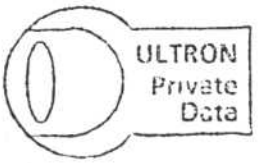
CRYPTO-ENGINE™ OVERVIEW

Ultron Labs Corporation
4423 Fortran Court
San Jose, Ca. 95134

SYSTEM DESCRIPTION
Version 7.7

Created: March 14, 1986
Revised: September 22, 1986
October 23, 1986
December 30, 1986
January 9, 1987
January 15, 1987
February 13, 1987
March 28, 1987
April 9, 1987
May 18, 1987



1.0 Introduction

(U) This document provides the System Description for Crypto-Engine™. The Crypto-Engine™ product, which belongs to the Project INSPIRE family of products, is being developed as part of the Commercial COMSEC Endorsement Program (CCEP).

1.1 References

(U) The following documents form a part of this specification to the extent referenced herein. Where conflicts in specifications occur, the contents of this document shall take precedent.

- Crypto-Engine™ Software Program Specification (SPS), ADRL item Y231-2, SS002, Document No. 0N408083,
- Software Development Plan, ADRL item Y231-4, SS004, Document No. 0N408116,
- Crypto-Engine™ Interface Specification, Y200-2, SC002, Document No. 0N397096,
- Crypto-Engine™ Security Fault Analysis, ADRL item Y300-4, SE002, Document No. 0N408067.

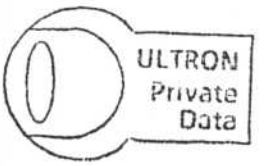
1.2 Key Features

- CMOS Technology, 2 micron Silicon Gate
- Single chip Security Solution
- CMOS operation and TTL Compatible Outputs
- Implements High-Grade NSA Algorithm
- High Performance Operation
- Optimized for DMA Applications
- Pipelined Architecture
- Automatic and Commanded Self-Test Capability
- Full Duplex Data Paths
- Internal Loopback (redundancy) Capability
- Standard Peripheral Interface Signals
- Single Phase Clock
- Static Operation: DC - 12 MHz
- Full Temperature range: 0 to +70 degree C
- Supply Current: 200mA typical @ 12 MHz
- Supply Voltage: 5.0V - 6.0V @ 12 MHz.
- Low Power Consumption
  Normal Operation: 200mA @ 5V, 12 MHz
  Idle Mode: 5mA @ 2.5V, 12 MHz
  Power Down Mode: 5µA @ 2.0V
- Package 68-Pin Grid-Array
- PC-Board mountable within Host equipment
- Three (3) Cryptographic Operating Modes
- Message Authentication Code (MAC) Mode
- Asynchronous Operation up to 50 Mbit/second

1.2.1 Security Features

- Encryption and Decryption
- Authentication
- Key Stream Generation
- Key Variable Zeroization
- Key Variable Handling and Storage
- Key Variable Update
- Key Variable Distribution
- Secondary and Split Variable Handling
- Randomization
- Initialization

- Alarm and Alarm Check
- Cryptographic Control
- Cryptologic Spoof Protection
- Cryptographic Bypass
- Cryptographic Ignition Key
- Cryptographic Synchronization



1.3 Overview

(U) What is the Crypto-Engine™? Conceptually, the Crypto-Engine™ is a self-contained cryptographic system capable of being implemented on a single silicon chip. The Crypto-Engine™ is separated into two major sections: the *algorithm* and the *control*. The algorithm section is implemented as a VLSI chip known as the Algorithm Data Path Chip (ADPC). The control section is implemented as a VLSI chip known as the Control Processor Chip (CPC). The ADPC and CPC are securely assembled in a custom *dual-cavity* carrier; the result being the Crypto-Engine™.

(U) The design goals for the Crypto-Engine™ are straight forward. Make it simple to use, versatile, fast, secure, and most of all, inexpensive. These traits allow the Crypto-Engine™ to be used in a large number of ystems and represent a revolution in the application of high-level cryptology to mass production.

(U) The Crypto-Engine™ can be separated into several sections for the purpose of describing its operation. There are basically eight functional areas to the Crypto-Engine™ (see Figure 1). They are:

- Control Unit
- Encode Algorithm Unit
- Decode Algorithm Unit
- Internal Loopback Compare Units
- Red Input Port
- Red Output Port
- Black Input Port
- Black Output Port

1.4 General Description

(U) At the heart of any cryptographic system is an *algorithm*. This is a high-grade algorithm supplied by the National Security Agency (NSA) intended for use in Government electronic equipment. The details of the algorithm are classified.

(U) Although the algorithm is complex, it lends itself to high speed VLSI implementation. The Crypto-Engine™ implements the pipelining and parallelism inherent in the iterative version of the algorithm. Fortunately, data throughput can be traded for gate complexity without impacting functionality. What kind of speeds are we talking about? Using a 12 megahertz clock rate, 50 million bits per second of data can be processed by the Crypto-Engine™. Data throughput in this range allows the chip to be used in just about any system.

(U) The ease of use of the Crypto-Engine™ is demonstrated by its external interface. Four dedicated Data Ports are provided, two of which are inputs, the other two outputs. One of each type is used for *red* data, the other for *black*. Thus there is clear red-black separation and direct support for full duplex operation. In addition, each port has its own two-wire handshake and can load or unload data asynchronously at rates exceeding that of the algorithm itself. For maximum applicability, all these ports are designed to be TTL compatible and the output ports are tri-stated.

(U) In addition to a simple and versatile data-port structure, the Crypto-Engine™ provides a TTL compatible Command Port and a 2-wire Serial Key Port (Transmit and Receive). The Crypto-Engine™ also implements in hardware the most important *modes of operation* (including Bypass).

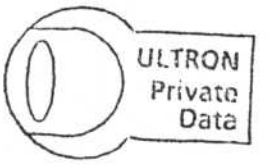
(U) Therefore, the design goals of simplicity, versatility, speed, and security are quite evident in the very structure of the Crypto-Engine™.

1.4.1 System Definition

(U) With the two major Crypto-Engine™ system components (ADPC and CPC) in mind, the following Systems Engineering documents should be referenced. The first level document, which defines the Crypto-Engine™ system specifications and system level security requirements, is the Crypto-Engine™ Theory of Compliance Plan, ADRL item Y300-1, SC001, Document #0N397097. The Theory of Compliance document defines the functions performed by the Crypto-Engine™ system as a whole. Also, it provides a

high level description of the system components and functions performed by each component.

(U) Security requirements allocated to, and security critical functions applicable to, the Crypto-Engine™ program is the domain of the Software Requirements Specification, ADRL item Y231-1, SS001, Document #0N408082. Definition of Crypto-Engine™ hardware functions subject to Security Fault Analysis (SFA) are described in Crypto-Engine™ Security Fault Analysis-Task A, ADRL item Y300-4, SE002, Document #0N408067. Definition of verification of normal intended cryptographic functions are described in the Crypto-Engine™ Cryptographic Verification Plan, ADRL item Y300-6, SE004, Document #0N408063. The detail keying scheme implemented within the Crypto-Engine™ can be found in the INSPIRE Key Management Plan (KMP), ADRL item S042-1, CD001, Document #397094. Finally, the Crypto-Engine™ Interface Specification, Y200-2, SC002, provides the Crypto-Engine™ system interface definition, from an electrical, environmental and functional (embeddable product) application point of view.



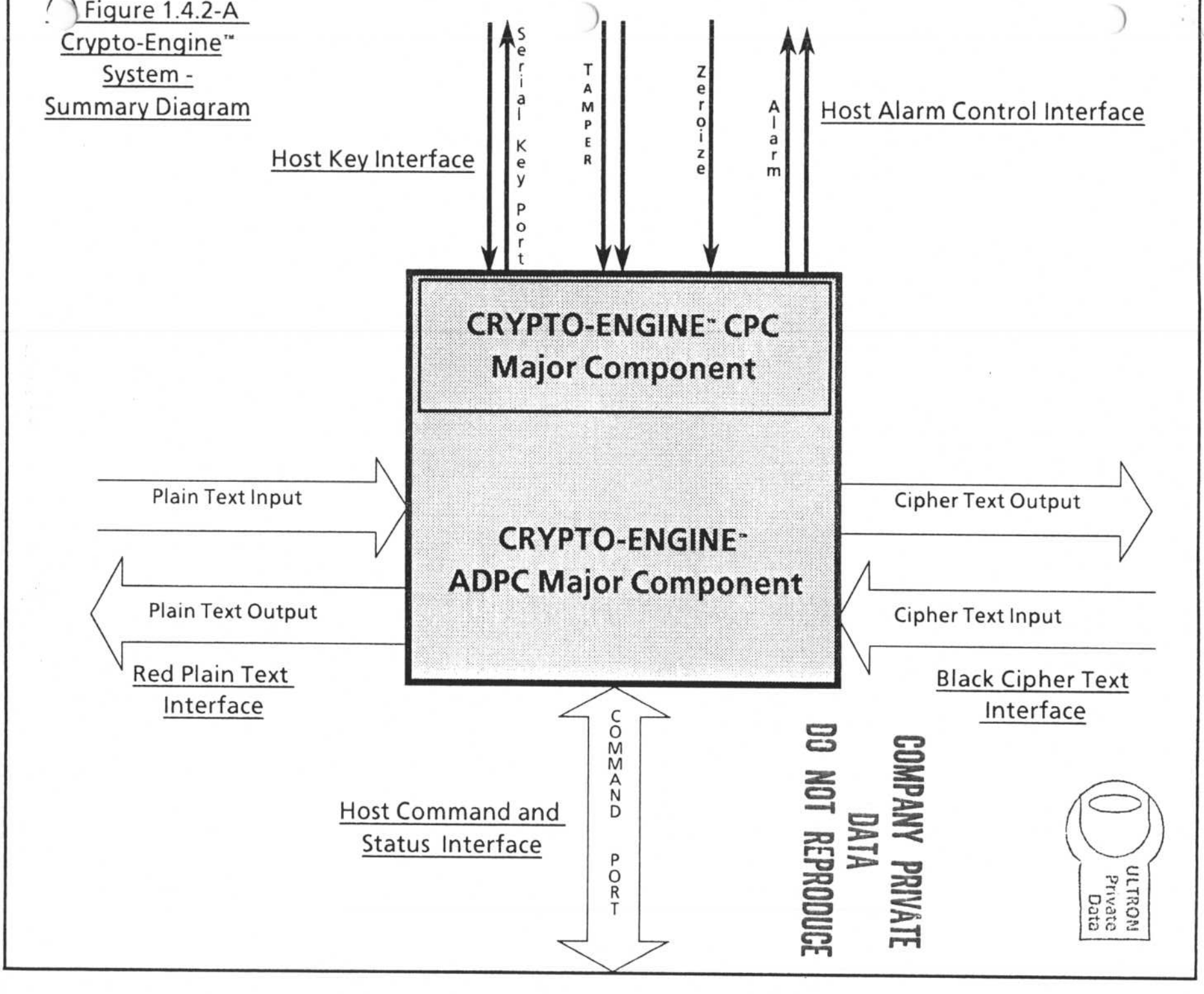
1.4.2 High Level System Diagram

(U) Figure 1.4.2-A shows the Crypto-Engine™ system appearance to a *host* system. Application of the Crypto-Engine™ system in a *host* is beyond the scope of this document.

