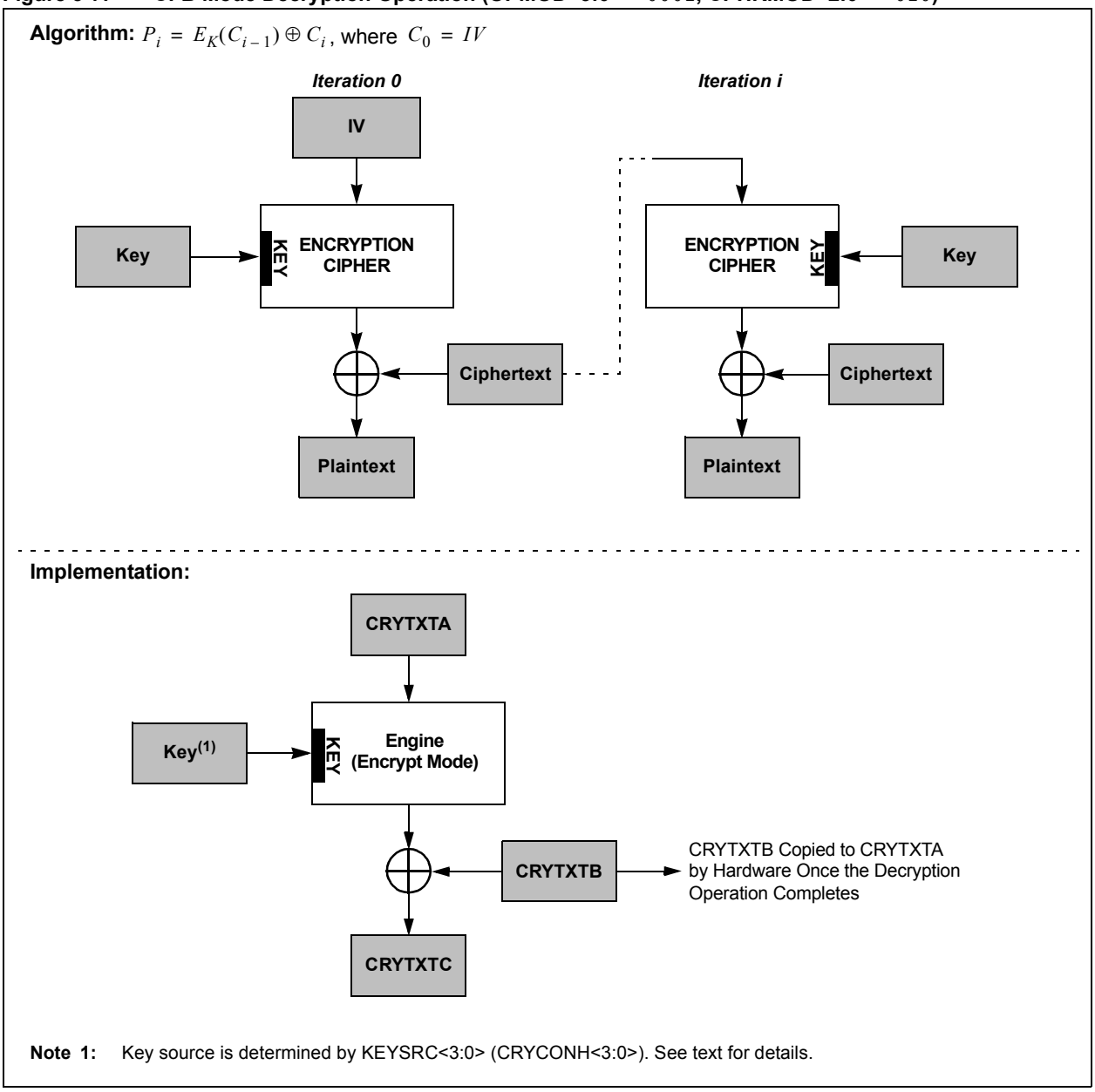


Implementation:



Note 1: Key source is determined by KEYSRC<3:0> (CRYCONH<3:0>). See text for details.

Figure 3-7: CFB Mode Decryption Operation (OPMOD<3:0> = 0001, CPHRMOD<2:0> = 010)



Note 1: Key source is determined by KEYSRC<3:0> (CRYCONH<3:0>). See text for details.

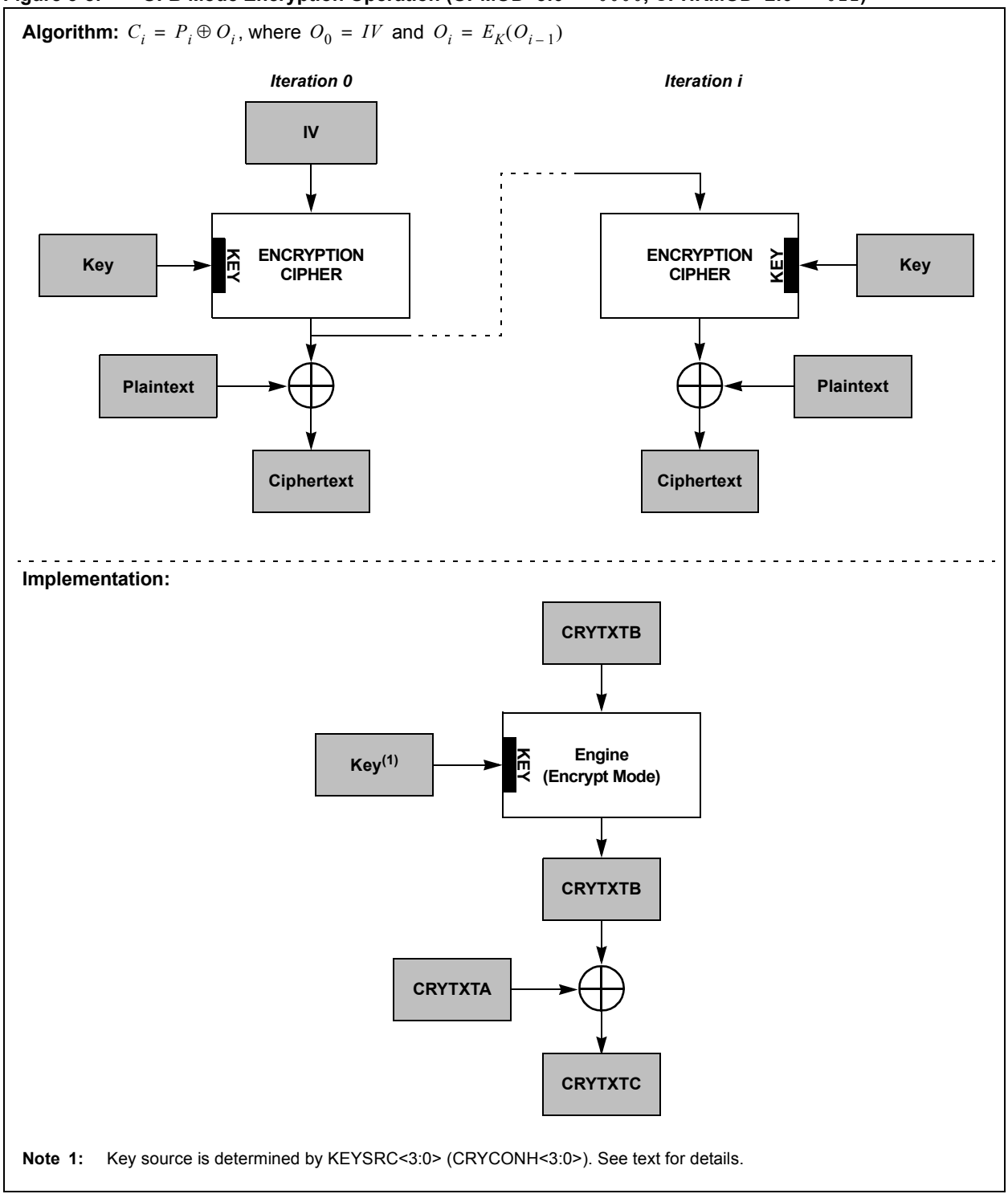
3.2.4 OUTPUT FEEDBACK MODE

In **Output Feedback (OFB)** mode, pictured in Figure 3-8 and Figure 3-9, each ciphertext block is produced by XORing the plaintext block with the encrypted version of the previous encryption cipher output (called, O_i, in the equations included in the figures). As with CBC and CFB modes, an IV is provided as an initial seed for the start of the message encryption process.

Like CBC and CFB mode, OFB mode cannot be 100% parallelized for encryption, but can be parallelized for decryption. However, OFB mode can be parallelized much more than CFB, due to the fact that each encryption operation only requires the results from the previous encryption operation and not the plaintext or ciphertext. Therefore, given the IV and the number of plaintext blocks that will be processed, it is possible to perform all of the encryption operations even before the first plaintext block is available. Once this step is completed, the ciphertext blocks can be computed in parallel.

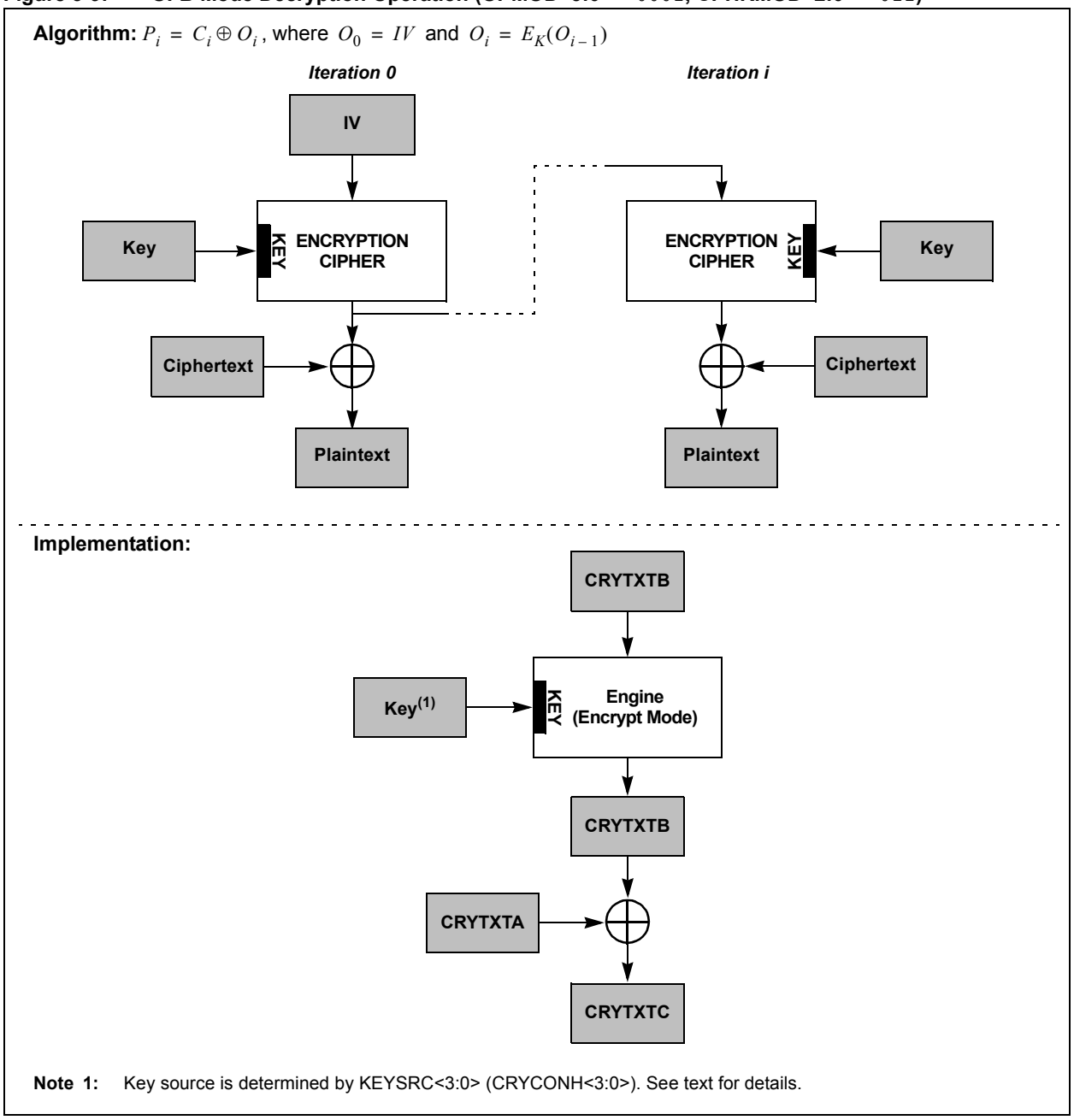
Like CFB mode, OFB mode provides low latency and the ability to reuse software or hardware resources. However, OFB mode has the advantage that the encryption and decryption operations are identical, further increasing reuse.

Figure 3-8: OFB Mode Encryption Operation (OPMOD<3:0> = 0000, CPHRMOD<2:0> = 011)



Note 1: Key source is determined by KEYSRC<3:0> (CRYCONH<3:0>). See text for details.

Figure 3-9: OFB Mode Decryption Operation (OPMOD<3:0> = 0001, CPHRMOD<2:0> = 011)



Note 1: Key source is determined by KEYSRC<3:0> (CRYCONH<3:0>). See text for details.

