MICROCHIP

ATECC608A

PRELIMINARY DATASHEET

Microchip CryptoAuthentication Device

Features

- Cryptographic Co-processor with Secure Hardware-based Key Storage
  - Protected Storage for up to 16 Keys, Certificates or Data
- Hardware support for Asymmetric Sign, Verify, Key Agreement
  - ECDSA: FIPS186-3 Elliptic Curve Digital Signature
  - ECDH: FIPS SP800-56A Elliptic Curve Diffie-Hellman
  - NIST Standard P256 Elliptic Curve Support
- Hardware support for Symmetric Algorithms
  - SHA-256 & HMAC Hash including off-chip context save/restore
  - AES-128: Encrypt/Decrypt, Galois Field Multiply for GCM
- Networking Key Management Support
  - Turnkey PRF/HMAC calculation for TLS 1.2 & 1.3
  - Ephemeral key generation and key agreement in SRAM
  - Small message encryption with keys entirely protected
- Secure Boot Support
  - Full ECDSA code signature validation, optional stored digest/signature
  - Optional communication key disablement prior to secure boot
  - Encryption/Authentication for messages to prevent on-board attacks
- Internal High-quality FIPS 800-90 A/B/C Random Number Generator (RNG)
- Two high-endurance monotonic counters
- Guaranteed Unique 72-bit Serial Number
- Two Interface options available
  - High-speed Single Pin Interface with One GPIO Pin
  - 1MHz Standard I^2C Interface
- 1.8V to 5.5V IO levels, 2.0V to 5.5V supply voltage
- <150nA Sleep Current
- 8-pad UDFN, 8-lead SOIC, and 3-lead CONTACT Packages

Applications

- IoT network endpoint key management & exchange
- Encryption for small messages and PII data
- Secure Boot and Protected Download
- Ecosystem Control, Anti-cloning

Pin Configuration and Pinouts

Table 1. Pin Configuration

Pin	Function
NC	No Connect
GND	Ground
SDA	Serial Data
SCL	Serial Clock Input
V_{CC}	Power Supply

Figure 1. Pinouts

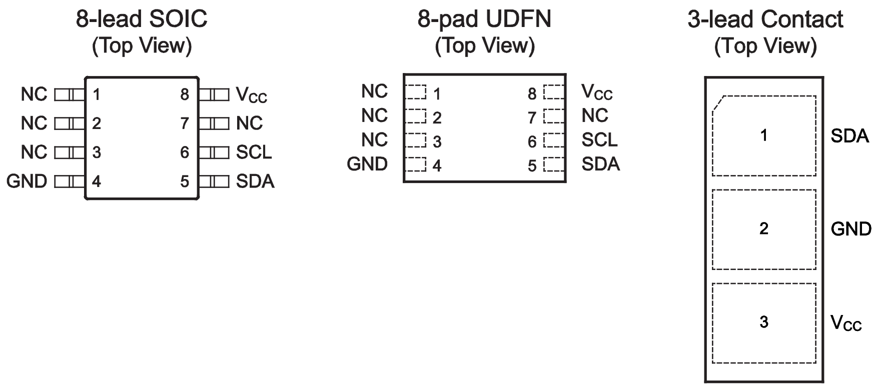


Table of Contents

1 Introduction

1.1 Applications

The Microchip® ATECC608A is a member of the Microchip CryptoAuthentication™ family of high-security cryptographic devices which combine world-class hardware-based key storage with hardware cryptographic accelerators to implement various authentication and encryption protocols.

The ATECC608A has a flexible command set that allows use in many applications, including the following:

- **Network/IoT Node Endpoint Security**
  Manage node identity authentication and session key creation & management. Supports the entire ephemeral session key generation flow for multiple protocols including TLS 1.2 (and earlier) and TLS 1.3
- **Secure Boot**
  Support the MCU host by validating code digests and optionally enabling communication keys on success. Various configurations to offer enhanced performance are available.
- **Small Message Encryption**

  Hardware AES engine to encrypt and/or decrypt small messages or data such as PII information. Supports all AES modes. Additional GFM calculation function to support AES-GCM.
- **Key Generation for Software Download**
  Supports local protected key generation for downloaded images. Both broadcast of one image to many systems, each with the same decryption key, or point-to-point download of unique images per system is supported.
- **Ecosystem control and Anti-Counterfeiting**
  Validates that a system or component is authentic and came from the OEM shown on the nameplate.