(U) For this document's system summary purposes, the *host* system's COMSEC boundary includes the entire Crypto-Engine™ system. Therefore, the whole Crypto-Engine™ system is shown highlighted in Figure 1.4.2-A.

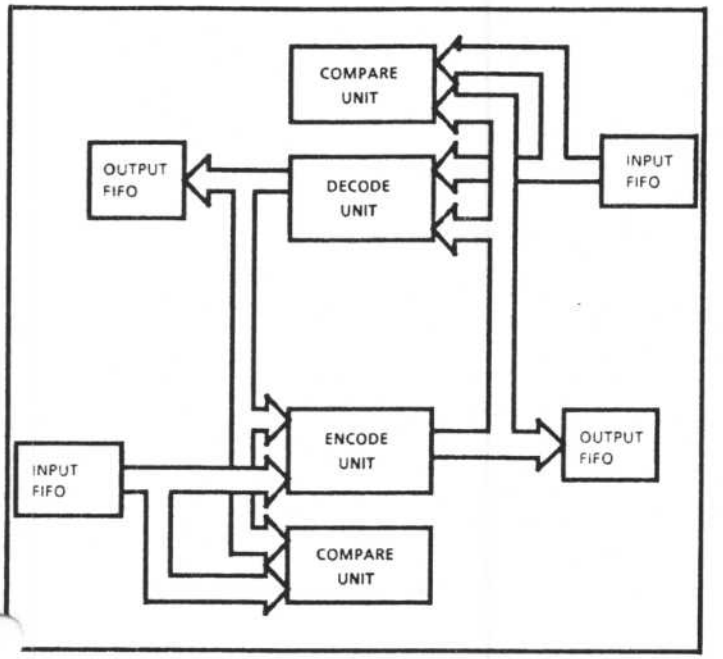
(U) It should be noted that the COMSEC boundary extends beyond the Crypto-Engine™ system because the entire suite of functional security requirments are not implemented by the Crypto-Engine™ system alone. Many of the *host's* functional security requirements are met by the Crypto-Engine™ system. Those not implemented by the Crypto-Engine™ system include portions of the anti-tamper, TEMPEST, key storage, key distribution, and anti-spoof *host* security requirments.

Figure 1.4.2-A
Crypto-Engine™
System -
Summary Diagram



1.4.1 Internal Interface

(U) The Crypto-Engine™ implements an *internal loopback* capability for checking of the algorithm data path. This unit compares the results from the two algorithm units on a bit-by-bit basis. If the compare is not successful, the data is not released and an alarm condition is signalled to the alarm interface pins.



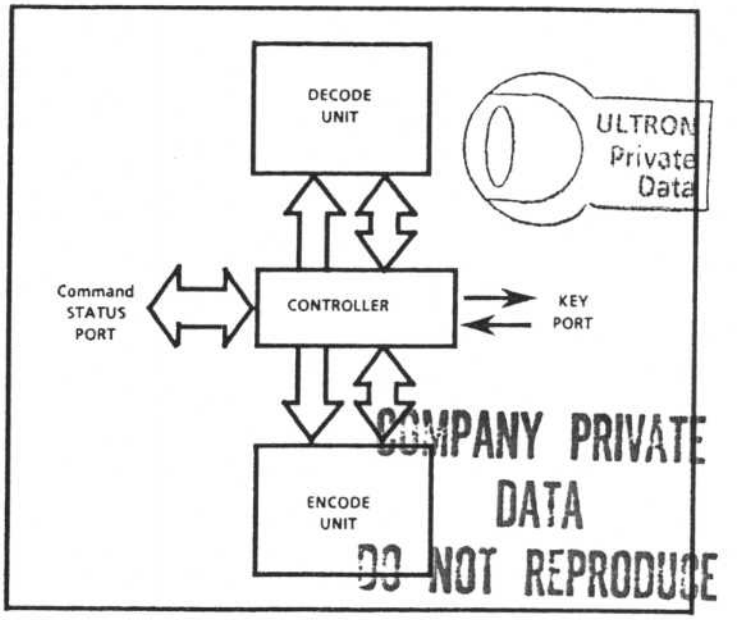
(U) The Crypto-Engine™ contains a high performance microcontroller (INTEL 80C52 type), known as the Control Processor Chip (CPC). The microcontroller functions include handling the Command Port and Serial Key Port. The microcontroller has complete access to both the algorithm units for complete cryptographic control (i.e., loading keying material, checking status, and use of the algorithm for enciphering or deciphering data, etc.).

(U) The Encode and Decode sections of the Crypto-Engine™ implement *multiple* modes of the NSA high-grade algorithm and are *independent* allowing simultaneous operation.

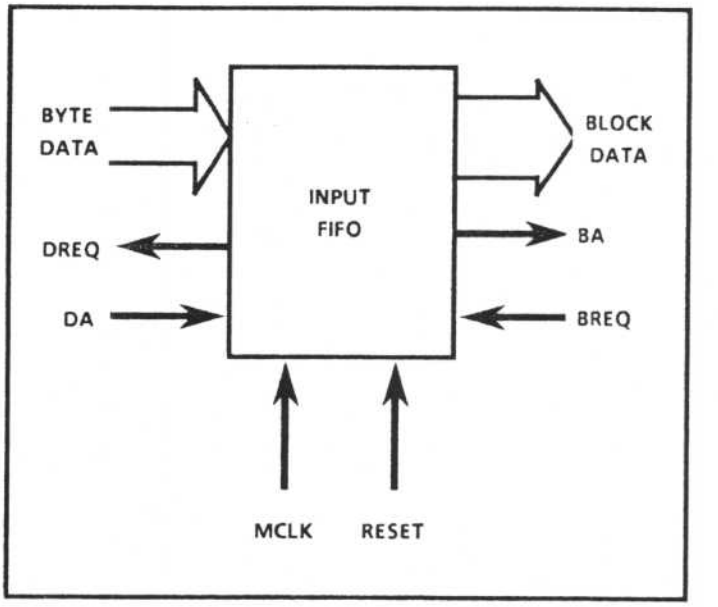
1.4.2 External Interface

(U) The Crypto-Engine™ *interface* is optimized for performance and ease of connection to commonly available circuitry, such as DMA controllers. The data paths are independent and implement a 64-bit wide FIFO buffer on each port.

(U) The input FIFO has an 8-bit wide interface for receiving external data and a 64-bit wide interface for

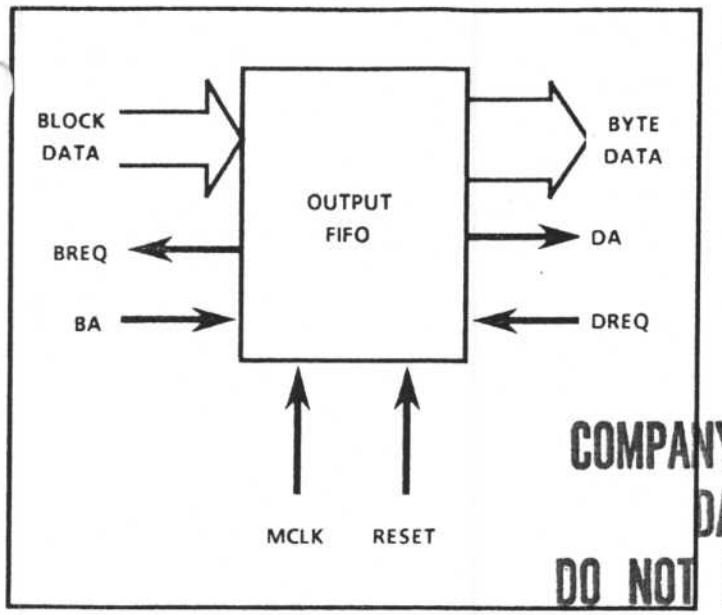


transmitting data to the particular algorithm unit. The input FIFO provides asynchronous control signals of Data Available and Data Request. These control signals greatly simplify the interface to a particular system. Internally, there are synchronous control signals of Block Available and Block Request. These signals interface to the particular algorithm unit permitting a full 64-bit block transfer in one clock cycle.