3.2.5 COUNTER MODE

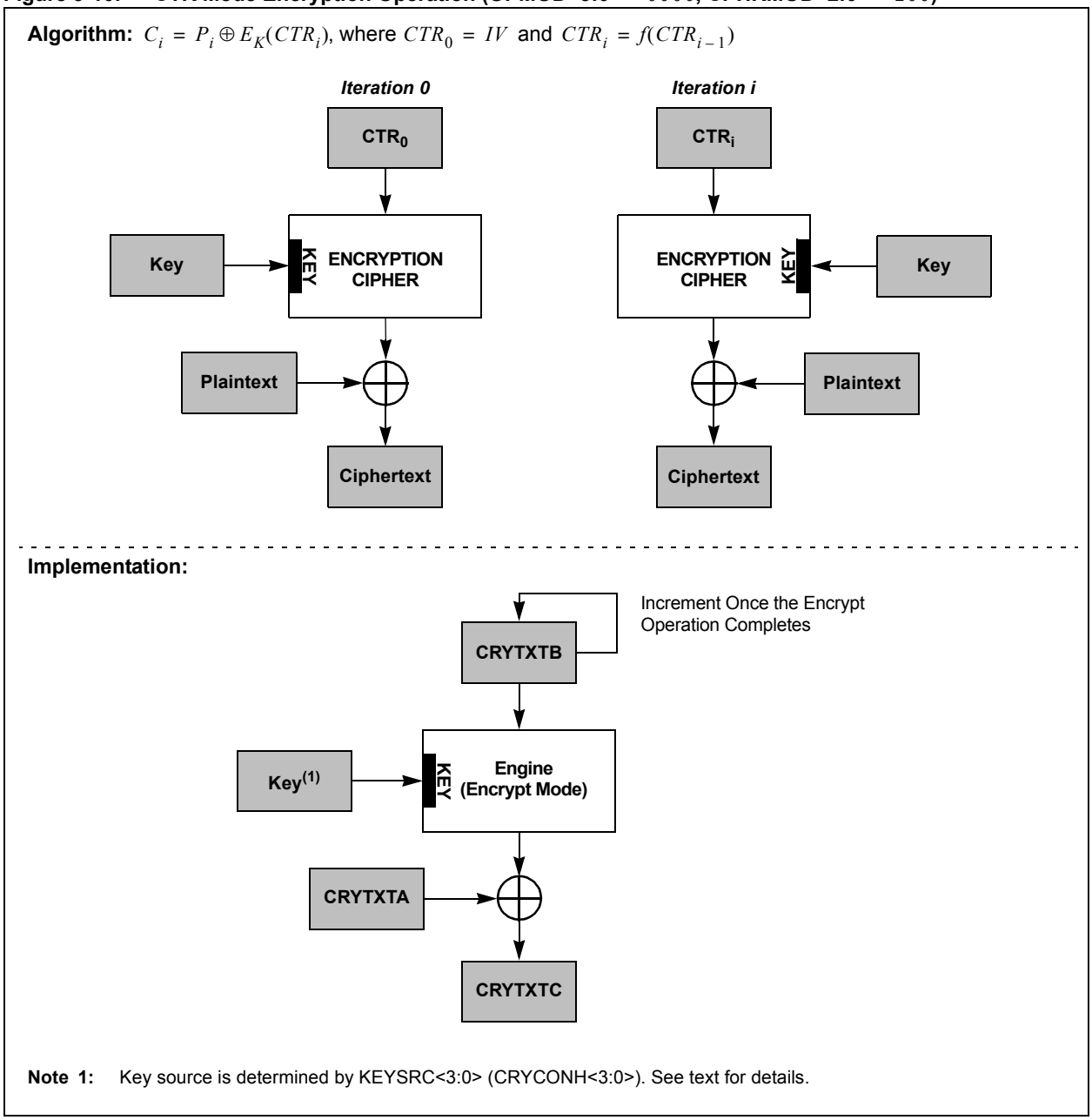
In **Counter (CTR)** mode (Figure 3-10 and Figure 3-11), each ciphertext block is produced by XORing the plaintext block with the encrypted version of a counter input. The Initial Value (IV) of the counter input serves as the IV for the message, and may be changed for each message, as with other modes.

The counter input is typically comprised of two parts: the nonce and the counter. The counter is the portion of the counter input that is changed for each block within a session, while the nonce is the portion of the counter input that changes only from session to session.

Although the term, “counter”, is used, this does not mandate the use of a true counter. Any easy-to-compute function, which is practically non-repeating (at least for a long time), may be used.

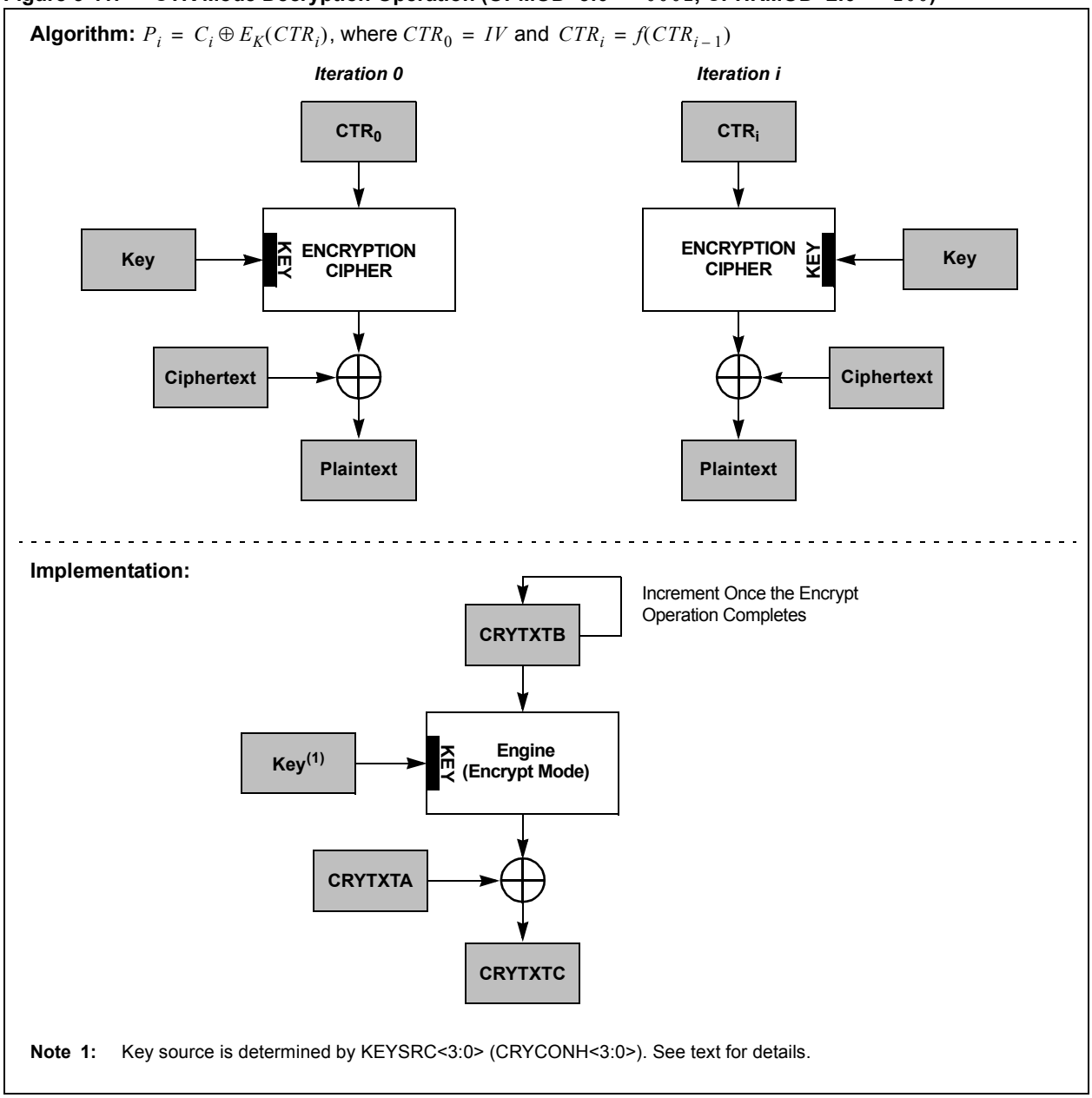
Unlike CBC, CFB and OFB modes, CTR mode may be 100% parallelized for both encryption and decryption.

Figure 3-10: CTR Mode Encryption Operation (OPMOD<3:0> = 0000, CPHRMOD<2:0> = 100)



Note 1: Key source is determined by KEYSRC<3:0> (CRYCONH<3:0>). See text for details.

Figure 3-11: CTR Mode Decryption Operation (OPMOD<3:0> = 0001, CPHRMOD<2:0> = 100)



Note 1: Key source is determined by KEYSRC<3:0> (CRYCONH<3:0>). See text for details.

3.3 Stream Ciphers

Stream Ciphers, as previously discussed, are ciphers that use a state (with a Next State function) and plaintext to compute the associated ciphertext (see Figure 3-1). In this context, the state is known as the Keystream, because it serves the function of an Encryption Key (i.e., combined with the plaintext to create the ciphertext) and because it is a continually changing value (i.e., a stream).

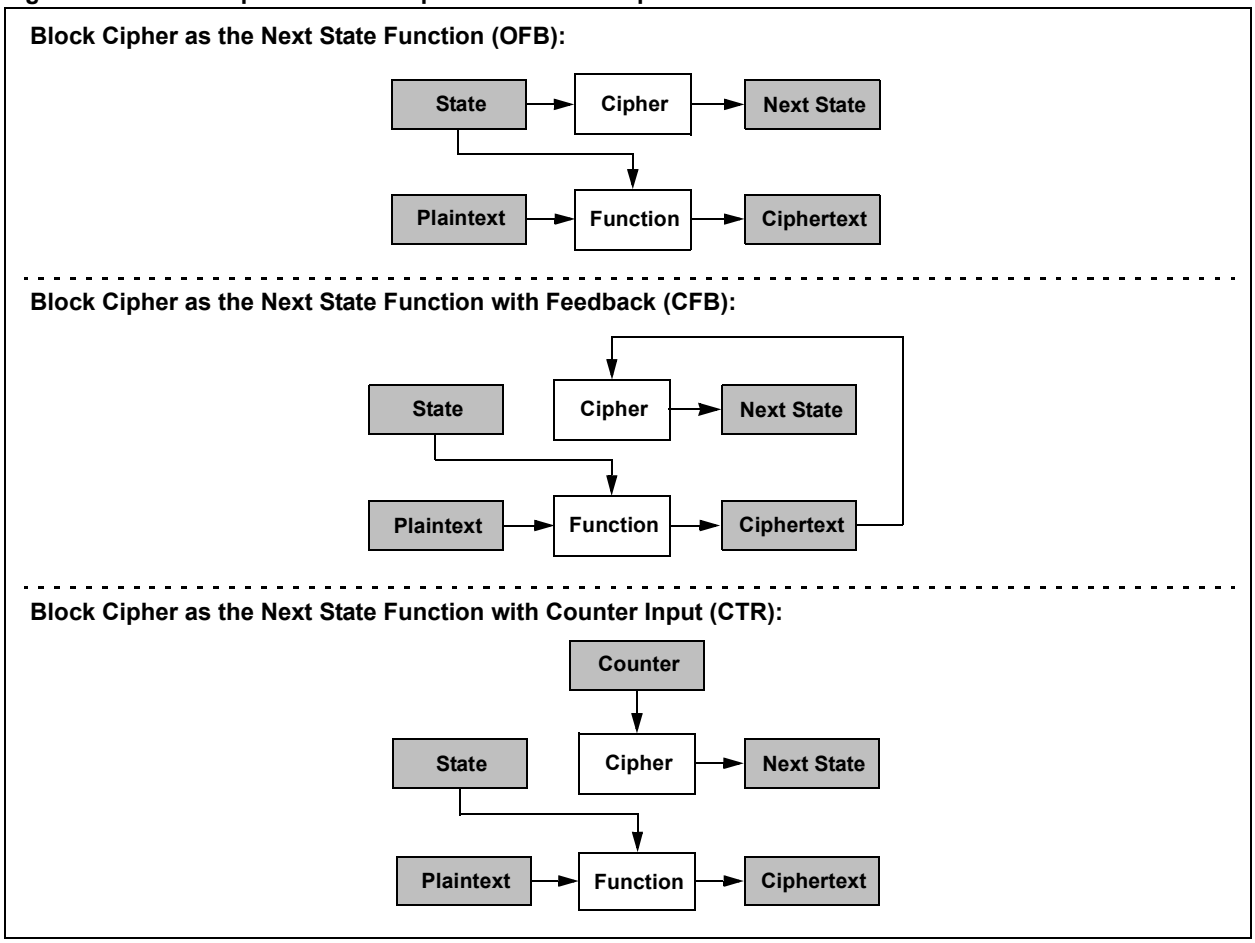
When the Keystream is generated independently of the plaintext or ciphertext (OFB and CTR modes in Figure 3-12), the Stream Cipher is known as a Synchronous Stream Cipher, because both the encryption and decryption ciphers must have the same state (Keystream) for a successful decryption operation to occur. If a piece of the ciphertext message is added or lost, then the receiver's Next State function will compute an incorrect Next State and the decryption of the next piece will fail.

If, however, the Next State is a function of the plaintext or ciphertext (CFB mode in Figure 3-12), then the Stream Cipher becomes a Self-Synchronizing Stream Cipher, because the Next State may be recovered after a suitable number of pieces of ciphertext have been received.