The ATECC608A is generally compatible with the ATECC508A when properly configured. See Section 12.1 for more details.

1.2 Device Features

The ATECC608A includes an EEPROM array which can be used for storage of up to 16 keys, certificates, miscellaneous read/write, read-only or secret data, consumption logging, and security configurations. Access to the various sections of memory can be restricted in a variety of ways and then the configuration can be locked to prevent changes.

Access to the device is made through a standard I^{2}C Interface at speeds of up to 1Mb/s (see Section 7, I2C Interface). The interface is compatible with standard Serial EEPROM I^{2}C interface specifications. The device also supports a Single-Wire Interface (SWI), which can reduce the number of GPIOs required on the system processor, and/or reduce the number of pins on connectors (see Section 6, Single-Wire Interface). If the Single-Wire Interface is enabled, the remaining pin is available for use as a GPIO, an authenticated output or tamper input.

Each ATECC608A ships with a guaranteed unique 72-bit serial number. Using the cryptographic protocols supported by the device, a host system or remote server can verify a signature of the serial number to prove that the serial number is authentic and not a copy. Serial numbers are often stored in a standard Serial EEPROM; however, these can be easily copied with no way for the host to know if the serial number is authentic or if it is a clone.

The ATECC608A features a wide array of defense mechanisms specifically designed to prevent physical attacks on the device itself, or logical attacks on the data transmitted between the device and the system (See Section 4.5, Security Features). Hardware restrictions on the ways in which keys are used or generated provide further defense against certain styles of attack.

1.3 Cryptographic Operation

The ATECC608A implements a complete asymmetric (public/private) key cryptographic signature solution based upon Elliptic Curve Cryptography and the ECDSA signature protocol. The device features hardware acceleration for the NIST standard P256 prime curve and supports the complete key life cycle from high quality private key generation, to ECDSA signature generation, ECDH key agreement, and ECDSA public key signature verification.

The hardware accelerator can implement such asymmetric cryptographic operations from ten to one-thousand times faster than software running on standard microprocessors, without the usual high risk of key exposure that is endemic to standard microprocessors.

The ATECC608A also implements AES-128, SHA256 and multiple SHA derivatives such as HMSC(SHA), PRF (the key derivation function in TLS) and HKDF in hardware. Support is included for the Galois Field Multiply (aka Ghash) to facilitate GCM encryption/decryption/authentication.

The device is designed to securely store multiple private keys along with their associated public keys and certificates. The signature verification command can use any stored or an external ECC public key. Public keys stored within the device can be configured to require validation via a certificate chain to speed-up subsequent device authentications.

Random private key generation is supported internally within the device to ensure that the private key can never be known outside of the device. The public key corresponding to a stored private key is always returned when the key is generated and it may optionally be computed at a later time.

The ATECC608A can generate high-quality random numbers using its internal random number generator. This sophisticated function includes runtime health testing designed to ensure that the values generated from the internal noise source contain sufficient entropy at the current time, with the current device and under the current voltage and temperature conditions. The random number generator is designed to meet the requirements documented in the NIST 800-90A, 800-90B and 800-90C documents.

These random numbers can be employed for any purpose, including usage as part of the device's crypto protocols. Because each random number is guaranteed to be essentially unique from all numbers ever generated on this or any other device, their inclusion in the protocol calculation ensures that replay attacks (i.e. re-transmitting a previously successful transaction) will always fail.

The ATECC608A also supports a standard hash-based challenge-response protocol in order to allow its use across a wide variety of additional applications. In its most basic instantiation, the system sends a challenge to the device, which combines that challenge with a secret key via the `MAC` command and then sends the response back to the system. The device uses a SHA-256 cryptographic hash algorithm to make that combination so that an observer on the bus cannot derive the value of the secret key, but preserving that ability of a recipient to verify that the response is correct by performing the same calculation with a stored copy of the secret on the recipient's system. There are a wide variety of variations possible on this symmetric challenge/response theme.