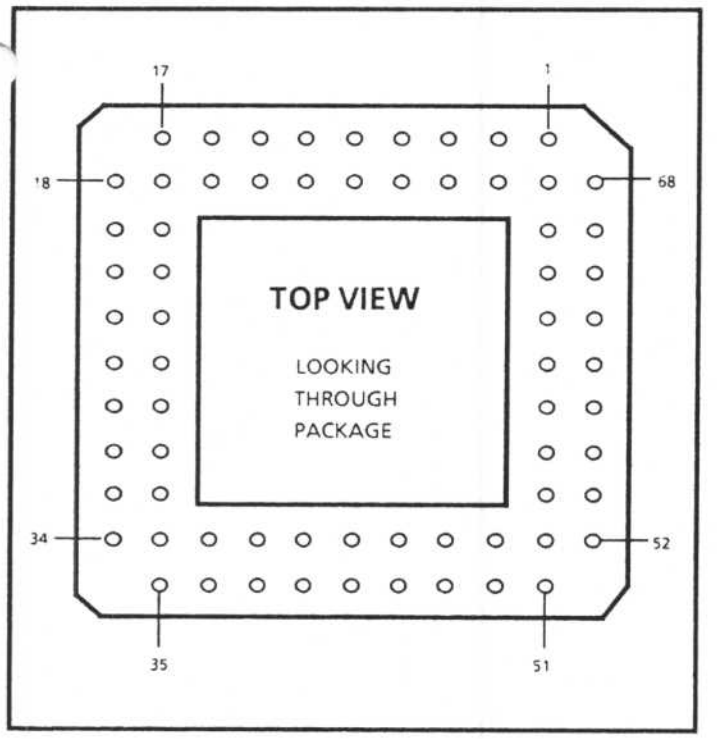


(U) The output FIFO has an 8-bit wide interface for transmitting data externally and a 64-bit wide interface to receive data from the particular algorithm unit. The output FIFO provides asynchronous control signals of Data Available and Data Request. These control signals greatly simplify the interface to a particular system. Internally, there are synchronous control signals Block Available and Block Request.

These signals interface to the check FIFO permitting a full 64-bit block transfer in one clock cycle.



(U) The Crypto-Engine™ is a 68-Pin Grid Array device. The data path channels are organized to permit good red-black *separation* and operate using asynchronous control signals. This permits the maximum performance interface to the Crypto-Engine™.



(U) The Crypto-Engine™ BUS pin naming convention conforms to the NSA standard where Pin 1 is the Most Significant Bit (MSB) and Pin 8 is the Least Significant Bit (LSB).

MCLK — The Master Clock input pin (44) provides a synchronizing clock for the Crypto-Engine™. This clock should be a clean, 50-50 duty cycle square wave.

RESET — The Reset input pin (61) is used to initialize the Crypto-Engine™ upon power up conditions.

VCC — Power pin (43) for +5 volts.

VSS — Ground pins (1, and 19).

TAMP1H — The Tamper 1 pin (8) is an active high input. A high signal on this pin causes the entire Crypto-Engine™ to be zeroized. In addition, the ALRM1H output signal will become active and remain so until the TAMP1H signal returns low and the Crypto-Engine™ has completed its *health* and *alarm* check procedure. (Note that the *red* key variable KK will have to be loaded and successfully checked before the ALRM1H pin is reset.)

TAMP2L — The Tamper 2 pin (7) is an active low input. When TAMP2L becomes active the ALRM2L output will also become active. Full zeroization will occur if either TAMP1H or TAMP2L pins become active. (Note that the *red* key variable KK will have to be loaded and successfully checked before the ALRM2L pin is reset.)

ALRM1H — The Alarm 1 pin (6) is an active high output. When active an alarm condition has occured in the operation of the Crypto-Engine™.

ALRM2L — The Alarm 2 pin (5) is an active low output. When active an alarm condition has occured in the operation of the Crypto-Engine™.

FILL2L — The Fill 2 pin (11) is an active low output. When active, the Crypto-Engine™ is requesting valid *red* key (KK) fill data.

FILL1H — The Fill 1 pin (53) is an active high output. When active, the Crypto-Engine™ is requesting valid *red* key (KK) fill data.

BYPASS — The Bypass pin (33) is an active low output. When active, it indicates that

the Crypto-Engine™ is in the *Bypass* mode.

CPCCLK — The Control Processor Chip (CPC) clock input pin (42) provices clock to internal high performance microcontroller. This clock should be a clean, 50-50 duty cycle square wave.

ZERO1L — The Zeroize pin (47) is an active low input. This pin is provided to allow the *host* to zeroize sensitive data stored inside the Crypto-Engine™.

CP1...8 — Command Port pins (14 thru 22) provide an 8-bit bidirectional data bus for communication with the Crypto-Engine™.

CRDL — The Command Read input pin (13) is an active low signal which controls the direction of the Command Port. When CRDL is low, the Command Port is confirgured as an output bus. When CRDL is high, the Command Port is an input bus.

CWRL — The Command Write pin (12) is an active low input. A strobe pulse on this pin will latch an input command on the Command Port bus into the Crypto-Engine™ if CRDL is high. If CRDL is low during the strobe, data is assumed to have been latched externally.

CDA — The Command Data Available output pin (10) provides an active high status signal which displays the state of the output register of the Command Port. When CDA is high there is data waiting to be read from the Command Port.

CDREQ — The Command Data Request output pin (9) provides an active high status signal which displays the state of the input register of the Command Port. When CDREQ is high the Command Port can accept data.

SERIN — The Serial Input pin (60) is used to load red key variables and control information into the Crypto-Engine™.

SEROUT — The Serial Output pin (59) is used to provide responses to requested red control functions.

BID1...8 — Black Input Data pins (64 thru 68 and 2 thru 4) provide a dedicated unidirectional 8-bit data bus for inputting ciphertext data.

BISTRBL — The Black Input Strobe pin (63) is an active low input pin which is used to latch the contents of the Black Input Data bus.

BIDREQ — The Black Input Data Request pin (62) is an active high output pin which is active when the Black Input port is ready to recieve data. This pin will remain active until eight bytes of data have been loaded.

BOD1...8 — The Black Output Data pins (34 thru 41) provide a dedicated unidirectional 8-bit data bus for outputting ciphertext data.

BOSTRBL — The Black Output Read pin (45) is an active low input pin which enables the Black Output Data pins to be driven.

BODA — The Black Output Data Available pin (46) is an active high output. When active, it indicates that data is available on the Black Output Port.

RID1...8 — Red Input Data pins (23 thru 30) provide a dedicated unidirectional 8-bit data bus for inputting plaintext data.

RISTRBL — Red Input Strobe pin (31) is an active low input pin which is used to latch the contents of the Red Input Data bus.

RIDREQ — Red Input Data Request pin (32) is an active high output pin which is active when the Red Input port is ready to recieve data. This pin will remain active until eight bytes of data have been loaded.

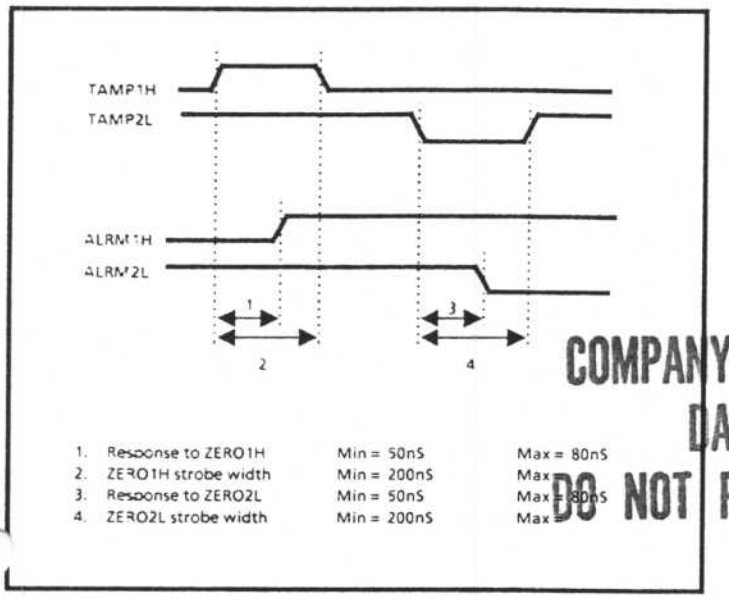
ROD1...8 — Red Output Data pins (50 thru 52 and 54 thru 58) provide a dedicated unidirectional 8-bit data bus for outputting plaintext data.

ROSTRBL — Red Output Read pin (48) is an active low input pin which enables the Red Output Data pins to be driven.

RODA — The Red Output Data Available pin (49) is an active high output. When active, it indicates that data is available on the Red Output Data Port.

(U) The security provided by the high-grade algorithm is augmented by the actual hardware of the chip. For

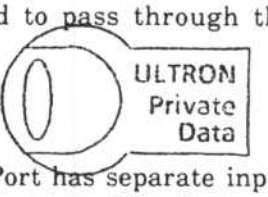
instance, there are two *tamper* input pins to guard against a failure of the tamper circuit (for complete zeroization) and two Alarm output pins to guard against a failure of the alarm circuit. The control signals for both tamper and alarm are complementary pairs. The *tamper* signals are inputs to the Crypto-Engine™ and will signal a hardware zeroization of all (including the Tamper Key) sensitive information. The *alarm* signals are outputs from the Crypto-Engine™ and indicate that an alarm condition (due to hardware zeroization or failure) has occurred. When the alarm signals are present, the Crypto-Engine™ data path channels are *inhibited.*