3.3.1 USING BLOCK CIPHERS TO CREATE STREAM CIPHERS

As discussed in **Section 3.1.1 "Block Ciphers Versus Stream Ciphers"**, Stream Ciphers consist of a Next State function and a plaintext-to-ciphertext function. In order to create a cryptographically secure Stream Cipher, it is only necessary that one of these functions be cryptographically secure. Thus, it is possible to use Block Cipher functions, which are readily available, in the creation of a secure Stream Cipher. In fact, the CFB, OFB and CTR modes, discussed previously, do exactly that. As shown in Figure 3-12, the Block Cipher is used as the Next State function, while a simple XOR function is used as the plaintext-to-ciphertext function to create a Stream Cipher that is as cryptographically secure as the underlying Block Cipher.

Figure 3-12: Examples of Block Ciphers as Stream Ciphers



3.4 Integrity Protection of Ciphers

Although the Cipher modes that we have discussed so far (both block and stream) protect privacy, they do not protect the integrity of the message. It may be possible for someone to modify an encrypted message without ever knowing the plaintext message and without the tampering being detected. For this reason, some form of integrity protection is typically provided with encryption in modern secure communications. Typically, this integrity protection is accomplished through the use of a **Message Authentication Code (MAC)**; see **Section 3.4.1 "Message Authentication Codes (MACs)"**.

When encryption is combined with a MAC to provide both privacy and authenticity (which implies integrity), the resulting algorithm is termed as, **Authenticated Encryption**.

3.4.1 MESSAGE AUTHENTICATION CODES (MACs)

A Message Authentication Code (MAC) protects the authenticity of a message and (implicitly) its integrity. A MAC algorithm takes the variable-length input message and a Secret Key, and produces a **MAC Tag** as an output. Any change to the content of the message will result in a change to the MAC Tag, thereby ensuring the integrity of the message. Since the same message with a different Secret Key will also produce a different MAC Tag, a MAC also provides protection of authenticity.

The terms, MIC and MAC, are often used interchangeably, and the difference between the two is the use of a Secret Key in a MAC. This ensures the authenticity (and the integrity) of the message, while a MIC can only ensure the integrity. In practice, MICs are themselves often encrypted for transmission, so as to provide some resistance to tampering. MACs, on the other hand, already make use of a Secret Key and do not require encryption before transmission.

MAC algorithms are commonly built using cryptographic hashes or ciphers, as described in the following sections.

3.4.2 CIPHER-BASED MACs

A cipher can be used to create a MAC Tag easily by combining the results of each block encryption operation to create a fixed-length tag. For those Block Cipher modes that already have a dependence from one encryption operation to another (CBC and CFB modes), the MAC Tag is simply the last ciphertext block in the message.

The natural inclination, when using such a MAC, is to reduce the amount of work that must be performed by encrypting the message, and then, simply using the last ciphertext block as the MAC Tag. However, it can be shown that doing this allows changing every block but the last one without detection. Therefore, when using the same cipher algorithm for both encryption and MAC Tag generation, at a minimum, different Secret Keys must be used.

Cipher-based MACs are typically named as, `<cipher>-<mode>-MAC`, as in AES-CBC-MAC.

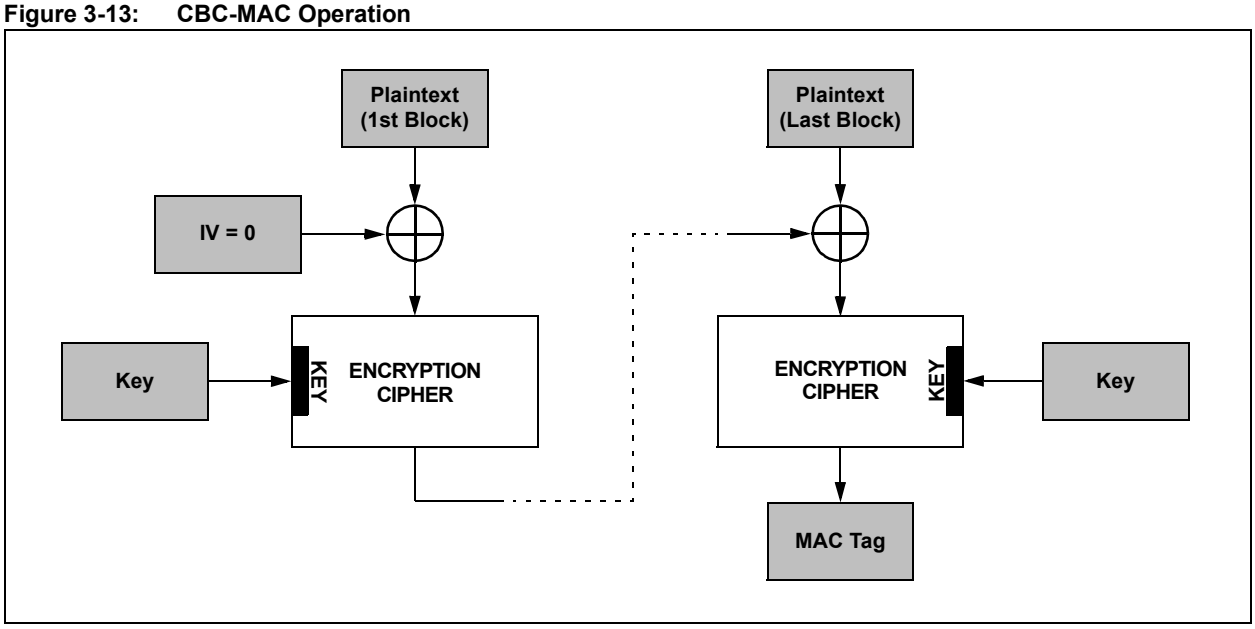
3.4.3 CBC-MAC

Figure 3-13 shows how a CBC-MAC Tag is generated from a message using a Block Cipher in CBC mode. As discussed earlier, since CBC mode naturally creates a dependence between subsequent blocks, the MAC Tag is simply the last ciphertext block.

It can be shown that given a message, and the CBC-MAC Tag corresponding to that message, it is easy to create a new message with a valid CBC-MAC Tag by simply concatenating blocks onto the old message. For this reason, CBC-MAC should only be used for fixed-length messages. For variable-length messages, other cipher-based MACs (such as CMAC) are typically used.

Note that creating a CBC-MAC Tag is identical to a CBC Encryption operation, except that the Initial Value (IV) is 0 and all intermediate ciphertext blocks are discarded.

Figure 3-13: CBC-MAC Operation



3.5 Pseudorandom Number Generation

Random numbers are very useful in cryptography for many purposes, such as generating temporary Encryption Keys (see below). Methods, such as NIST SP 800-90, may be used to implement a Pseudorandom Number Generator (PRNG). A PRNG produces a number that is deterministic, but non-repeating within a given number of generated numbers (called the *reseeding interval*). This method is opposed to a True Random Number Generator (TRNG), which produces a number that is non-deterministic, and may therefore, repeat values. (The Cryptographic Engine also includes a hardware-based TRNG, which is discussed separately in **Section 4.4.3 "True Random Number Generation"**.)

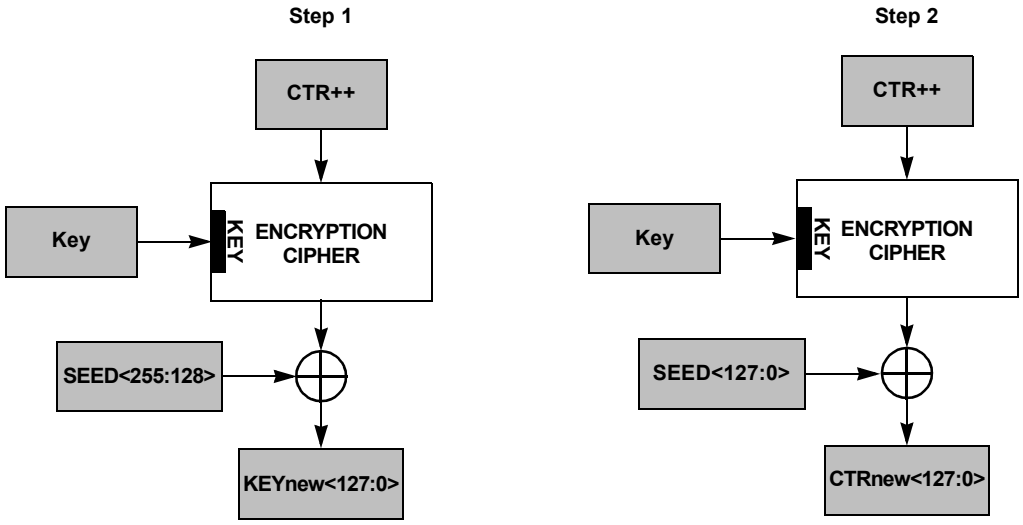
The process flow for SP 800-90 is shown in Figure 3-14 and Figure 3-15. Note that the reseeding process is actually the starting point for generating the initial PRN. A typical PRNG implementation requires a True Random Number to seed the PRNG. Once the series of PRNG numbers repeats, the reseeding operation must be performed in order to start a new sequence of PRNG numbers.

3.6 Key Generation and Wrapping

In applications that may want to provide an additional level of protection against gaining access to keys, a Session Key may be used in place of a fixed key. A Session Key can be thought of as a temporary key used for a fixed amount of time and then discarded. This provides a limited window of time under which the system must be compromised in order to gain access to a useful key. After the Session Key has expired, a new key is generated (Key Generation) by either the host or the slave in the system and then encrypted (Key Wrapping) before being transmitted.

Figure 3-14: NIST SP 800-90 PRNG Reseed Operation (128-Bit PRNG)

Algorithm:

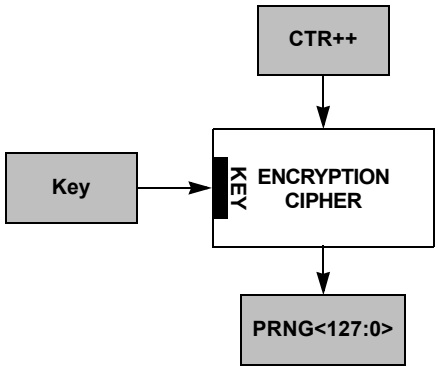


Note: Before the first reseed operation, software must initialize Key = 0 and CTR = 0. Before each reseed operation, software must reload the correct key and CTR value generated from the last reseed for generation operation.

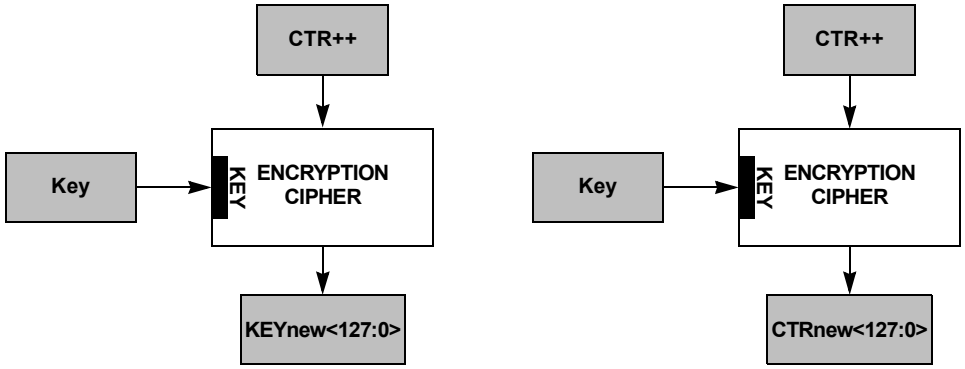
Figure 3-15: NIST SP 800-90 Generate PRNG (CTR_DRBG) Operation (128-Bit PRNG)

Algorithm:

Step 1. Key Generation



Step 2. Key Update and Reseed



Note: Before the first generate operation, software must perform a reseed operation. Subsequent reseed operations must also be performed whenever the seed interval elapses (i.e., the counter portion of CRYTXTB rolls over).

4.0 MODULE OPERATION

4.1 Enabling the Engine

The Cryptographic Engine is enabled by setting the CRYON bit. Clearing this bit disables both the DES and AES engines, as well as causing the following register bits to be held in Reset:

- CRYGO (CRYCONL<8>)
- TXTABSY (CRYSTAT<6>)
- CRYWR (CRYOTP<0>)

All other register bits and registers may be read and written while CRYON = 0.

4.2 Endianness within the Module

4.2.1 KEY AND DATA ENDIANNESS

Key and data storage within the module is byte-big-endian; that is, the left most byte of test case data in the FIPS specification is in the Least Significant Byte (LSB) of the key/data storage. Example 4-1 shows how the data for a hypothetical encryption, using CRYTXTC as the ciphertext output, would be stored.

Example 4-1: Key and Data Endianness

```
DATA:
COUNT = 0
KEY = 80000000000000000000000000000000
PLAINTEXT = 00000000000000000000000000000000
CIPHERTEXT = 0edd33d3c621e546455bd8ba1418bec8
LS BYTES OF STORED DATA:
CRYKEY<7:0> = 80
CRYTXTC<7:0> = 0E
```

4.2.2 COUNTER (CRYTXTB) ENDIANNESS

Endianness of the counter in CRYTXTB (CPHRMOD<2:0> = 100) is handled according to the endianness specified by NIST, defined as byte-big-endian, bit-little-endian. From a normal data perspective, a multibyte register might be expected to proceed in a uniform bit-little-endian fashion, incrementing uniformly, digit-by-digit, from right to left across all bytes. In the case of the NIST definition, the left most byte increments within the byte in a normal, little-endian fashion; when it rolls over, the next byte to the right begins to increment from its Least Significant bit (LSb), and so on.