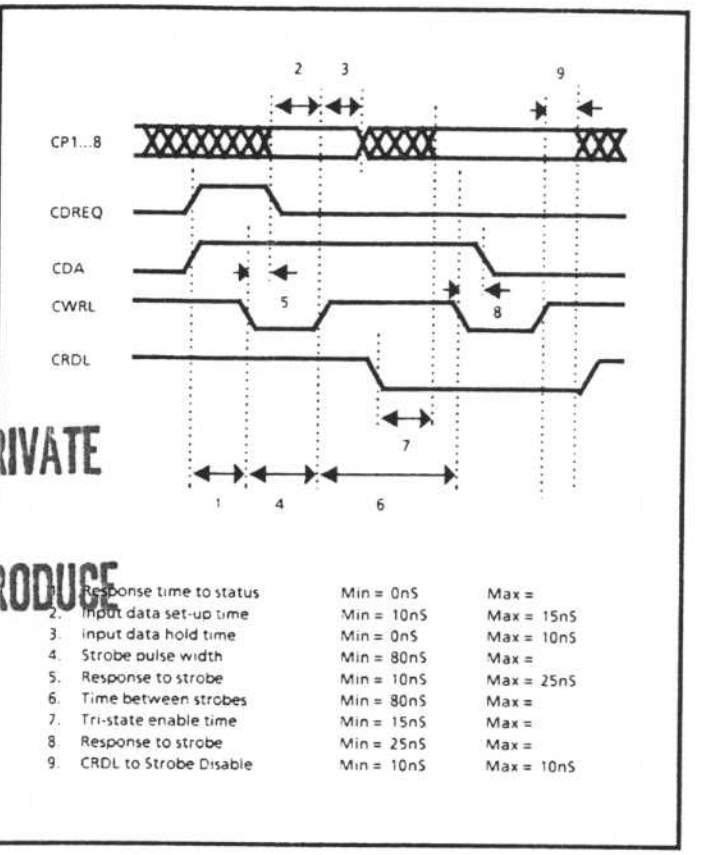
1.	Response to ZERO1H	Min = 50nS	Max = 80nS
2.	ZERO1H strobe width	Min = 200nS	Max =
3.	Response to ZERO2L	Min = 50nS	Max = 80nS
4.	ZERO2L strobe width	Min = 200nS	Max =

1.5 Using the Command Port

(U) The Command Port is an eight bit wide bidirectional port equipped with a special four-wire handshake to maximize versatility and speed. This port is used to bring commands, limited amounts of data directed here by *host*, black key variables, and control information (i.e., CCW and Checkwords) into the Crypto-Engine™. Conversely, it is also used to read responses, status and black key variables from the Crypto-Engine™. To maintain red-black separation no red key variables are allowed to pass through the Command Port.



(U) Internally, the Command Port has separate input and output registers. The direction of the bus and hence the access to these registers is controlled by the Command Read pin (CRDL). When the CRDL pin is low, the output drivers are turned on and the contents of the command output register are placed on the bus. When the CRDL pin is high, the output drivers are off and input data on the bus is fed to the input register. Once the direction of the bus is selected and the data has settled, a low-going pulse is placed on the command strobe pin (CWRL). The rising edge of this pulse latches the data into the input register or the external circuitry. The status of these two buffers is visible as the Command Data Available (CDA) pin and the Command Data Request (CDREQ) pin. The CDA pin is an active high output signal which signifies that the output buffer contains data to be read out. Similarly the CDREQ pin is an active high output signal which signifies that the input buffer is ready to receive data.



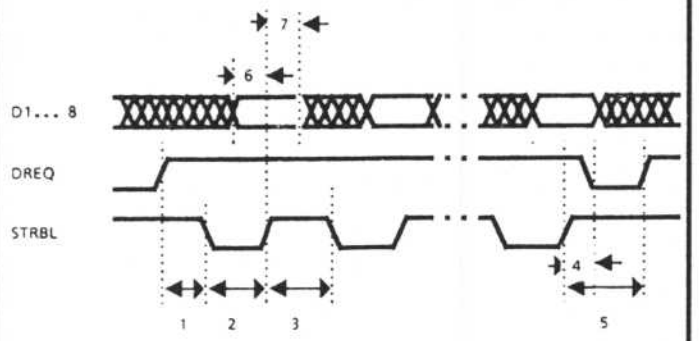
1.	Response time to status	Min = 0nS	Max =
2.	Input data set-up time	Min = 10nS	Max = 15nS
3.	Input data hold time	Min = 0nS	Max = 10nS
4.	Strobe pulse width	Min = 80nS	Max =
5.	Response to strobe	Min = 10nS	Max = 25nS
6.	Time between strobes	Min = 80nS	Max =
7.	Tri-state enable time	Min = 15nS	Max =
8.	Response to strobe	Min = 25nS	Max =
9.	CRDL to Strobe Disable	Min = 10nS	Max = 10nS

1.6 Using the Data Ports

(U) The chip is equipped with four high-speed data ports. Each port is eight bits wide and has its own two-wire handshake. In order to maintain red-black separation and to allow full duplex operation, the function of each port is dedicated to one of the following: red input data; red output data; black input data; and black output data. In addition, each port is designed to handle data asynchronously. The data transfer rate of the port is very fast because the handshake circuit is designed to take advantage of the asynchronous nature of the data and the fact that data arrives in 8 byte *bursts*. Bursting occurs because all data to be processed by the Crypto-Engine™ must appear in 64-bit blocks. When a port receives the first byte of data, the other seven are not far behind. The data port will actually stack the eight bytes together to form a block before sending it to the algorithm to be

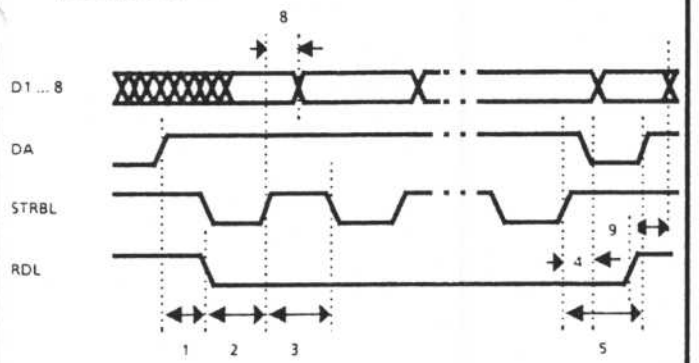
processed. This stacking occurs asynchronously to the Master Clock and as fast as the external circuit can send data.

INPUT PORTS
(RI and BI)



1.	Response time to DREQ or DA.	Min = 10nS	Max =
2.	Strobe width	Min = 80nS	Max =
3.	Time between strobes	Min = 80nS	Max =
4.	Status response to eighth strobe	Min = 10nS	Max = 25nS
5.	Delay between blocks (primary mode)	Min = 5.2uS	Max =
6.	Input data set-up time	Min = 15nS	Max =
7.	Input data hold time	Min = 10nS	Max =
8.	TriState Disable Time	Min = 0 nS	Max = 10nS
9.	Output bus tri-state response time	Min = 5 nS	Max = 15nS

OUTPUT PORTS
(RO and BO)



(U) The handshake scheme is *simple*. When an input port is ready to receive data, it places a high signal on its Data Request (DREQ) line for that input. When the external circuit senses this, it will place data on the bus and then place a low-going pulse on the Data Strobe (STRB). The rising edge of this pulse latches the data. The input data request line will remain high until eight such bytes have been latched and then it will drop low. Once the algorithm unit accepts the block, the Data Request returns high. The only limiting factors on the speed of the data transfer are the minimum pulse width of the low-going pulse and the minimum time between pulses.

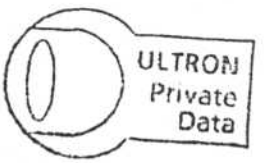
(U) A similar sequence of events occurs with the output port. When the port is ready to send data, the Data Available (DA) pin will go high and the first byte of data will be placed on the output bus. When the external circuit is ready to receive data it places a low-going pulse on the Data Strobe (STRB). The rising edge of this pulse signals that the data is latched in the external circuit. The rising edge will also cause the next byte to be placed on the output bus. When the last byte is read out, the Data Available pin drops low until the next block of data is ready to send.

1.7 Using the Serial Key Port

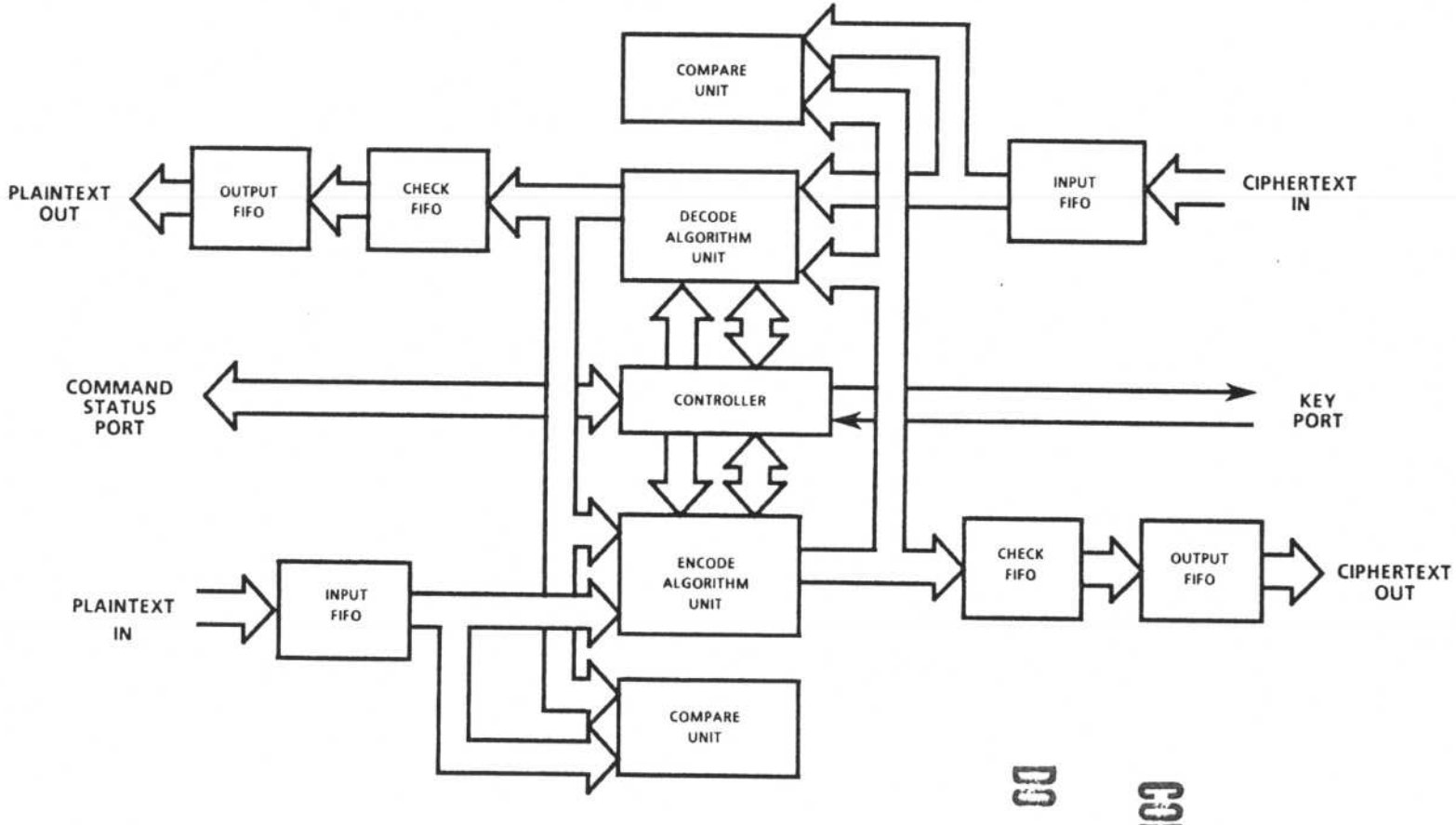
(U) The Serial Key Port is used to load red key variables and their associated control information (i.e., CCW and Checkword) into the Crypto-Engine™.