The pattern is shown in Example 4-2.

Example 4-2: Incrementing CRYTXTB to Show Counter Endianness

```
       CRYTXTB     | 127   120 | 119   112 | ... | 15      8 | 7       0 |
Operation #1       | 0000_0000 | 0000_0000 | ... | 0000_0000 | 0000_0000 |
Operation #2       | 0000_0001 | 0000_0000 | ... | 0000_0000 | 0000_0000 |
Operation #3       | 0000_0010 | 0000_0000 | ... | 0000_0000 | 0000_0000 |
    :
Operation #255     | 1111_1110 | 0000_0000 | ... | 0000_0000 | 0000_0000 |
Operation #256     | 1111_1111 | 0000_0000 | ... | 0000_0000 | 0000_0000 |
Operation #257     | 0000_0000 | 0000_0001 | ... | 0000_0000 | 0000_0000 |
Operation #258     | 0000_0001 | 0000_0001 | ... | 0000_0000 | 0000_0000 |
    :
```

4.3 Key Management

It is possible to select one of multiple sources for the Secret Key used during an encryption or decryption operation. The different sources are summarized in Table 4-1 and Table 4-2, and discussed below.

4.3.1 STORED KEYS (SECURE OTP ARRAY)

The Cryptographic Engine includes a 512-bit One-Time-Programmable (OTP) memory array for the storage of cryptographic keys. To ensure their security, the contents of this memory space cannot be read accessed from outside of the Cryptographic Engine, by any means, during run time or device programming. The OTP array is thus referred to as being “programmatically secure”.

The operation of the Secure OTP Array is controlled by a separate 32-bit OTP array, referred to as Configuration Page 0 (CFGPAGE). For practical purposes, CFGPAGE can be thought of as a 32-bit Configuration register. The location and function of its control bits are shown in Register 2-5.

Note: The CFGPAGE register must not be confused with the standard device Configuration registers or Flash Configuration Words of PIC24 devices.

CFGPAGE, along with other memory locations in the Secure OTP Array, does not return to a default setting on a device Reset. Once a bit position is programmed (= 1), it cannot be cleared by any means. Leaving individual bits unprogrammed (= 0) during an OTP program operation allows them to be programmed later.

The user may preprogram up to 7 DES, or up to 4 AES Keys, into secure key storage. Available key storage and locations are determined by the Cryptographic Engine’s Operating and Key Source modes, detailed in Table 4-1 and Table 4-2.

When SKEYEN (CFGPAGE<19>) is set, stored Key #1 is used exclusively as a Key Encryption Key (KEK) to encrypt and decrypt Session Keys. It is not available for use as a general purpose Encryption/Decryption Key.

4.3.1.1 Programming an OTP Page

The OTP is programmed, 64 bits at a time, and is programmed to the page specified by KEYPG<3:0>. The data being programmed is taken from CRYTXTC<63:0>. After the OTP configuration page is programmed, the CRYREAD bit should be set and the user should wait for this bit to clear before performing any other operations using the OTP as a source. In order to start an OTP write, a programming unlock sequence must be performed before the CRYWR bit can be set. This is done by writing 55h to the NVMKEY register, immediately followed by a write of AAh to the NVMKEY register. The CRYWR bit can then be set within the few cycles following the unlock sequence. Example 4-3 shows how this can be done in assembly language. When using the MPLAB® XC16 compiler, there is a built-in function that provides this functionality: `__builtin_write_CRYOTP()`. Example 4-4 shows a complete sequence of writing a key to the OTP.

Example 4-3: Unlocking the CRYWR Bit in Assembly Language

```
mov   #0x55, W0
mov   W0, _NVMKEY
mov   #0xAA, W0
mov   W0, _NVMKEY
nop
bset  CRYOTP, #0
```

Example 4-4: Programming and Locking the OTP with 4 Keys

```
/* Select 4 different AES-128 keys */
uint8_t key1[] = {
    0x00, 0x01, 0x02, 0x03,
    0x04, 0x05, 0x06, 0x07,
    0x08, 0x09, 0x0A, 0x0B,
    0x0C, 0x0D, 0x0E, 0x0F
};

uint8_t key2[] = {
    0x10, 0x11, 0x12, 0x13,
    0x14, 0x15, 0x16, 0x17,
    0x18, 0x19, 0x1A, 0x1B,
    0x1C, 0x1D, 0x1E, 0x1F
};

uint8_t key3[] = {
    0x20, 0x21, 0x22, 0x23,
    0x24, 0x25, 0x26, 0x27,
    0x28, 0x29, 0x2A, 0x2B,
    0x2C, 0x2D, 0x2E, 0x2F
};

uint8_t key4[] = {
    0x30, 0x31, 0x32, 0x33,
    0x34, 0x35, 0x36, 0x37,
    0x38, 0x39, 0x3A, 0x3B,
    0x3C, 0x3D, 0x3E, 0x3F
};


// Enable 128-bit AES for all pages and lock all pages for modification:
// <0> SWKYDIS ----------------------------------------- 0
// <1> WRLOCK0 ..................................... 1|
// <2> WRLOCK1 ----------------------------------- 1.|
// <3> WRLOCK2 ................................... 1|.|
// <4> WRLOCK3 ------------------------------------ 1.|.|
// <5> WRLOCK4 ................................... 1|.|.|
// <6> WRLOCK5 -------------------------------- 1.|.|.|
// <7> WRLOCK6 ............................... 1|.|.|.|
// <8> WRLOCK7 ---------------------------- 1.|.|.|.|
// <9> WRLOCK8 ............................ 1|.|.|.|.|
// <10> SRCLCK ---------------------------- 0.|.|.|.|.|
// <18:11> LKYSRC ................... 00000000|.|.|.|.|.|
// <19> SKEYNEN -------------------- 0........|.|.|.|.|.|
// <21:20> KEY1TYPE ............... 10|........|.|.|.|.|.|
// <23:22> KEY3TYPE ------------- 10..|........|.|.|.|.|.|
// <25:24> KEY5TYPE ............ 10||..|........|.|.|.|.|.|
// <27:26> KEY7TYPE --------- 10..||..|........|.|.|.|.|.|
// <29:28> KEYSZRAM ....... 10||..||..|........|.|.|.|.|.|
// <30> TSTPGM ----------- 1..||..||..|........|.|.|.|.|.|
// <31> Reserved ........ 0|..||..||..|........|.|.|.|.|.|
uint32_t otpConfigReg = 0b01101010101000000000001111111110;

CRYCONH = 0;                        //Clear all registers
CRYSTAT = 0;
CRYOTP = 0;
CRYCONLbits.CRYON = 0b1;            //Turn module on
```

Example 4-4: Programming and Locking the OTP with 4 Keys (Continued)

```
/* Program Key 1 (Pages 1 and 2) */
CRYOTPbits.KEYPG = 1;         //point to page 1
memcpy((void*)&CRYTXTC0, &key1[0], 8);
__builtin_write_CRYOTP();
while(CRYOTPbits.CRYWR == 1){}

CRYOTPbits.KEYPG = 2;         //point to page 2
memcpy((void*)&CRYTXTC0, &key1[8], 8);
__builtin_write_CRYOTP();
while(CRYOTPbits.CRYWR == 1){}

/* Program Key 2 (Pages 3 and 4) */
CRYOTPbits.KEYPG = 3;         //point to page 3
memcpy((void*)&CRYTXTC0, &key2[0], 8);
__builtin_write_CRYOTP();
while(CRYOTPbits.CRYWR == 1){}

CRYOTPbits.KEYPG = 4;         //point to page 4
memcpy((void*)&CRYTXTC0, &key2[8], 8);
__builtin_write_CRYOTP();
while(CRYOTPbits.CRYWR == 1){}

/* Program Key 3 (Pages 5 and 6) */
CRYOTPbits.KEYPG = 5;         //point to page 5
memcpy((void*)&CRYTXTC0, &key3[0], 8);
__builtin_write_CRYOTP();
while(CRYOTPbits.CRYWR == 1){}

CRYOTPbits.KEYPG = 6;         //point to page 6
memcpy((void*)&CRYTXTC0, &key3[8], 8);
__builtin_write_CRYOTP();
while(CRYOTPbits.CRYWR == 1){}

/* Program Key 4 (Pages 7 and 8) */
CRYOTPbits.KEYPG = 7;         //point to page 7
memcpy((void*)&CRYTXTC0, &key4[0], 8);
__builtin_write_CRYOTP();
while(CRYOTPbits.CRYWR == 1){}

CRYOTPbits.KEYPG = 8;         //point to page 8
memcpy((void*)&CRYTXTC0, &key4[8], 8);
__builtin_write_CRYOTP();
while(CRYOTPbits.CRYWR == 1){}

/* Configure the OTP */
/* This is done last in case we are locking the OTP from writes. */

CRYOTPbits.KEYPG = 0b0000;    //point to page 0
memcpy((void*)&CRYTXTC0, (uint8_t*)&otpConfig, sizeof(otpConfig));
__builtin_write_CRYOTP();

while(CRYOTPbits.CRYWR == 1){}
CRYOTPbits.CRYREAD = 1;
while(CRYOTPbits.CRYREAD == 1){}
```

4.3.1.2 Using the OTP Key

In order to use the keys stored in the OTP, the key type programmed into the OTP configuration page must match the Encryption/Decryption mode selected by KEYMOD<1:0>. If the mode does not match, a configuration error is given and the request operation is not performed (Example 4-5).

Example 4-5: Using an OTP Key for a 128-Bit AES Encryption

```
/* Example from NIST KAT tests ECBKeySbox128.rsp */

BYTE key[] = {
    0x10, 0xa5, 0x88, 0x69,
    0xd7, 0x4b, 0xe5, 0xa3,
    0x74, 0xcf, 0x86, 0x7c,
    0xfb, 0x47, 0x38, 0x59
};

BYTE plaintext[] = {
    0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00
};

BYTE expected_results[] = {
    0x6d, 0x25, 0x1e, 0x69,
    0x44, 0xb0, 0x51, 0xe0,
    0x4e, 0xaa, 0x6f, 0xb4,
    0xdb, 0xf7, 0x84, 0x65
};

CRYCONLbits.CRYON = 1;         //Turn module on.
CRYCONHbits.KEYSRC = 0b0001;   //Select the key source (Key #1)
CRYCONLbits.OPMOD = 0b0000;    //Select the operating mode (Encryption)
CRYCONLbits.CPHRSEL = 0b1;     //Select the cipher (AES)
CRYCONLbits.CPHRMOD = 0b000;   //Select the encryption mode (ECB)
CRYCONHbits.KEYMOD = 0b00;     //Set the key strength to 128-bit

//Load the plaintext block into CRYTXTA (16 bytes for AES)
memcpy((void*)&CRYTXTA0, plaintext, 16);

CRYCONLbits.CRYGO = 0b1;       //Start the encryption
while(CRYCONLbits.CRYGO == 1){}

if(CRYSTATbits.KEYFAIL == 1){ return FAIL_KEYFAIL_SET; }
```

4.3.2 SOFTWARE KEY

When the SWKYDIS bit (CFGPAGE<0>) is '0', the Secret Key used for encryption/decryption operations is taken directly from the CRYKEY register. The KEYSRC<3:0> bits must also be '0000' (if SRCLCK is '0') or LKYSRC<7:0> must be 00h (if SRCLCK is '1').

Programming SWKYDIS (= 1) disables software writes to CRYKEY and permanently disables the software key feature. This is recommended for applications which do not use a software key. However, setting KEYSRC<3:0> to '0000' still allows CRYKEY to be selected as a key source, but only in conjunction with Session Keys. This is described in **Section 4.4.4 "Session Key Encryption"**.

Note:	Storing a software key effectively bypasses secure key storage of the device, potentially exposing the keys to a loss of secrecy. Hardware-generated keys can provide a higher level of key security.

4.3.3 KEY ENCRYPTION KEY (KEK)

The Key Encryption Key (KEK) is the key that is used to encrypt all generated Session Keys. For security, this key should never be used to encrypt or decrypt anything but the Session Keys. For this reason, the SKEYEN bit is used to enable this feature.

When SKEYEN is programmed, the KEK may only be used to encrypt or decrypt Session Keys during a Session Key Encryption or Load operation. Attempting an encryption or decryption with KEYSRC<3:0> = 0001 will result in the KEYFAIL status bit being set. In this case, no encrypt or decrypt operations are allowed until the KEYSRC<3:0> bits are changed to a valid value.

4.3.4 KEY RAM

In addition to the Secure OTP Array, select devices may also incorporate a secure, volatile memory array that may be used for key storage. This array is known as the Key RAM. Key RAM allows the user to temporarily store key data while the device is under power, rather than having the data stored permanently in the OTP array. This can allow for the use of other encryption/decryption methods when the OTP array is fully programmed, or to provide an extra layer of security by ensuring that no key data is ever persistently stored in the device.

Like the OTP array, Key RAM is not memory-mapped and is not accessible by any combination of run-time operations; it is considered to be programmatically secure.

Key RAM is organized identically to the OTP array, with 8 pages of 64 bits each. The engine selects which key store to use via the KEYPSEL bit (CRYOTP<8>). Setting KEYPSEL selects the Key RAM as the target for key programming. Key configuration for the entire array is determined by the KEYSZRAM<1:0> bits (CFGPAGE<29:28>).