(U) The Serial Key Port is a 2-wire interface; one dedicated *transmit* line for protocol and acknowledgement information, one dedicated *receive* line for receipt of the *red* key variable itself. It operates at 1200 Baud and is receive-buffered, meaning it can commence reception of a second byte before a previously received byte has been read from the receive register. It is important to note that the *red* key variable is input only!

(U) Internally, the Serial Key Port has separate input (Receive) and output (Transmit) registers. Ten bits are transmitted or received: a start bit (0), 8 data bits (LSB first), and a stop bit (1).



CRYPTO-ENGINE™ FUNCTIONAL BLOCK DIAGRAM



1.8 State Diagram

(U) The Crypto-Engine™ can be in one of eight finite states (see attached diagram).

(U) The *Shutdown* State is entered whenever a *reset* is signalled, either with the hardware pin or the software command. This State implies that the Crypto-Engine™ is completely *sanitized* (i.e., zeroized).

(U) The *Alarm* State is entered whenever there is a Crypto-Engine™ failure (including cryptographic failures) or the *tamper* pins (TAMP1H and TAMP2L) are signalled. A complete alarm check sequence must be executed before processing can continue. This sequence is initiated by the *reset* (either hardware pin signalled or the Reset command).

(U) The *NonInitialized* State is entered by one way. That is, while in a *Shutdown* State, by the successful execution of the LoadKeyEncryptingKey command. When in the *NonInitialized* State, the KS, KD's and their associated Checkwords, and MIs are sanitized. While in this state, the Crypto-Engine™ will generate a new KT and await the LoadZeroizeKey command.

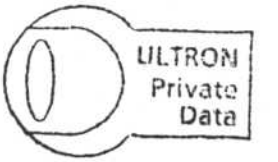
(U) The *Locked* State is entered whenever the CIK is removed from the *host* device. This can occur from either the *Stopped*, *Running*, or *Bypass* States by issuing the Lock command. When in the *Locked* State, the KS, KD's, and and their associated Checkwords, and MIs are sanitized. Once locked, processing can continue by *unlocking* the Crypto-Engine™. This is done with the Unlock command.

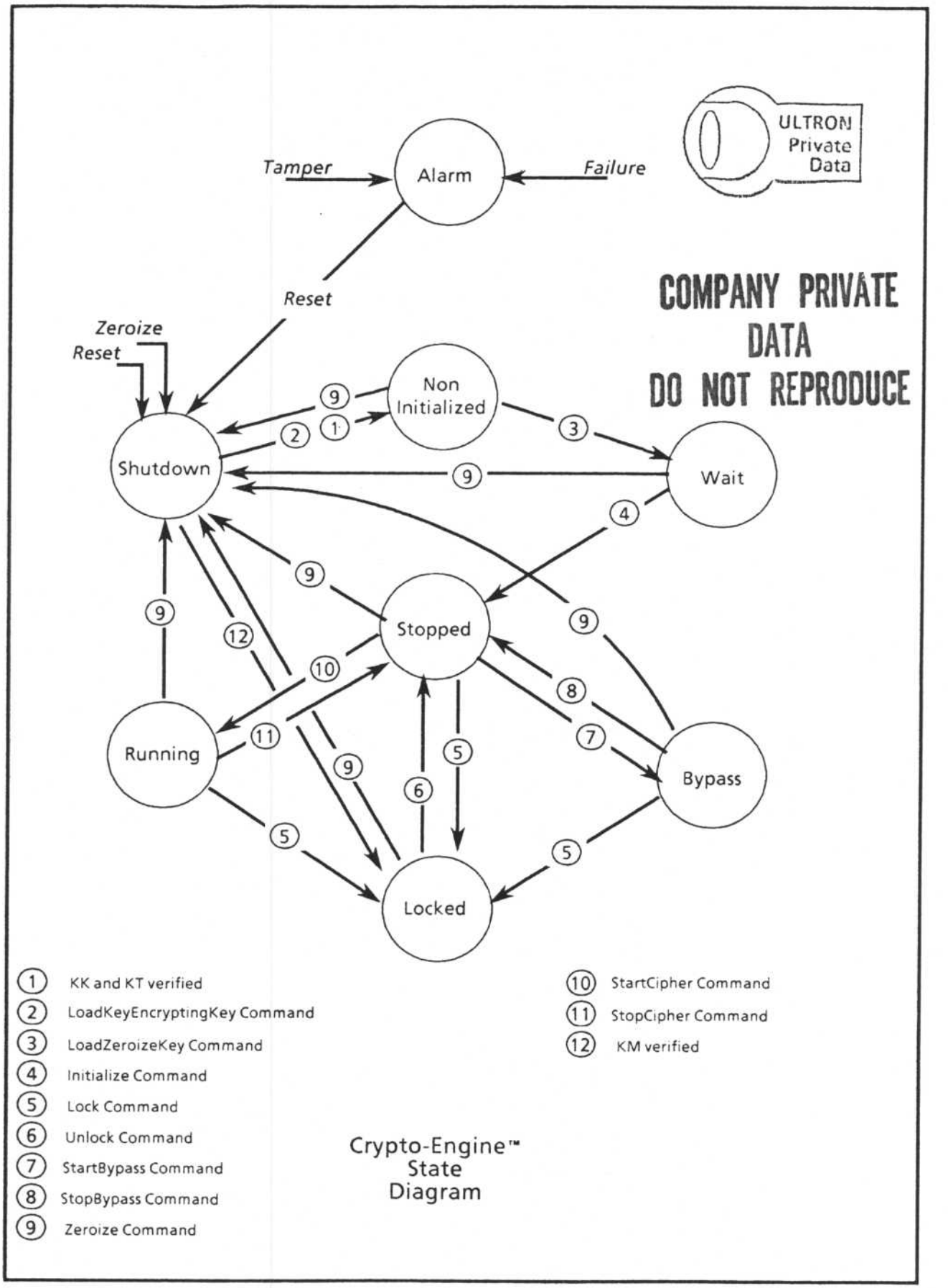
(U) The *Running* State is entered by issuing the StartCipher command. This state implies that the data path channels are enabled and the Crypto-Engine™ is ready to cipher or decipher data. While in this state, the Crypto-Engine™ can be stopped using the StopCipher command, zeroized using the Zeroize command, or locked using the Lock command.

(U) The *Bypass* State is entered by issuing the StartBypass command. This state implies that the data path channels are in a clear (plaintext) mode. While in this state, the Crypot-Engine™ can be stopped using the StopBypass command, zeroized using the Zeroize command, or locked using the Lock command.

(U) The *Stopped* State is entered whenever the Crypto-Engine™ is *initialized* (using the Initialize command) from the Wait state or upon receiving a StopCipher command from the Running state, StopBypass command from the Bypass state, or Unlock command from the Locked state. The data path channels are inhibited while in this state. Basically when in this state, the Crypto-Engine™ can process commands but no traffic. Therefore, most commands are valid.

(U) The *Wait* State is entered by issuing the LoadZeroizeKey command. This state implies that the Random Number Generator is *free-running* and no traffic can be processed. The Crypto-Engine™ must be initialized (via the Initialize command) before processing can continue.





Crypto-Engine™
State
Diagram

1.10 Crypto-Engine Commands

(U) Commands and parameter data, and command responses, are issued to the Crypto-Engine™ Command Port. Commands and parameter data are optionally issued through the Key Port on selected commands for added flexibility/security. This feature allows the separation of secure (*red*) and non-secure (*black*) operations. Data to be authenticated or encrypted are entered through the *red* Data Input Port and removed from the *black* Data Output Port. Data to be decrypted are entered through the *black* Data Input Port and removed from the *red* Data Output Port.

(U) Parameters are defined in terms of bits or bit fields. They are first broken up into bytes (P1,P2,P3,...,Pn); each byte is then displayed graphically with the rightmost bit being the most significant (7..0). Explanations of each bit or bit field appear below the byte.

1.10.1 Command Format

<Command><Response>[<DataIn>]
<Terminator>[<DataOut>]

<Command>
One of the commands described in this document. Commands enter the Crypto-Engine™ through either the Command Port or the Serial Key Port.

<Response>
Either <ACK>, meaning the command was accepted, or one of the error codes, meaning that the command was not accepted. A Response is read from the same port by which the command is entered.

<DataIn>
Any parameters required by a command. These data enter the Crypto-Engine™ through the same Port as the <Command>.