Key RAM does not a have a Configuration Page 0 like the OTP array and does not have visible write lock bits. Instead, to prevent accidental overwriting of Key RAM data, each block of Key RAM has an internal write lock bit that is not accessible from software. When a block is programmed, its write lock is set; this prevents further writes to the block. All write locks are cleared when the Key RAM is erased (resulting from either a tamper event or a software-initiated wipe) or on a device POR.

Note:	Before programming the Key RAM, ensure that the RAM is in a cleared, writable state by first setting the KEYWIPE bit and then allowing the bit to clear.

4.3.4.1 Key RAM Hardware Security

Although Key RAM is not accessible by run-time operations, and therefore, programmatically secure, the devices incorporating it include one or more hardware features to enhance security.

All implementations of the Cryptographic Engine with Key RAM also implement the KEYWIPE bit (CRYCONH<4>). Setting this bit causes the entire Key RAM to be erased and all write locks to be cleared. KEYWIPE acts immediately when set, then clears to '0' on the next clock cycle. This bit can be set in software as part of a routine security process after application software exits from an encryption/decryption routine. It may also be set as the result of a security or anti-tamper routine that is incorporated into the application.

Select devices may include hardware anti-tamper features, such as a dedicated Tamper Detection pin ($\overline{\text{TMPR}}$). These features are generally controlled by Configuration bits. When enabled, tamper events may trigger an automatic wipe of Key RAM. Refer to the specific device data sheet for more information.

Table 4-1: DES/3DES Key Source Selection

Mode of Operation	KEYMOD<1:0>	KEYSRC<3:0>	Session Key Source (SKEYEN)		OTP OR RAM Array Address
			0	1	
64-Bit DES	00	0000[1]	CRYKEY<63:0>		—
		0001	DES Key #1	Key Config Error[2]	<63:0>
		0010	DES Key #2		<127:64>
		0011	DES Key #3		<191:128>
		0100	DES Key #4		<255:192>
		0101	DES Key #5		<319:256>
		0110	DES Key #6		<383:320>
		0111	DES Key #7		<447:384>
		1001	DES Key #1 (RAM)		<63:0>
		1010	DES Key #2 (RAM)		<127:64>
		1011	DES Key #3 (RAM)		<191:128>
		1100	DES Key #4 (RAM)		<255:192>
		1101	DES Key #5 (RAM)		<319:256>
		1110	DES Key #6 (RAM)		<383:320>
		1111	DES Key #7 (RAM)		<447:384>
		All Others	Key Config Error[2]		—
64-Bit, 2-Key 3DES (Standard 2-Key E-D-E/D-E-D)	01	0000[1]	CRYKEY<63:0> (1st/3rd) CRYKEY<127:64> (2nd)		—
		0001	DES Key #1 (1st/3rd) DES Key #2 (2nd)	Key Config Error[2]	<63:0> <127:64>
		0010	DES Key #3 (1st/3rd) DES Key #4 (2nd)		<191:128> <255:192>
		0011	DES Key #5 (1st/3rd) DES Key #6 (2nd)		<319:256> <383:320>
		0100	DES Key #7 (1st/3rd) DES Key #8 (2nd)		<447:384> <511:448>
		1001	DES Key #9 (1st/3rd) (RAM) DES Key #10 (2nd) (RAM)		<63:0> <127:64>
		1010	DES Key #11 (1st/3rd) (RAM) DES Key #12 (2nd) (RAM)		<191:128> <255:192>
		1011	DES Key #13 (1st/3rd) (RAM) DES Key #14 (2nd) (RAM)		<319:256> <383:320>
		1100	DES Key #15 (1st/3rd) (RAM) DES Key #16 (2nd) (RAM)		<447:384> <511:448>
		1111	Reserved[2]		—
		All Others	Key Config Error[2]		—
(Reserved)	10	xxxx	Key Config Error[2]		—

Note 1: This configuration is considered a Key Configuration Error (KEYFAIL bit is set) if SWKYDIS is also set.

2: The KEYFAIL bit (CRYSTAT<1>) is set when these configurations are selected and remains set until a valid configuration is selected.

Table 4-1: DES/3DES Key Source Selection (Continued)

Mode of Operation	KEYMOD<1:0>	KEYSRC<3:0>	Session Key Source (SKEYEN)		OTP OR RAM Array Address
			0	1	
64-Bit, 3-Key 3DES	11	0000[1]	CRYKEY<63:0> (1st Iteration) CRYKEY<127:64> (2nd Iteration) CRYKEY<191:128> (3rd Iteration)		—
		0001	DES Key #1 (1st) DES Key #2 (2nd) DES Key #3 (3rd)	Key Config Error[2]	<63:0> <127:64> <191:128>
		0010	DES Key #4 (1st) DES Key #5 (2nd) DES Key #6 (3rd)		<255:192> <319:256> <383:320>
		1001	DES Key #4 (1st) (RAM) DES Key #5 (2nd) (RAM) DES Key #6 (3rd) (RAM)		<63:0> <127:64> <191:128>
		1010	DES Key #7 (1st) (RAM) DES Key #8 (2nd) (RAM) DES Key #9 (3rd) (RAM)		<255:192> <319:256> <383:320>
		1111	Reserved[2]		—
		All Others	Key Config Error[2]		—

Note 1: This configuration is considered a Key Configuration Error (KEYFAIL bit is set) if SWKYDIS is also set.

2: The KEYFAIL bit (CRYSTAT<1>) is set when these configurations are selected and remains set until a valid configuration is selected.

Figure 4-1: DES Key OTP Page Assignment (SKEYEN = 0)

KEYMOD<1:0> = 00/01/11:

	63 32	31 0
Page 0		CFGPAGE
Page 1	DES Key #1	
Page 2	DES Key #2	
Page 3	DES Key #3	
Page 4	DES Key #4	
Page 5	DES Key #5	
Page 6	DES Key #6	
Page 7	DES Key #7(1)	
Page 8	DES Key #8(1)	

Note 1: Not valid for all modes; refer to Table 4-1 to verify availability.

Figure 4-2: DES Key OTP Page Assignment (SKEYEN = 1)

KEYMOD<1:0> = 00:

	63 32	31 0
Page 0		CFGPAGE
Page 1	Key Encryption Key #1	
Page 2	DES Key #2	
Page 3	DES Key #3	
Page 4	DES Key #4	
Page 5	DES Key #5	
Page 6	DES Key #6	
Page 7	DES Key #7	
Page 8	N/A	

KEYMOD<1:0> = 01:

	63 32	31 0
Page 0		CFGPAGE
Page 1	Key Encryption Key #1	
Page 2	Key Encryption Key #2	
Page 3	DES Key #3	
Page 4	DES Key #4	
Page 5	DES Key #5	
Page 6	DES Key #6	
Page 7	DES Key #7	
Page 8	DES Key #8	

KEYMOD<1:0> = 11:

	63 32	31 0
Page 0		CFGPAGE
Page 1	Key Encryption Key #1	
Page 2	Key Encryption Key #2	
Page 3	Key Encryption Key #3	
Page 4	DES Key #4	
Page 5	DES Key #5	
Page 6	DES Key #6	
Page 7	N/A	
Page 8	N/A	

Table 4-2: AES Key Mode/Source Selection

Mode of Operation	KEYMOD<1:0>	KEYSRC<3:0>	Key Source		OTP Address
			SKEYEN = 0	SKEYEN = 1	
128-Bit AES	00	0000[1]	CRYKEY<127:0>		—
		0001	AES Key #1	Key Config Error[2]	<127:0>
		0010	AES Key #2		<255:128>
		0011	AES Key #3		<383:256>
		0100	AES Key #4		<511:384>
		1001	AES Key #5 (RAM)		<127:0>
		1010	AES Key #6 (RAM)		<255:128>
		1011	AES Key #7 (RAM)		<383:256>
		1100	AES Key #8 (RAM)		<511:384>
		1111	Reserved[2]		—
		All Others	Key Config Error[2]		—
192-Bit AES	01	0000[1]	CRYKEY<191:0>		—
		0001	AES Key #1	Key Config Error[2]	<191:0>
		0010	AES Key #2		<383:192>
		1001	AES Key #3 (RAM)		<191:0>
		1010	AES Key #4 (RAM)		<383:192>
		1111	Reserved[2]		—
		All Others	Key Config Error[2]		—
256-Bit AES	10	0000[1]	CRYKEY<255:0>		—
		0001	AES Key #1	Key Config Error[2]	<255:0>
		0010	AES Key #2		<511:256>
		1001	AES Key #3 (RAM)		<255:0>
		1010	AES Key #4 (RAM)		<511:256>
		1111	Reserved[2]		—
		All Others	Key Config Error[2]		—
(Reserved)	11	xxxx	Key Config Error[2]		—

Note 1: This configuration is considered a Key Configuration Error (KEYFAIL bit is set) if SWKYDIS is also set.

2: The KEYFAIL bit (CRYSTAT<1>) is set when these configurations are selected and remains set until a valid configuration is selected.

Figure 4-3: AES Key OTP Page Assignment (SKEYEN = 0)

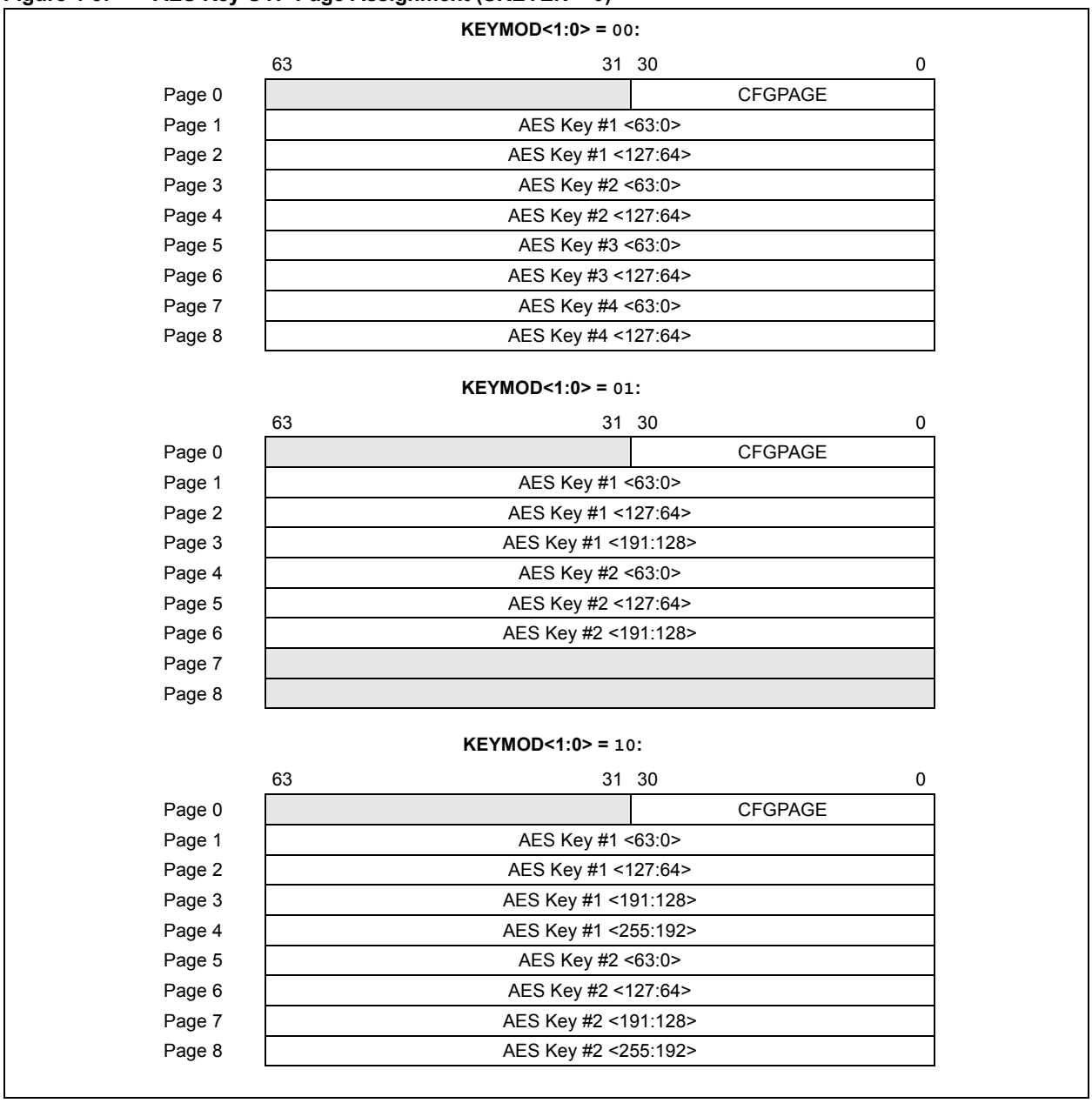
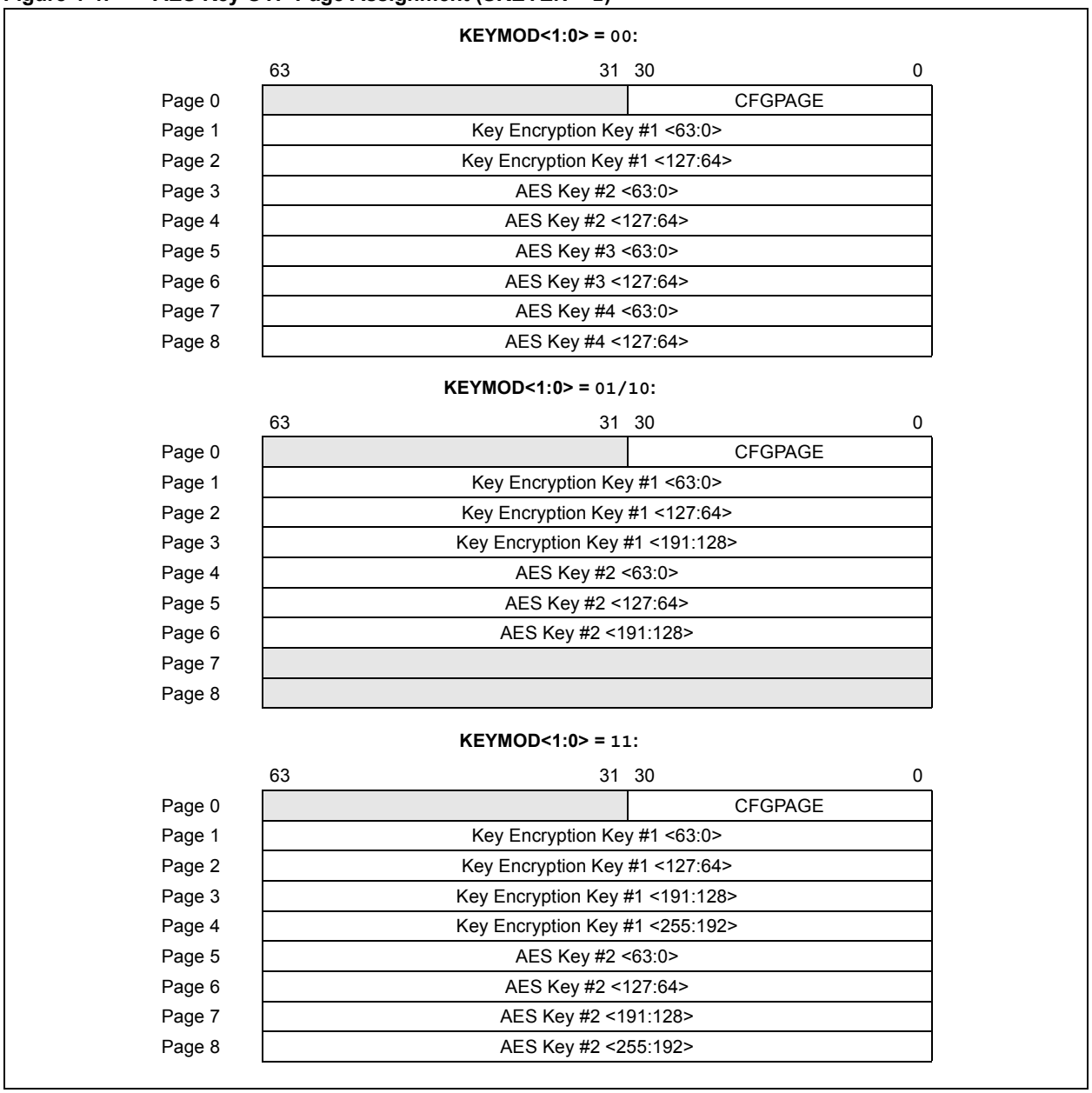