<Terminator>
Either <Done>, meaning the command was executed correctly, or one of the error codes, meaning that the command failed. A Terminator is read from the same port by which the <Command> is entered.

<DataOut>
Any data generated or returned by a command. These data are read from the same Port as the <Command>.

1.10.2 Command Protocol

(U) When the Crypto-Engine™ receives a command, it must perform various checks prior to executing the request. The command itself is checked for validity. If not valid, the error code <INVCMD> is returned. If valid, the command is checked for operational validity. This means the Crypto-Engine™ is in a *state* which permits the execution of the request. If not, the error code <INVST> is returned.

(U) If the checks are successful, the Crypto-Engine™ will accept <DataIn>, if applicable; perform the requested function; signal termination via <Terminator>; produce <DataOut>, if applicable. The exception to this protocol is the EncryptData and DecryptData commands.

1.10.3 Response Codes

The valid *hexadecimal* response codes are:

<ACK>	=	06
<INVCMD>	=	43
<INVST>	=	45

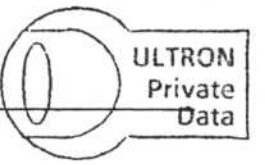
1.10.4 Terminator Codes

The valid *hexadecimal* termination codes are:

<DONE>	=	22
<ALARM>	=	55
<MINL>	=	49
<KNL>	=	48
<KUNL>	=	45
<INVCKWD>	=	51
<INVMAC>	=	77
<INVCCW>	=	53
<INVCOUNT>	=	69

1.10.5 GenMAC

(U) This command is used to *authenticate* data which is presented through the Command Port of the Crypto-Engine™. Data to be authenticated is presented as 64-bit blocks.



States

Valid State[s]: *Stopped*
Result: *Stopped*

Syntax

<Cmd> = <EA>
<Resp> = <ACK>|<INVCMD>|
<INVST>|<KNL>|
<MINL>
<DataIn> = <Count><Block[s]>
<Term> = <DONE>|<ALARM>
<DataOut> = <AuthCode>

Parameters

<Count> = 8-bit block count indicating the number of blocks to be authenticated.
<Block(s)> = One or more 64-bit plaintext blocks.
<AuthCode> = A 32-bit Authentication Code.

1.10.6 DecodeKeyPhrase

(U) This command processes a *rekey* phrase in order to *assist* the host device in implementing remote rekeying. This is accomplished by using the cryptographic algorithm and a *unique* key variable.

States

Valid State(s): *Stopped*
Result: *Stopped*

Syntax

<Cmd> = <D8>
<Resp> = <ACK>|<INVCMD>|
<INVST>|<KUNL>
<DataIn> = $<MI_{RKP}>$
$<E_{KU}[KD,CCW_{KD},Ckwd_{KD}]>$
<Term> = <DONE>|<ALARM>|
<INVCKWD>
<DataOut> = $<MI_{KD}>$
$<E_{KS}[KD,CCW_{KD},Ckwd_{KD}]>$

Parameters

$<E_{KS}[KD]>$ = *Remote* key variable encrypted with the *memory* key KS.
$<E_{KS}[CCW_{KD}]>$ = Crypto-Control Word (CCW) value for the *remote* key KD'.
$<E_{KS}[Ckwd_{KD}]>$ = Checkword value for the *remote* key KD'.
$<E_{KU}[KD]>$ = *Remote* key variable encrypted with the *unique* key KU.
$<E_{KU}[CCW_{KD}]>$ = Crypto-Control Word (CCW) value for the *remote* key KD'.
$<E_{KU}[Ckwd_{KD}]>$ = Checkword value for the *remote* key KD'.
$<MI_{RKP}>$ = Message Indicator used to process the *rekey* phrase.
$<MI_{KD}>$ = Message Indicator used to encrypt the *remote* key KD'.

1.10.7 DecryptData

(U) This command is used to *decrypt* data through the Command Port of the Crypto-Engine™. Data to be decrypted is presented as 64-bit blocks. Note that the incoming block is decrypted and returned before the next block can be presented. This continues until all blocks (specified by <Count>) is processed.

States

Valid State[s]: *Stopped*
Result: *Stopped*

Syntax

<Cmd> = <D9>
<Resp> = <ACK>|<INVCMD>|
<INVST>|<KNL>|
<MINL>
<DataIn> = <Count>
<Term> = <DONE>|<INVCOUNT>
<DataIn> = $<E_{KD}[Block(s)]>$
<DataOut> = <Block(s)>

Parameters

<Count> = 8-bit block count (greater than zero) indicating the number of blocks to be decrypted.

<E_{KD}[Block(s)]> = One or more 64-bit blocks of data encrypted with the *traffic* variable KD.

<Block(s)> = One or more 64-bit plaintext blocks of data.

1.10.8 EncodeKeyPhrase

(U) This command generates a *rekey* phrase in order to *assist* the host device in implementing remote *rekeying*. This is accomplished by using the cryptographic algorithm and a *unique* key variable.

States

Valid State(s): *Stopped*
Result: *Stopped*

Syntax

<Cmd> = <B8>
<Resp> = <ACK>|<INVCMD>|<INVST>|<KUNL>
<DataIn> = <MI_{KD}>
<E_{KS}[KD,CCW_{KD},$Ckwd_{KD}$]>
<Term> = <DONE>|<ALARM>|<INVCKWD>
<DataOut> = <MI_{RKP}>
<E_{KU}[KD,CCW_{KD},$Ckwd_{KD}$]>

Parameters

<E_{KS}[KD]> = Remote key KD' encrypted with the memory key KS.

<E_{KS}[CCW_{KD}]> = Crypto-Control Word (CCW) value for the *remote* key KD'.

<E_{KS}[$Ckwd_{KD}$]> = Checkword value for the *remote* key KD'.

<E_{KU}[KD]> = *Remote* key variable encrypted with the *unique* key KU.

<E_{KU}[CCW_{KD}]> = Crypto-Control Word (CCW) value for the *remote* key KD.

<E_{KU}[$Ckwd_{KD}$]> = Checkword value for the *remote* key KD.

<MI_{RKP}> = Message Indicator used to process the *rekey* phrase.

<MI_{KD}> = Message Indicator used to encrypt the *remote* key KD'.

1.10.9 EncryptData

(U) This command is used to *encrypt* data through the Command Port of the Crypto-Engine™. Data to be encrypted is presented as 64-bit blocks.Note that the incoming block is encrypted and returned before the next block can be presented. This continues until all blocks (specified by <Count>) is processed.

States

Valid State[s]: *Stopped*
Result: *Stopped*

Syntax

<Cmd> = <C8>
<Resp> = <ACK>|<INVCMD>|<INVST>|<KNL>|<MINL>
<DataIn> = <Count>
<Term> = <DONE>|<INVCOUNT>
<DataIn> = <Block(s)>
<DataOut> = <E_{KD}[Block(s)]>

Parameters

<Count> = 8-bit block count indicating the number of blocks to be encrypted.

<E_{KD}[Block(s)]> = One or more 64-bit blocks of data encrypted with the *traffic* variable KD.

<Block(s)> = One or more 64-bit plaintext blocks of data.

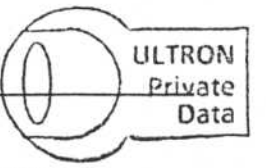
1.10.10 GenMessageIndicator

(U) This command generates a Message Indicator (MI) of 64 bits in length. Once generated, the Crypto-Engine™ automatically loads (Encryption only) the MI and also returns its value.

States

Valid State(s): *Stopped*
Result: *Stopped*

Syntax



<Cmd> = <F1>
<Resp> = <ACK>|<INVCMD>|
<INVST>
<Term> = <DONE>|<ALARM>
<DataOut> = <MI>

Parameters

<MI> = A 64-bit pseudo-random *generated* Message Indicator.

1.10.11 GenRandomNumber

(U) This command generates a pseudo-random number (RN) of 64 bits in length. This is accomplished by using the cryptographic algorithm and key variable variant.

(U) The procedure is such that it is not computationally feasible to deduce the variant used to encrypt the *seed* value or subsequent unknown values of RN using knowledge of one or more prior values of RN. Furthermore, this process is completely isolated from other cryptographic functions because the variant used is only for generating pseudo-random numbers.

(U) The RN is *generated* within the Crypto-Engine™ as a result of the dynamically changing and unpredictable nature of the resource demands placed upon the system by its users.

States

Valid State(s): *Stopped*
Result: *Stopped*

Syntax

<Cmd> = <E1>
<Resp> = <ACK>|<INVCMD>|
<INVST>
<Term> = <DONE>|<ALARM>
<DataOut> = <RN>

Parameters

<RN> = A 64-bit pseudo-random *generated* Random Number.

1.10.12 GetStatus

(U) This command will return the current *status* of the Crypto-Engine™.

States

Valid State(s): *All*
Result: *Same*

Syntax

<Cmd> = <D1>
<Resp> = <ACK>|<INVCMD>
<Term> = <DONE>
<DataOut> = <V0><V1><V2><V3>
<Version><Revision>

Parameters

<V0> = [7.6.5.4.3.2.1.0]
0..7: Key Variable Update Counter value

<V1> = [7.6.5.4.3.2.1.0]
0: Encrypt Key Loaded
1: Decrypt Key Loaded
2: Encrypt MI Loaded
3: Decrypt MI Loaded
4: KU Loaded
5..7: Reserved

<V2> = [7.6.5.4.3.2.1.0]
0: Shutdown State
1: Zeroized State
2: Locked State
3: Stopped State
4: Running State
5: Bypass State
6: Wait State
7: Alarm State

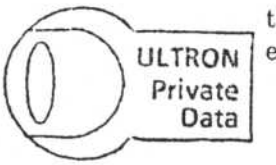
<V3> = [7.6.5.4.3.2.1.0]
0..3: Cryptographic Mode
0000B: Mode 0
0001B: Mode 1
0010B: Mode 2
4: Direction
0: Encrypt
1: Decrypt
5..7: Reserved

1.10.13 Initialize

(U) This command will initialize the Crypto-Engine™, establishing the following:

- Tamper Key (KT)
- Checkword for KT

- Zeroize Variable (ZV)
- Master Key (KM)
- Checkword for KM
- SubMaster Key (KS)
- Checkword for KS
- Random Number (RN)
- *Split* Key Component (KC_1)



(U) The initialization of the Crypto-Engine™ is a multi-phase process. During the first phase, various *health* checks are performed to ensure the *integrity* of the Crypto-Engine™. The next phase involves the generation of the Tamper Key (KT) and associated Checkword, Master Key (KM), Checkword for KM, and the SubMaster Key (KS) and its associated Checkword. The last phase generates two split components (KC_1 and KC_2) from the SubMaster Key (KS). Once the last phase is completed, the Crypto-Engine™ outputs the split component KC_2 and enters the *Stopped* State.