Figure 4-4: AES Key OTP Page Assignment (SKEYEN = 1)



4.4 Selecting a Mode of Operation

The Cryptographic Engine supports several modes of operation, determined by the OPMOD<3:0> bits:

- Block Encryption
- Block Decryption
- AES Decryption Key Expansion
- Random Number Generation
- Session Key Generation
- Session Key Encryption
- Session Key Loading

The OPMODx bits may be changed while CRYON is set. They should only be changed when a cryptographic operation is not being done (CRYGO = 0).

Once the encryption operation, and the appropriate and valid key configuration is selected, the operation is performed by setting the CRYGO bit. The bit is automatically cleared by hardware when the operation is complete. The CRYGO bit can also be manually cleared by software; this causes any operation in progress to terminate immediately. Clearing the bit in software also sets the CRYABRT bit (CRYSTAT<5>).

For most operations, CRYGO can only be set when an OTP operation is not being performed and there are no other error conditions. CRYREAD, CRYWR, CRYABRT, ROLLOVR, MODFAIL and KEYFAIL must all be set to '0'.

Setting CRYWR and CRYGO simultaneously will not initiate an OTP programming operation or any other operation. Setting CRYGO when the module is disabled (CRYON = 0) also has no effect.

4.4.1 BLOCK ENCRYPTION

In a block encryption operation, plaintext is loaded into one of the three Cryptographic Text registers (CRYTXTA, CRYTXTB or CRYTXTC), depending on the Cipher mode chosen. This data is encrypted using the key specified by KEYSRC<3:0> and KEYMOD<1:0>, and the mode specified by CPHRMOD<2:0>. The resulting ciphertext is placed into one of the three Text registers, depending on the mode chosen. (See **Section 3.2 "Block Ciphers"** for the specific details of each mode.)

During an encryption operation, the contents of the CRYKEY register are not disturbed.

If the CTRSIZE<6:0> bits are set to something other than 00h (typically for CTR modes), the CRYTXTB register is incremented *after* the completion of the encryption operation. The counter size within CRYTXTB is determined by the value of the CTRSIZE<6:0> bits.

Example 4-6 provides an example of a single block encryption in EBC mode, written in C. Example 4-7 provides a similar example for CBC mode encryption. Block encryptions in different Encryption and Cipher modes are similar, subject to the block and key size constraints of each mode.

Example 4-6: AES-ECB Mode Encryption (NIST Example)

```
/* The example is from the AES Multiblock Message Test (MMT) sample vectors
 * available on the NIST website
 * (http://csrc.nist.gov/groups/STM/cavp/documents/aes/aesmmt.zip)
 * Counts 3 of encrypt section of the CBCMMT128.rsp file
 */
unsigned char plaintext[] = {
0x9a, 0xc1, 0x99, 0x54, 0xce, 0x13, 0x19, 0xb3,
0x54, 0xd3, 0x22, 0x04, 0x60, 0xf7, 0x1c, 0x1e,
0x37, 0x3f, 0x1c, 0xd3, 0x36, 0x24, 0x08, 0x81,
0x16, 0x0c, 0xfd, 0xe4, 0x6e, 0xbf, 0xed, 0x2e,
0x79, 0x1e, 0x8d, 0x5a, 0x1a, 0x13, 0x6e, 0xbd,
0x1d, 0xc4, 0x69, 0xde, 0xc0, 0x0c, 0x41, 0x87,
0x72, 0x2b, 0x84, 0x1c, 0xda, 0xbc, 0xb2, 0x2c,
0x1b, 0xe8, 0xa1, 0x46, 0x57, 0xda, 0x20, 0x0e };

unsigned char key[] = {
0xb7, 0xf3, 0xc9, 0x57, 0x6e, 0x12, 0xdd, 0x0d,
0xb6, 0x3e, 0x8f, 0x8f, 0xac, 0x2b, 0x9a, 0x39 };

unsigned char iv[] = {
0xc8, 0x0f, 0x09, 0x5d, 0x8b, 0xb1, 0xa0, 0x60,
0x69, 0x9f, 0x7c, 0x19, 0x97, 0x4a, 0x1a, 0xa0 };

/* expected results:
 *  0x19, 0xb9, 0x60, 0x97, 0x72, 0xc6, 0x3f, 0x33,
 *  0x86, 0x08, 0xbf, 0x6e, 0xb5, 0x2c, 0xa1, 0x0b,
 *  0xe6, 0x50, 0x97, 0xf8, 0x9c, 0x1e, 0x09, 0x05,
 *  0xc4, 0x24, 0x01, 0xfd, 0x47, 0x79, 0x1a, 0xe2,
 *  0xc5, 0x44, 0x0b, 0x2d, 0x47, 0x31, 0x16, 0xca,
 *  0x78, 0xbd, 0x9f, 0xf2, 0xfb, 0x60, 0x15, 0xcf,
 *  0xd3, 0x16, 0x52, 0x4e, 0xae, 0x7d, 0xcb, 0x95,
 *  0xae, 0x73, 0x8e, 0xbe, 0xae, 0x84, 0xa4, 0x67
 */
unsigned char ciphertext[sizeof(plaintext)] = {0};
unsigned char i;

CRYCONLbits.CRYON = 0b1;                          //Turn module on
CRYCONHbits.KEYSRC = 0b0000;                      //Select the key source (CRYKEY)
CRYCONLbits.OPMOD = 0b0000;                       //Select the operational mode
                                                  //(Encryption)
CRYCONLbits.CPHRSEL = 0b1;                        //Select the cipher (AES)
CRYCONLbits.CPHRMOD = 0b001;                      //Select encryption mode (CBC)
CRYCONHbits.KEYMOD = 0b00;                        //Set key strength to 128-bit
memcpy((void*)&CRYKEY0, key, 16);                 //Load the key into CRYKEY
                                                  //(128-bit key in this example)
memcpy((void*)&CRYTXTB0, iv, 16);                 //Load the initial vector (IV)
                                                  //into CRYTXTB
for(i=0; i<sizeof(plaintext); i+=16)              //Loop over the data we have,
                                                  //one 16-block at a time (AES)
{
   memcpy((void*)&CRYTXTA0, plaintext + i, 16);   //Load the plaintext block
                                                  //into CRYTXTA
   CRYCONLbits.CRYGO = 0b1;                       //Start the encryption
   while(CRYCONLbits.CRYGO == 0b1){}              //Wait for encryption to
                                                  //complete
   memcpy(ciphertext + i, (void*)&CRYTXTB0, 16);  //Read the results out of
                                                  //CRYTXTB
}
```

Example 4-7: AES-CBC Mode Encryption (NIST Example)

```
/* The example is from the AES Multiblock Message Test (MMT) sample vectors
 * available on the NIST website
 * (http://csrc.nist.gov/groups/STM/cavp/documents/aes/aesmmt.zip)
 * Counts 3 of encrypt section of the CBCMMT128.rsp file
 */

unsigned char plaintext[] = {
0x9a, 0xc1, 0x99, 0x54, 0xce, 0x13, 0x19, 0xb3,
0x54, 0xd3, 0x22, 0x04, 0x60, 0xf7, 0x1c, 0x1e,
0x37, 0x3f, 0x1c, 0xd3, 0x36, 0x24, 0x08, 0x81,
0x16, 0x0c, 0xfd, 0xe4, 0x6e, 0xbf, 0xed, 0x2e,
0x79, 0x1e, 0x8d, 0x5a, 0x1a, 0x13, 0x6e, 0xbd,
0x1d, 0xc4, 0x69, 0xde, 0xc0, 0x0c, 0x41, 0x87,
0x72, 0x2b, 0x84, 0x1c, 0xda, 0xbc, 0xb2, 0x2c,
0x1b, 0xe8, 0xa1, 0x46, 0x57, 0xda, 0x20, 0x0e };

unsigned char key[] = {
0xb7, 0xf3, 0xc9, 0x57, 0x6e, 0x12, 0xdd, 0x0d,
0xb6, 0x3e, 0x8f, 0x8f, 0xac, 0x2b, 0x9a, 0x39 };

unsigned char iv[] = {
0xc8, 0x0f, 0x09, 0x5d, 0x8b, 0xb1, 0xa0, 0x60,
0x69, 0x9f, 0x7c, 0x19, 0x97, 0x4a, 0x1a, 0xa0 };

unsigned char ciphertext[sizeof(plaintext)];
unsigned char i;

CRYCONbits.ON = 0b1;                      //Turn module on
CRYCONbits.KEYSRC = 0b0000;               //Select the key source (CRYKEY)
CRYCONbits.OPMOD = 0b0000;                //Select the operational mode
                                          //(Encryption)
CRYCONbits.CPHRSEL = 0b1;                 //Select the cipher (AES)
CRYCONbits.CPHRMOD = 0b001;               //Select the encryption mode (CBC)
CRYCONbits.KEYMOD = 0b00;                 //Set the key strength to 128-bit
memcpy(CRYKEY, key, 16);                  //Load the key into CRYKEY
                                          //(128-bit key in this example)
memcpy(CRYTXTB, iv, 16);                  //Load the initial vector (IV)
                                          //into CRYTXTB

for(i=0; i<sizeof(plaintext); i+=16)      //Loop over the data we have,
                                          //one 16-block at a time (AES)
{
    memcpy(CRYTXTA, plaintext + i, 16);   //Load the plaintext block
                                          //into CRYTXTA (16-bytes for AES)
    CRYCONbits.START = 0b1;               //Start the encryption
    while(CRYCONbits.START == 0xb1){}     //Wait for encryption to complete
    memcpy(ciphertext + i, CRYTXTB, 16);  //Read the results out of CRYTXTB
                                          //(16-bytes for AES)
}
```

4.4.2 BLOCK DECRYPTION

In a block decryption operation, ciphertext is loaded into one of the three Text registers (CRYTXTA, CRYTXTB or CRYTXTC), depending on the Cipher mode chosen. This data is decrypted using the key specified by KEYSRC<3:0> and KEYMOD<1:0>, and the mode specified with the CPHRMOD<3:0> bits. The resulting plaintext is placed into one of the three Text registers, depending on the mode chosen. (See **Section 3.2 "Block Ciphers"** for the specific details of each mode.)

During a decryption operation, the contents of the CRYKEY register are not disturbed.