States

Valid State(s): *Wait*
Result: *Stopped*

Syntax

<Cmd> = <A1>
<Resp> = <ACK>|<INVCMD>|<INVST>
<Term> = <DONE>|<ALARM>
<DataOut> = $<MI_{KC_2}>$ $<E_{KM_2}[KC_2], CCW_{KC_2}, Ckwd_{KC_2}>$

Parameters

$<MI_{KC_2}>$ = Message Indicator used to encrypt the *split* components of the *memory* variable KS.

$<E_{KM_2}[KC_2]>$ = *Split* Key Component of the *memory* variable KS encrypted with the Master Key *variant* KM_2.

1.10.14 LoadEncryptDataKey

(U) This command loads the encrypting *traffic* Data Key (KD) and its associated Crypto-Control Word (CCW), and Checkword. The KD is used to *authenticate* or *cipher* data passing through the Data Port(s) or the Command Port. Before this command can be executed, the Crypto-Engine™ must have been *unlocked*, which establishes the KS needed to decrypt the incoming KD.

States

Valid State(s): *Stopped*
Result: *Stopped*

Syntax

<Cmd> = <D2>
<Resp> = <ACK>|<INVCMD>|<INVST>
<DataIn> = $<MI_{KD}>$ $<E_{KS}[KD,CCW_{KD},Ckwd_{KD}]>$
<Term> = <DONE>|<ALARM>|<INVCKWD>

Parameters

$<MI_{KD}>$ = Message Indicator used to decrypt the key variable KD.

$<E_{KS}[KD]>$ = *Decrypting* key variable KD, encrypted with the *memory* variable KS.

$<E_{KS}[CCW_{KD}]>$ = Crypto-Control Word (CCW) for the key variable KD encrypted with the memory variable KS.

$<E_{KS}[Ckwd_{KD}]>$ = Checkword for the key variable KD encrypted with the memory variable KS.

1.10.15 LoadDecryptDataKey

(U) This command loads the decrypting *traffic* Data Key (KD) and its associated Crypto-Control Word (CCW), and Checkword. The KD is used to *authenticate* or *decipher* data passing through the Data Port(s) or the Command Port. Before this command can be executed, the Crypto-Engine™ must have been *unlocked*, which establishes the KS needed to decrypt the incoming KD.

States

Valid State(s): *Stopped*
Result: *Stopped*

Syntax

<Cmd> = <DA>
<Resp> = <ACK>|<INVCMD>|<INVST>
<DataIn> = <MI_{KD}>
<E_{KS}[KD,CCW_{KD},$Ckwd_{KD}$]>
<Term> = <DONE>|<ALARM>|<INVCKWD>

Parameters

<MI_{KD}> = Message Indicator used to decrypt the key variable KD.
<E_{KS}[KD]> = *Encrypting* key variable KD, encrypted with the *memory* variable KS.
<E_{KS}[CCW_{KD}]> = Crypto-Control Word (CCW) for the key variable KD encrypted with the memory variable KS.
<E_{KS}[$Ckwd_{KD}$]> = Checkword for the key variable KD encrypted with the memory variable KS.

1.10.16 LoadKeyEncryptingKey

(U) This command loads a *plaintext* Key Variable (KK) and its associated Crypto-Control Word (CCW), and Checkword.

(U) Before this command can be executed, the Crypto-Engine™ must be in the *Shutdown* State. Upon loading the Key Variable, a Checkword test is performed.

(U) This command must be entered through the **Serial Key Port** of the Crypto-Engine™ and not through the Command Port.

States

Valid State(s): *Shutdown*
Result: *Zeroized*

Syntax

<Cmd> = <D2>
<Resp> = <ACK>|<INVCMD>|<INVST>
<DataIn> = <[KK,CCW_{KK},$Ckwd_{KK}$]>
<Term> = <DONE>|<ALARM>|<INVCKWD>

Parameters

<KK> = *Plaintext* variable.
<CCW_{KK}> = Crypto-Control Word (CCW) for the key variable KK.
<$Ckwd_{KK}$> = Checkword for the key variable KK.

1.10.17 LoadMessageIndicator

(U) This command allows for the direct loading of a Message Indicator (MI) for *decryption* purposes only. The MI will be placed into the DMI Register of the Crypto-Engine™.

States

Valid State(s): *Stopped*
Result: *Stopped*

Syntax

<Cmd> = <D3>
<Resp> = <ACK>|<INVCMD>|<INVST>
<DataIn> = <MI>
<Term> = <DONE>

Parameters

<MI> = A 64-bit pseudo-random value to be used for decryption.

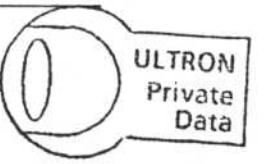
1.10.18 LoadUniqueKey

(U) This command loads the Unique Key (KU) and its associated Crypto-Control Word (CCW), and Checkword.

(U) Once the KU is loaded, a Checkword test is performed. If not successful, a non-fatal error condition will occur.

States

Valid State(s): *Stopped*
Result: *Stopped*



Syntax

<Cmd> = <C7>
<Resp> = <ACK>|<INVCMD>|<INVST>
<DataIn> = <MI_{KU}><E_{KS}[KU,CCW_{KU},$Ckwd_{KU}$]>
<Term> = <DONE>|<ALARM>|<INVCKWD>

Parameters

<MI_{KU}> = Message Indicator used to decrypt the *unique* variable KU.

<E_{KS}[KU]> = *Unique* variable KU, encrypted with the *memory* variable KS.

<E_{KS}[CCW_{KU}]> = Crypto-Control Word (CCW) for the unique variable KU encrypted with the *memory* variable KS.

<E_{KS}[$Ckwd_{KU}$]> = Checkword for the *unique* variable KU encrypted with the *memory* variable KS.

1.10.19 LoadZeroizeKey

(U) This command loads the Zeroize Key (KZ) and its associated Crypto-Control Word (CCW), and Checkword. The KZ is used as part of a *warm-start* procedure.

(U) Once the KZ is loaded, a Checkword test is performed. If successful, the Crypto-Engine™ enters the *Wait* State.

States

Valid State(s): *NonInitialized*
Result: *Wait*

Syntax

<Cmd> = <A8>
<Resp> = <ACK>|<INVCMD>|<INVST>
<DataIn> = <MI_{KZ}><E_{KK}[KZ,CCW_{KZ},$Ckwd_{KZ}$]>
<Term> = <DONE>|<ALARM>|<INVCKWD>

Parameters

<MI_{KZ}> = Message Indicator used to decrypt the *zeroization* variable KZ.

<E_{KK}[KZ]> = *Zeroization* variable KZ, encrypted with the *distribution* variable KK.

<E_{KK}[CCW_{KZ}]> = Crypto-Control Word (CCW) for the zeroization variable KZ encrypted with the *distribution* variable KK.

<E_{KK}[$Ckwd_{KZ}$]> = Checkword for the *zeroization* variable KZ encrypted with the *distribution* variable KK.

1.10.20 Lock

(U) This command is used to place the Crypto-Engine™ in a *Locked* State. In order to do this, the Crypto-Engine™ is *sanitized* by zeroizing the SubMaster Key (KS) and associated KDs. After a *lock* operation has been performed, the Unlock command is used to place the Crypto-Engine™ in the *Stopped* state again.

States

Valid State(s): *Stopped*, Running, or *Bypass*
Result: *Locked*

Syntax

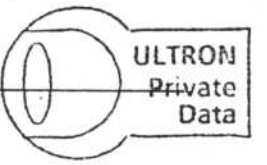
<Cmd> = <B2>
<Resp> = <ACK>|<INVCMD>|<INVST>
<Term> = <DONE>

Parameters

None.

1.10.21 ReadMessageIndicator

(U) This command will return the 64-bit Message Indicator (MI) from the Crypto-Engine™.



States

Valid State(s): *Stopped*
Result: *Stopped*

Syntax

<Cmd> = <F3>
<Resp> = <ACK>|<INVCMD>|<INVST>
<Term> = <DONE>
<DataOut> = <MI>

Parameters

<MI> = A 64-bit Message Indicator value.

1.10.22 Reset

(U) This command forces the Crypto-Engine™ back to its *uninitialized* configuration, flushes all I/O buffers, and places the Crypto-Engine™ in the *NonInitialized* State.

States

Valid State(s): *All*
Result: *Zeroized*

Syntax

<Cmd> = <B7>
<Resp> = <ACK>|<INVCMD>|<INVST>
<Term> = <DONE>

Parameters

None.

1.10.23 SetCryptoMode

(U) This command is used to select the cryptographic mode of the Crypto-Engine™ for *ciphering*, *authenticating*, or *binding* data.

(U) When authenticating or binding, data is entered into the Crypto-Engine™, but ciphertext is not returned. Instead, the data is discarded since the ciphertext is not needed when authenticating data. Upon completion of the authentication process, the Authentication Code (AuthCode) is available from the Crypto-Engine™.