If CTRSIZE<6:0> are set to something other than 00h (typically for AES-CTR mode), the CRYTXTB register is incremented *after* the completion of the encryption operation. The counter size within CRYTXTB is determined by the value of the CTRSIZE<6:0> bits.

Note:	It is the responsibility of the user to ensure key sources, key lengths, Data mode and initial counter value match between the encryption and decryption operations.

4.4.2.1 Generating an AES Decryption Key from a Round Key

When a new key is to be used for an AES-ECB or an AES-CBC Decryption operation (including Session Key Loading), an AES Decryption Key Expansion operation must be performed before the first decryption operation. This operation is performed by selecting the key to be used for the subsequent decryption operations (including writing the key into CRYKEY if a software key is to be used) and then performing an AES Decryption Key Expansion operation. Note that the contents of the CRYTXTn registers are irrelevant, as they will be overwritten during the operation. Once the Decryption Key is expanded, this operation does not need to be performed again while AES-ECB or AES-CBC Decryption, or Session Key Loading operations are being performed with the same key.

The contents of the CRYKEY register are overwritten during an AES Decryption Key Expansion operation. If a software key is being used, switching from AES-ECB or AES-CBC Decryption operations to some other mode will, therefore, require rewriting the CRYKEY register. AES Decryption Key Expansion mode is only valid when CPHRSEL = `1` and CPHRMOD<2:0> = `00x`. Any other settings will result in the MODFAIL bit being set.

Example 4-8: Calculating a Decryption Key from an Encryption Key

```
/* This example comes from the FIPS-197 standard, Appendix A.1
 * (http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf).
 */
unsigned char encryption_key[] = {
0x2b, 0x7e, 0x15, 0x16,
0x28, 0xae, 0xd2, 0xa6,
0xab, 0xf7, 0x15, 0x88,
0x09, 0xcf, 0x4f, 0x3c };

CRYCONLbits.CRYON = 0b1;                    //Turn module on
CRYCONHbits.KEYSRC = 0b0000;                //Select the key source (CRYKEY)
CRYCONLbits.OPMOD = 0b0010;                 //Select the operational mode
CRYCONLbits.CPHRSEL = 0b1;                  //Select the cipher (AES)
                                            //(AES decryption key generation)
CRYCONHbits.KEYMOD = 0b00;                  //Set the key strength (128-bit key)

memcpy((void*)&CRYKEY0, encryption_key, 16); //Load the key into CRYKEY
                                            //(128-bit key in this example)
CRYCONLbits.CRYGO = 0b1;                    //Start the encryption
while(CRYCONLbits.CRYGO == 0b1){}           //Wait for encryption
                                            //to complete

/* The module is now ready to perform decryption operations. CRYKEY
 * has been updated with the decryption key. You may now perform decryption
 * operations. If you need to switch keys, you will need to recalculate the
 * decryption key if you use the AES engine to calculate it for you.
 * The CRYKEY register is write only so the value of the decryption key
 * can't be read out.
*/
```

4.4.3 TRUE RANDOM NUMBER GENERATION

The Cryptographic Engine includes a hardware-based True Random Number Generator (TRNG). The module generates a 128-bit random number from a non-deterministic seed. The TRNG is triggered by setting OPMOD<3:0> to the appropriate value ('1010' or '1011') and setting the CRYGO bit.

The TRNG can be configured to store the result in CRYTXTA or in either half (<255:128> or <127:0>) of CRYKEY. When the OPMOD<3:0> bits are '1010', the TRN is written to CRYTXTA.

When the OPMOD<3:0> bits are '1011', the TRN is written to CRYKEY; the value of SKEYSEL determines if the TRN is written to the upper half (SKEYSEL = 1) or lower half (SKEYSEL = 0) of CRYKEY.

Example 4-9: Generating a Random Number

```
CRYCONLbits.CRYON = 0b1;          /* Turn module on */
CRYCONLbits.OPMOD = 0b1010;       /* Select to generate a random number */
                                  /* and store in CRYTXTA
CRYCONLbits.CRYGO = 1;            /* Start the process */

while(CRYCONLbits.CRYGO == 1){}   /* Wait until the module is done */

/* The random number is now located in CRYTXTA. */
```

4.4.4 SESSION KEY ENCRYPTION

In a Session Key Encryption operation, a software or hardware-generated Session Key is encrypted using the current algorithm, key length and Cipher mode. This Session Key is placed into the CRYKEY register and then encrypted using the Key Encryption Key (KEK), with the encrypted Session Key being written into the appropriate CRYTXTn register(s) (depending on the Cipher mode).

Because the Session Key is stored in CRYKEY like a software key, the SWKYDIS bit CANNOT be set for Session Key Encryption to work.

Encryption of Session Keys is not allowed in OFB Cipher mode. Setting OPMODE<3:0> = 111x and CPHRMOD<2:0> = 011 will result in the MODFAIL bit being set.

Note:	Do not set SWKYDIS while SKEYEN is also set; this will permanently disable Session Key operations.

4.4.5 SESSION KEY LOADING

In a Session Key Loading operation, the encrypted Session Key found in the appropriate CRYTXTn register (depending on the Cipher mode) is decrypted using the Key Encryption Key (KEK) and then written into either the lower (SKEYSEL = 0) or upper (SKEYSEL = 1) 128 bits of CRYKEY.

If the Session Key being loaded into CRYKEY is 192 bits or 256 bits, two decryption operations must be performed. The first operation loads CRYKEY<127:0> (by clearing SKYSEL) and performs a KEK decryption/loading operation. The second operation loads CRYKEY<255:128> (by setting SKEYSEL) and performs the KEK decryption/loading operation. When decrypting/loading a 128-bit Session Key, only one operation is required; SKEYSEL is cleared for this operation.

Once the Session Key Loading (Decryption) is complete, software must select the CRYKEY register as the source of the Encryption Keys, by writing KEYSRC<3:0> to '0000', before the decrypted Session Key can be used to encrypt or decrypt data.

If the Session Key Loading feature is enabled (SKEYEN = 1), the Key Encryption Key may be used ONLY for Session Key Encryption and Decryption. In this case, the hardware will disable the use of the Key Encryption Key for ANY encryption or decryption operations outside of the generated Session Key. This prevents malicious software from simply decrypting an encrypted Session Key into the CRYTXTA register.

Because the Session Key is a software key, the SWKYDIS bit cannot be set for Session Key Encryption to work. However, SWKYDIS only disables software writes to CRYKEY, so Session Key Loading operations will still work on devices when SWKYDIS is set.

Loading of Session Keys is not allowed in OFB Cipher mode. Setting OPMOD<3:0> = 111x and CPHRMOD<2:0> = 011 will result in the MODFAIL bit being set.

Note:	Session Key Loading is not available in ECB or CBC modes with a key size greater than 128 bits.

4.4.6 TESTING THE KEY SOURCE CONFIGURATION

The validity of the key source configuration can always be tested by writing the appropriate register bits and then reading the KEYFAIL register bit. No operation needs to be started to perform this check; the module does not even need to be enabled.

4.4.7 VERIFYING PROGRAMMED KEYS

To maintain key security, the Secure OTP Array has no provision to read back its data to any user-accessible memory space in any operating mode. Therefore, there is no way to directly verify programmed data. The only method for verifying that the keys have been programmed correctly is to perform an encryption operation with a known plaintext/ciphertext pair for each programmed key.

4.5 Operation Times

Table 4-3 shows the base operation times for the different operations that can be performed. Keep in mind that these operations are performed without CPU intervention and that the device will keep executing instructions while a cryptographic operation is being performed.

Note that Session Key Encryption is merely a subset of a general encryption operation and Session Key Loading is merely a subset of a general decryption operation.

Table 4-3: Approximate Operation Cycle Count

Mode	Clock Cycles (Approximate)	
	Per Block	Additional Load/Unload
DES Encryption/Decryption	10(1)	2
3DES Encryption/Decryption	26(1)	2
128-Bit AES Encryption/Decryption	219(2,3)	32
192-Bit AES Encryption/Decryption	275(2,3)	32
256-Bit AES Encryption/Decryption	299(2,3)	32
DES 64-Bit Session Key Encryption	10	2
DES 2x 64-Bit Session Key Encryption	10	2
DES 3x 64-Bit Session Key Encryption	20	4
AES 128-Bit Session Key Encryption (128-Bit KEK)	219	32
AES 128-Bit Session Key Encryption (192-Bit KEK)	275	32
AES 128-Bit Session Key Encryption (256-Bit KEK)	299	32
AES 192-Bit/256-Bit Session Key Encryption (128-Bit KEK)	438	48
AES 192-Bit/256-Bit Session Key Encryption (192-Bit KEK)	550	48
AES 192-Bit/256-Bit Session Key Encryption (256-Bit KEK)	598	48
DES 64-Bit Session Key Loading	10	2
DES 2x 64-Bit Session Key Loading	10	2
DES 3x 64-Bit Session Key Loading	20	4
AES 128-Bit Session Key Loading (128-Bit KEK)	219(3)	32
AES 128-Bit Session Key Loading (192-Bit KEK)	275(3)	32
AES 128-Bit Session Key Loading (256-Bit KEK)	299(3)	32
AES 192-Bit/256-Bit Session Key Loading (128-Bit KEK)	438(3)	48
AES 192-Bit/256-Bit Session Key Loading (192-Bit KEK)	550(3)	48
AES 192-Bit/256-Bit Session Key Loading (256-Bit KEK)	598(3)	48

Note 1: 64-bit block.

2: 128-bit block.

3: Does not include the cycles required for initialization for AES Decryption operations after switching keys.

4.6 Interrupts

The Cryptographic Engine generates four interrupts to indicate the occurrence of key events:

- Cryptographic Operation Done (CRYDONIF)
- OTP Operation Complete (KEYSTRIF)
- CRYTXTA Empty (Free) (CRYFREEIF)
- CRYTXTB Rollover Event (CRYROLLIF)

Several interrupts may be triggered by more than one condition. Users may need to use software context, or to poll other bits within the CRYSTAT and CRYOTP registers, to determine the exact nature of the interrupt.

4.6.1 OPERATION DONE INTERRUPT

Setting the DONEIE bit (CRYCONL<11>) causes an interrupt to be generated whenever the current cryptographic operation completes. The interrupt is only generated when the CRYGO bit is cleared by hardware; manually clearing CRYGO to abort an operation will not generate an interrupt.

Polling the CRYGO bit can be used to verify the interrupt source. Polling the CRYABRT bit (CRYSTAT<5>) may be used to confirm the software termination of an operation, if necessary.

4.6.2 OTP OPERATION COMPLETE INTERRUPT

Setting the OTPIE bit (CRYOTP<6>) causes an interrupt to be generated whenever the current OTP programming or read operation completes.

The CRYBSY bit (CRYSTAT<7>) is set by hardware whenever any OTP operation is being performed and is automatically cleared when the operation is complete. The CRYREAD and CRYWR bits (CRYOTP<5,0>) indicate when an OTP read or program operation is under way; they are similarly set and cleared automatically. CRYREAD and CRYBSY bits also become set briefly on a device Power-on Reset (POR), and are cleared after the initial read of the array has been completed. Verifying that all three bits are set to '0' confirms that no OTP operation is under way.

4.6.3 CRYTXTA FREE INTERRUPT

Setting the FREEIE bit (CRYCONL<10>) causes an interrupt to be generated whenever the CRYTXTA register has consumed all of its data for an operation and is free to be written with new data (either plaintext or ciphertext). This interrupt occurs only for ECB and CBC mode operations. In all other modes, CRYTXTA is used through the end of, or near the end of, the operation.

The TXTABSY bit (CRYSTAT<6>) indicates the status of CRYTXTA. The bit remains set while unprocessed data remains in the register; it is automatically cleared by hardware when all data is processed.

TXTABSY is only valid when ECB operations, CBC Encryption, CFB Decryption or generate Session Key operations are being performed. For all other cases, CRYTXTA should be considered in use whenever CRYGO is set.

Note:	Separate interrupts are not provided for the status of CRYTXTB and CRYTXTC. These registers are always assumed to be busy whenever the CRYGO bit is set.

4.6.4 CRYTXTB ROLLOVER INTERRUPT

Setting the ROLLIE bit (CRYCONL<12>) causes an interrupt to be generated whenever a CRYTXTB rollover occurs. Naturally, this only occurs in operating modes where CRYTXTB is used as a counter (CPHRMOD<2:0> = 100). The interrupt also does not occur when CRYTXTB is used as a single bit counter (CTRSIZE<6:0> = 00h).

The ROLLOVR bit (CRYSTAT<4>) becomes set when a CRYTXTB rollover event has occurred.

5.0 OPERATION DURING SLEEP AND IDLE MODES

5.1 Operation During Sleep Modes

Whenever the device enters any Sleep or Deep Sleep mode, all operations are halted and all engine state machines are reset. This feature helps to preserve the integrity of any data being encrypted or decrypted by discarding any intermediate text that might be used to break the key.

Any OTP programming operations under way when a Sleep mode is entered are also halted. Depending on what is being programmed, this may result in permanent loss of a memory location or potentially the use of the entire Secure OTP Array. Users are advised to perform OTP programming only when entry into power-saving modes is disabled.

Note:	OTP programming errors, regardless of the source, are *not* recoverable errors. Users should ensure that all foreseeable interruptions to the programming operation, including device interrupts and entry into power-saving modes, are disabled.

5.2 Operation During Idle Mode

When the CRYSIDL bit (CRYCONL<13>) is '0', the engine will continue any ongoing operations without interruption when the device enters Idle mode.

When CRYSIDL is '1', the module behaves as if in Sleep modes.

5.3 Peripheral Module Disable (PMD) Register

The Peripheral Module Disable (PMD) registers provide a method to disable the Cryptographic Engine by stopping all clock sources supplied to the module. When the module is disabled by setting the CRYMD control bit (located in the PMDx register), the peripheral is in a minimum power consumption state. The control and status registers associated with the peripheral will also be disabled, so writes to those registers will have no effect, and read values will be invalid.

The module is only enabled if the CRYMD bit is cleared.

6.0 EFFECTS OF A RESET

To maintain the security of the Cryptographic Engine, the behavior of the module during Reset states is different than most other dsPIC33/PIC24 modules.

As with most dsPIC33/PIC24 peripherals, all bits in the control registers are reset to their indicated default states on any device Reset. In addition, most bits are also reset whenever the module is disabled, using the Peripheral Module Disable bit (i.e., CRYMD = 1). A few additional bits are also reset when the module is turned off (CRYON = 0). These additional Reset conditions help to ensure that any cryptographic operations in progress are terminated on any deactivation of the module.

Page 0 of the Secure OTP Array contains configuration information for the Cryptographic Engine's run-time operation. After a device Reset, the information in Page 0 must be read and loaded to configure the Cryptographic Engine. During this interval, many features are not accessible. After a POR Reset, the CRYREAD bit is automatically set to start an OTP read operation; the Cryptographic Engine is unavailable until the operation is complete. CRYREAD is automatically cleared by hardware when the read operation is complete. At this point, the Cryptographic Engine is available for operations. Waking from Sleep follows the same sequence.

During an OTP read, other flag bits behave in different ways:

- The CRYREAD (as mentioned) and CRYBSY bits are both set (= 1) during the initial OTP read, and are automatically cleared when the read is done.
- The PGMFAIL and KEYFAIL bits are cleared (= 0) during the initial OTP read; they assume their appropriate values (based on the selected key and OTP program configurations) when the read is done.
- The TSTPGM bit is cleared (= 0) during the initial OTP read; it assumes the current value of the PGMTST bit when the read is done.

7.0 REGISTER MAPS

A summary of the memory-mapped registers and data spaces associated with the Cryptographic Engine are shown in Table 7-1.

Note: CFGPAGE is not mapped into data memory space, so it is not shown here. See Register 2-5 for details on its structure.

Table 7-1: Cryptographic Engine Register Map

Register	Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Resets
CRYCONH	—	CTRSIZE6	CTRSIZE5	CTRSIZE4	CTRSIZE3	CTRSIZE2	CTRSIZE1	CTRSIZE0	SKEYSEL	KEYMOD1	KEYMOD0	KEYWIPE	KEYSRC3	KEYSRC2	KEYSRC1	KEYSRC0	0000
CRYCONL	CRYON	—	CRYSIDL	ROLLIE	DONEIE	FREEIE	—	CRYGO	OPMOD3	OPMOD2	OPMOD1	OPMOD0	CPHRSEL	CPHRMOD2	CPHRMOD1	CPHRMOD0	0000
CRYSTAT	—	—	—	—	—	—	—	—	CRYBSY	TXTABSY	CRYABRT	ROLLOVR	—	MODFAIL	KEYFAIL	PGMFAIL	--00
CRYOTP	—	—	—	—	—	—	—	KEYPSEL	PGMTST	OTPIE	CRYREAD	KEYPG3	KEYPG2	KEYPG1	KEYPG0	CRYWR	--00
CRYKEY	Cryptographic Key Register (256 bits wide, write-only)																xxxx xxxx(1)
CRYTXTA	Cryptographic Text Register A (128 bits wide)																xxxx xxxx(2)
CRYTXTB	Cryptographic Text Register B (128 bits wide)																xxxx xxxx(2)
CRYTXTC	Cryptographic Text Register C (128 bits wide)																xxxx xxxx(2)

Legend: — = unimplemented, read as '0'; x = unknown or undefined value. Reset values are shown in hexadecimal. Relative sizes of data spaces are not shown to scale.

Note 1: Reset value is 32 bytes of 'xx'; however, the actual Reset value cannot be read.

2: Reset value is 16 bytes of 'xx'.

8.0 SELECTED REFERENCES

For more information on cryptography and data security, please see the web site for the Computer Security Resource Center at the National Institute of Standards and Technology (NIST):

http://csrc.nist.gov/groups/STM/cavp/

The web site provides complete information on AES (FIPS-197) and Triple DES (NIST SP 800-67), including test vectors for validating candidate algorithms. Information on other Data Encryption Standards is also available here.

Microchip Technology Inc. also provides a cryptographic software library for its dsPIC33/PIC24 DSCs and microcontrollers. This library includes an API that allows application developers to use the Cryptographic Engine with a minimum of additional code development. This software library is available on CD-ROM (Part Number SW300052) at **www.microchipdirect.com**.

Note:	All Microchip Technology Inc. products containing cryptographic security features are subject to licensing restrictions and federal export regulations. Please contact Microchip Technology Inc. for more information.

9.0 REVISION HISTORY

Revision A (September 2013)

Original version of this document.

Revision B (April 2015)

Updates **Section 1.0 "Introduction"** and Figure 1-1 to include newly added hardware features.

Updates text to add Key RAM:

- Adds **Section 4.3.4 "Key RAM"**
- Modifies Register 2-4 to add the KEYPSEL bit at CRYOTP<8>; also adds a new Footnote 1 to the register and renumbers all existing footnotes accordingly
- Modifies the OTP Configuration Page (Register 2-5) to add the KEYSZRAM<1:0> bits at CFGPAGE<29:28>
- Modifies Table 4-1 and Table 4-2 to include the combinatorial options for Key RAM key storage
- Updates Figure 4-1 through Figure 4-3 to reflect changes in OTP page availability in some Cryptographic modes

Updates text to include Key RAM security features:

- Adds **Section 4.3.4.1 "Key RAM Hardware Security"**
- Modifies Register 2-1 to add the KEYWIPE bit at CRYCONH<4>

Updates text to add True Random Number Generation:

- Adds **Section 4.4.3 "True Random Number Generation"**
- Updates the definition of the SKEYSEL bit (Register 2-1) to include TRNG storage
- Modifies Register 2-2 to include TRNG options in the OPMOD<3:0> bits field
- Amends **Section 3.5 "Pseudorandom Number Generation"** to include a reference to TRNG

Other updates:

- Amends **Section 4.4.5 "Session Key Loading"** to update the description of SKEYSEL in Session Key Loading
- Replaces Example 4-4 entirely with new example code

Other minor typographic changes and corrections throughout the document.

Note the following details of the code protection feature on Microchip devices:

- Microchip products meet the specification contained in their particular Microchip Data Sheet.
- Microchip believes that its family of products is one of the most secure families of its kind on the market today, when used in the intended manner and under normal conditions.
- There are dishonest and possibly illegal methods used to breach the code protection feature. All of these methods, to our knowledge, require using the Microchip products in a manner outside the operating specifications contained in Microchip's Data Sheets. Most likely, the person doing so is engaged in theft of intellectual property.
- Microchip is willing to work with the customer who is concerned about the integrity of their code.
- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of their code. Code protection does not mean that we are guaranteeing the product as "unbreakable."

Code protection is constantly evolving. We at Microchip are committed to continuously improving the code protection features of our products. Attempts to break Microchip's code protection feature may be a violation of the Digital Millennium Copyright Act. If such acts allow unauthorized access to your software or other copyrighted work, you may have a right to sue for relief under that Act.

Trademarks

ISBN: 978-1-63277-315-9

QUALITY MANAGEMENT SYSTEM
CERTIFIED BY DNV
═ ISO/TS 16949 ═

Microchip received ISO/TS-16949:2009 certification for its worldwide headquarters, design and wafer fabrication facilities in Chandler and Tempe, Arizona; Gresham, Oregon and design centers in California and India. The Company's quality system processes and procedures are for its PIC® MCUs and dsPIC® DSCs, KEELOQ® code hopping devices, Serial EEPROMs, microperipherals, nonvolatile memory and analog products. In addition, Microchip's quality system for the design and manufacture of development systems is ISO 9001:2000 certified.

Worldwide Sales and Service

AMERICAS

Corporate Office
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 480-792-7200
Fax: 480-792-7277
Technical Support:
http://www.microchip.com/support
Web Address:
www.microchip.com

Atlanta
Duluth, GA
Tel: 678-957-9614
Fax: 678-957-1455

Austin, TX
Tel: 512-257-3370

Boston
Westborough, MA
Tel: 774-760-0087
Fax: 774-760-0088

Chicago
Itasca, IL
Tel: 630-285-0071
Fax: 630-285-0075

Cleveland
Independence, OH
Tel: 216-447-0464
Fax: 216-447-0643

Dallas
Addison, TX
Tel: 972-818-7423
Fax: 972-818-2924

Detroit
Novi, MI
Tel: 248-848-4000

Houston, TX
Tel: 281-894-5983

Indianapolis

Noblesville, IN
Tel: 317-773-8323
Fax: 317-773-5453

Los Angeles
Mission Viejo, CA
Tel: 949-462-9523
Fax: 949-462-9608

New York, NY
Tel: 631-435-6000

San Jose, CA
Tel: 408-735-9110

Canada - Toronto
Tel: 905-673-0699
Fax: 905-673-6509

ASIA/PACIFIC

Asia Pacific Office
Suites 3707-14, 37th Floor
Tower 6, The Gateway
Harbour City, Kowloon

Hong Kong
Tel: 852-2943-5100
Fax: 852-2401-3431

Australia - Sydney
Tel: 61-2-9868-6733
Fax: 61-2-9868-6755

China - Beijing
Tel: 86-10-8569-7000
Fax: 86-10-8528-2104

China - Chengdu
Tel: 86-28-8665-5511
Fax: 86-28-8665-7889

China - Chongqing
Tel: 86-23-8980-9588
Fax: 86-23-8980-9500

China - Dongguan
Tel: 86-769-8702-9880

China - Hangzhou
Tel: 86-571-8792-8115
Fax: 86-571-8792-8116

China - Hong Kong SAR
Tel: 852-2943-5100
Fax: 852-2401-3431

China - Nanjing
Tel: 86-25-8473-2460
Fax: 86-25-8473-2470

China - Qingdao
Tel: 86-532-8502-7355
Fax: 86-532-8502-7205

China - Shanghai
Tel: 86-21-5407-5533
Fax: 86-21-5407-5066

China - Shenyang
Tel: 86-24-2334-2829
Fax: 86-24-2334-2393

China - Shenzhen
Tel: 86-755-8864-2200
Fax: 86-755-8203-1760

China - Wuhan
Tel: 86-27-5980-5300
Fax: 86-27-5980-5118

China - Xian
Tel: 86-29-8833-7252
Fax: 86-29-8833-7256

ASIA/PACIFIC

China - Xiamen
Tel: 86-592-2388138
Fax: 86-592-2388130

China - Zhuhai
Tel: 86-756-3210040
Fax: 86-756-3210049

India - Bangalore
Tel: 91-80-3090-4444
Fax: 91-80-3090-4123

India - New Delhi
Tel: 91-11-4160-8631
Fax: 91-11-4160-8632

India - Pune
Tel: 91-20-3019-1500

Japan - Osaka
Tel: 81-6-6152-7160
Fax: 81-6-6152-9310

Japan - Tokyo
Tel: 81-3-6880- 3770
Fax: 81-3-6880-3771

Korea - Daegu
Tel: 82-53-744-4301
Fax: 82-53-744-4302

Korea - Seoul
Tel: 82-2-554-7200
Fax: 82-2-558-5932 or
82-2-558-5934

Malaysia - Kuala Lumpur
Tel: 60-3-6201-9857
Fax: 60-3-6201-9859

Malaysia - Penang
Tel: 60-4-227-8870
Fax: 60-4-227-4068

Philippines - Manila
Tel: 63-2-634-9065
Fax: 63-2-634-9069

Singapore
Tel: 65-6334-8870
Fax: 65-6334-8850

Taiwan - Hsin Chu
Tel: 886-3-5778-366
Fax: 886-3-5770-955

Taiwan - Kaohsiung
Tel: 886-7-213-7828

Taiwan - Taipei
Tel: 886-2-2508-8600
Fax: 886-2-2508-0102

Thailand - Bangkok
Tel: 66-2-694-1351
Fax: 66-2-694-1350

EUROPE

Austria - Wels
Tel: 43-7242-2244-39
Fax: 43-7242-2244-393

Denmark - Copenhagen
Tel: 45-4450-2828
Fax: 45-4485-2829

France - Paris
Tel: 33-1-69-53-63-20
Fax: 33-1-69-30-90-79

Germany - Dusseldorf
Tel: 49-2129-3766400

Germany - Munich
Tel: 49-89-627-144-0
Fax: 49-89-627-144-44

Germany - Pforzheim
Tel: 49-7231-424750

Italy - Milan
Tel: 39-0331-742611
Fax: 39-0331-466781

Italy - Venice
Tel: 39-049-7625286

Netherlands - Drunen
Tel: 31-416-690399
Fax: 31-416-690340

Poland - Warsaw
Tel: 48-22-3325737

Spain - Madrid
Tel: 34-91-708-08-90
Fax: 34-91-708-08-91

Sweden - Stockholm
Tel: 46-8-5090-4654

UK - Wokingham
Tel: 44-118-921-5800
Fax: 44-118-921-5820

01/27/15