States

Valid State(s): *Stopped*
Result: *Stopped*

Syntax

<Cmd> = <B3>
<Resp> = <ACK>|<INVCMD>|<INVST>
<DataIn> = <Mode><Direction>
<Term> = <DONE>

Parameters

<Mode> = bits 0, 1, 2, and 3 which is a value for the cryptographic mode:
0: = Mode 0
1: = Mode 1
2: = Mode 2
3: = Reserved

<Direction> = bit 4
0: = Encrypt
1: = Decrypt

1.10.24 StartBypass

(U) This command is used to select the cryptographic bypass mode of the Crypto-Engine™ for processing plaintext data.

States

Valid State(s): *Stopped*
Result: *Bypass*

Syntax

<Cmd> = <C2>
<Resp> = <ACK>|<INVCMD>|
<INVST>
<Term> = <DONE>

Parameters

None.

1.10.25 StartCipher

(U) This command enables the data path channels of the Crypto-Engine™. (This is the inverse of the Stop command which inhibits the data path channels.) The execution of this command will place the Crypto-Engine™ in the *Running* state.

States

Valid State(s): *Stopped*
Result: *Running* or *Bypass*

Syntax

<Cmd> = <B1>
<Resp> = <ACK>|<INVCMD>|
<INVST>
<Term> = <DONE>|<KNL>|
<MINL>

Parameters

None

1.10.26 StopBypass

(U) This command is used to deselect the cryptographic bypass mode of the Crypto-Engine™.

States

Valid State(s): *Bypass*
Result: *Stopped*

Syntax

<Cmd> = <F9>
<Resp> = <ACK>|<INVCMD>|
<INVST>
<Term> = <DONE>

Parameters

None.

1.10.27 StopCipher

(U) This command *inhibits* the data path channels of the Crypto-Engine™. (This is the inverse of the Start command which enables the data path channels.) The execution of this command will place the Crypto-Engine™ in the *Stopped* State.

States

Valid State(s): *Running* or *Bypass*
Result: *Stopped*

Syntax

<Cmd> = <C1>
<Resp> = <ACK>|<INVCMD>|
<INVST>
<Term> = <DONE>

Parameters

None

1.10.28 TranslateKV

(U) This command performs a translation of a cryptogram (KV) protected by a Key-Encryption-Key (KK) to a cryptogram (KV) protected by the Sub-Master Key (KS) within the Crypto-Engine™. A checkword test is performed on the KV being translated as part of the execution of this command. The KK must have been previously loaded.

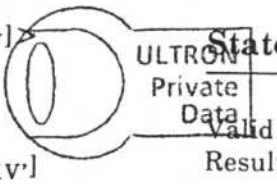
States

Valid State(s): *Stopped*
Result: *Stopped*

Syntax

<Cmd> = <B4>

<Resp> = <ACK>|<INVCMD>|
<INVST>
<DataIn> = <MI_{KK}>
<E_{KK}[KV,CCW_{KV},$Ckwd_{KV}$]>
<Term> = <DONE>|<ALARM>|
<KNL>|<INVCKWD>
<DataOut> = <MI_{KS}>
<E_{KS}[KV',$CCW_{KV'}$,$Ckwd_{KV'}$]
>



Parameters

<MI_{KK}> = Message Indicator supplied to decrypt (the incoming) KD or generated to encrypt (the outgoing) KD.

<E_{KK}[KV]> = Key variable encrypted with the *distribution* variable KK.

<E_{KK}[CCW_{KV}]> = Crypto-Control Word (CCW) value for KV encrypted with the *distribution* variable KK.

<E_{KK}[$Ckwd_{KV}$]> = Checkword value for KV encrypted with the *distribution* variable KK.

<MI_{KS}> = Message Indicator supplied to encrypt (the outgoing) KV.

<E_{KS}[KV]> = Key variable encrypted with the *memory* variable KS.

<E_{KS}[CCW_{KV}]> = Crypto-Control Word (CCW) value for KV encrypted with the *memory* variable KS.

<E_{KS}[$Ckwd_{KV}$]> = Checkword value for KV encrypted with the *memory* variable KS.

1.10.29 Unlock

(U) This command *unlocks* the Crypto-Engine™. This is accomplished by the computation of a SubMaster Key (KS) which provides protection to the Data Keys (KDs). The Unlock command performs the inverse operation of *locking* the Crypto-Engine™.

(U) The fundamentals of this command include the concept of a *split* key comprised of two key components (KC_1 and KC_2). The Crypto-Engine™ decrypts and *combines* (transforms) the KCs into the KS. Then the Crypto-Engine™ checks the KS using a Checkword test. If successful, the Crypto-Engine™ generates and returns a new KC_2 which can be used for the next *lock-unlock* cycle.

States

Valid State(s): *Locked*
Result: *Stopped*

Syntax

<Cmd> = <AB>
<Resp> = <ACK>|<INVCMD>|
<INVST>
<DataIn> = <MI_{KC}>
<E_{KM_2}[KC_2, CCW_{KC_2},
$Ckwd_{KC_2}$]>
<Term> = <DONE>|<ALARM>|
<INVCKWD>
<DataOut> = <$MI_{KC'}$>
<E_{KM_2}[KC_2', CCW_{KC_2},
$Ckwd_{KC_2}$]>

Parameters

<MI_{KC}> = Message Indicator supplied to decrypt (incoming) *split* components.

<E_{KM_2}[KC_2]> = *Split* key component of the *memory* variable KS encrypted with the master key *variant* KM_2.

<$MI_{KC'}$> = Message Indicator generated to decrypt (incoming) encrypt (the outgoing) *split* components.

<E_{KM_2}[KC_2']> = Newly generated *split* key component of the *memory* variable KS encrypted with the master key *variant* KM_2.

<E_{Km_2}[CCW_{KC_2}]> = Crypto-Control Word (CCW) value for the *split* key component KC_2 encrypted with the *variant* master variable KM_2.

<E_{KM_2}[$Ckwd_{KC_2}$]> = Checkword value for the *split* key component KC_2 encrypted with the *variant* master variable KM_2.

1.10.30 UpdateKey

(U) This command is used to generate new key variables from old ones in a deterministic and irreversible way when the user desires to protect back traffic. Note that, if any UPDATED key variable (KV) is compromised, future traffic is compromised (since UPDATING is deterministic), but back traffic is not (since the process is irreversible). The number of updates allowable is determined as a function of the cryptologic and application. Assurances are made by the Crypto-Engine™ that this number is not exceeded.

States

Valid State(s): *Stopped*
Result: *Stopped*

Syntax

<Cmd> = <E3>
<Resp> = <ACK>|<INVCMD>|
<INVST>|
<NOTAUTH>
<DataIn> = <MI_{KV}>
<E_{KS}[KV,CCW_{KV},$Ckwd_{KV}$]>
<Term> = <DONE>|<ALARM>|
<INVCKWD>
<DataOut> = <$MI_{KV'}$>
<E_{KS}[KV',$CCW_{KV'}$,$Ckwd_{KV'}$]
>

Parameters

<$MI_{KV'}$> = Message Indicator supplied to decrypt (incoming) KV or generated to encrypt (outgoing) KV.

<E_{KS}[KV']> = Key variable KV encrypted with the *memory* variable KS.

<E_{KS}[$CCW_{KV'}$]> = Crypto-Control Word (CCW) value for the key variable KV encrypted with the *memory* variable KS.

<E_{KS}[$Ckwd_{KV'}$]> = Checkword value for the key variable KV encrypted with the *memory* variable KS.

1.10.31 VerifyMAC

(U) This command allows the computing and checking of the Message Authentication Code (MAC) within the Crypto-Engine™. This is accomplished by performing a bit-for-bit compare of the incoming supplied MAC and the internally computed MAC (based on the incoming data) within the Crypto-Engine™. If the comparison test fails, a non-fatal error condition will occur and an error response will be returned to the *host*..

States

Valid State(s): *Stopped*
Result: *Stopped*

Syntax

<Cmd> = <A6>
<Resp> = <ACK>|<INVCMD>|
<INVST>|<KNL>|
<MINL>
<DataIn> = <Count><Block[s]>
<MAC>
<Term> = <DONE>|<INVMAC>

Parameters

<Count> = 8-bit block count indicating the number of blocks to be authenticated.

<Block(s)> = One or more 64-bit plaintext blocks.

<MAC> = A 32-bit Authentication Code.

Parameters

None.

1.10.32 SelfTest

(U) This command allows the *host* to request a *health* check sequence to be executed. This test is the same as the *warm-start* health test. See Section 13.0 entitled Health and Alarm Checks in this Theory of Compliance for further details. It should be noted that if this *health* test fails, a fatal error condition will occur which will place the Crypto-Engine™ in an *Alarm* State.

States

Valid State(s): *Stopped*
Result: *Stopped or Alarm*

Syntax

<Cmd> = <A2>
<Resp> = <ACK>|<INVCMD>|<INVST>
<Term> = <DONE>|<ALARM>

Parameters

None.

1.10.33 VerifyTamperKey

(U) This command allows the checking of the Tamper Key (KT) within the Crypto-Engine™. This is accomplished by performing a Checkword test on the KT within the Crypto-Engine™. If the Checkword test fails, an alarm condition will occur.

States

Valid State(s): *Stopped*
Result: *Alarm*

Syntax

<CMD> = <A3>
<Resp> = <ACK>|<INVCMD>|<INVST>
<Term> = <DONE>

Parameters

None.

1.10.33 Zeroize

(U) This command will *sanitize* the Crypto-Engine™ except for the KeyEncryptingKey and associated Checkword, Tamper Key (KT) and associated Checkword and Random Number (RN). Caution should be exercised in executing this command since the Crypto-Engine™ will be set to a *NonInitialized* State.

States

Valid State(s): *Stopped*
Result: *Zeroized*

Syntax

<Cmd> = <A4>
<Resp> = <ACK>|<INVCMD>
<Term> = <DONE>

Parameters

None.