2 EEPROM Memory

The EEPROM memory contains a total of 11,200 bits and is divided into the following zones:

Table 2-1. ATECC608A Zones

Zone	Description	Nomenclature
Data	Zone of 1,208 bytes (9.7Kb) split into 16 general purpose read-only or read/write memory slots of 36 bytes (288 bits), 72 bytes (576 bits), or 416 bytes (3,328 bits) each that can be used to store keys (public or private), signatures, certificates, calibration, model number, or other information, typically that relate to the item to which the ATECC608A device is attached.	Slot[YY] = The entire contents stored in Slot YY of the Data zone.
Configuration	Zone of 128 bytes (1,024-bit) EEPROM that contains the serial number and other ID information, as well as, access the permission information for each slot of the data memory.	SN[a:b] = Arrange of bytes within a field of the Configuration zone.
One Time Programmable (OTP)	Zone of 64 bytes (512 bits) of OTP bits. Prior to locking the OTP zone, the bits may be freely written using the standard `Write` command. The OTP zone can be used to store read-only data.	OTP[bb] = A byte within the OTP zone, while OTP[aa:bb] indicates a range of bytes.

Terms discussed within this document will have the following meanings:

Table 2-2. Document Terms

Term	Meaning
Block	A single 256-bit (32-byte) area of a particular memory zone. The industry SHA-256 documentation also uses the term “block” to indicate a 512-bit section of the message input. Within this document, this convention is used only when describing hash input messages.
keyID	keyID is equivalent to the slot number for those slots designated to hold key values. Key 1 (sometimes referred to as key[1]) is stored in Slot[1] and so on. While all 16 slots can potentially hold keys, those slots which are configured to permit clear-text reads would not normally be used as private or secret keys by the crypto commands.
mode:b	Indicates bit b of the parameter mode.
SRAM	Contains input and output buffers, as well as, state storage locations. See Section 3

2.1 EEPROM Data Zone

The Data zone is broken into 16 slots, for which access restrictions are individually programmable. While all slots can be used for private or secret keys or user data, only Slots 8 thru 15 are large enough to store an ECC public key or ECDSA certificate/signature. When a slot is used for a private or secret key, the excess memory not required by the particular algorithm is in general unusable. The following table lists the typical uses for each group of slots, along with any special characteristics of slots within that group.

Table 2-3. Data Zone

Slot	Blocks[1]	Bytes	Bits	Typical Use	Notes
0 – 7	2	36	288	Private or Secret Key	Can also be used for data.
8	13	416	3328	Data	Reads and Writes can be configured to be restricted in the same manner as all other slots. If this slot is used as a key, then the remaining bytes not required for the secret or private key storage will be ignored.
9 – 15	3	72	576	Public Key, Signature or Certificate	For the curve supported by this device, these slots are large enough to contain both the X and Y components of an ECDSA public key or the R and S components of an ECDSA signature.

Note 1. The last block in some data slots contains fewer than 32 bytes.

Data slots which contain ECC public or private keys should be formatted according to Section 5.1.1, ECC Key Formatting. The device uses the keyType and pubInfo fields of KeyConfig to determine what is stored in a slot. Private keys can never be read from the device under any circumstances. ECC key slot contents may not be usable by the ECC commands unless they are validated as follows:

- **ECC Private Keys**
  Prior to the first `PrivWrite` or `GenKey(Create)` command execution on a slot, private keys are invalid. The key may also be invalid if the `PrivWrite` command is started, but power is interrupted prior to its completion.
- **ECC Public Keys**
  The key must be validated using an input signature and the ECC `Verify` command if the PubInfo bit of KeyConfig is one. If that bit is zero, then ECC usage does not depend on the key `Verify` operation. These keys may be stored in Slots 8 thru 15 *only*. This feature is optional.

Data slots may also contain AES keys. If they are to be used by the AES encrypt/decrypt command, they must be tagged with the AES data type. Either two or four keys may be stored within a slot, depending on the size of the slot.

2.1.1 Certificate Storage

Certificates of private keys stored with the ATECC608A generally take more storage than the 72 bytes (576 bits) which are available in Slots 9 thru 15. Assuming that the Certificate Authority signing key is a 256 bit ECC key, then these certificates can be efficiently stored within a single slot in the following manner:

1. The first 8 bytes of the key slot can be used to identify a template from a table stored in the host memory. This table can include static data such as the algorithm, issuer's name, use restrictions, etc. These bytes can also provide compressed versions of variable information such as generation and expiry dates, etc.
2. The public key component of the initial section of the certificate can be computed based on the private key stored within the ATECC608A.
3. The last 64 bytes can be used to store the authority's ECDSA signature of the TBS section.

Slot 8 may contain sufficient space to store a single ECC certificate in uncompressed format. Also, uncompressed certificates can be split in pieces and stored in multiple slots.

2.2 EEPROM Configuration Zone

The 128 bytes in the Configuration zone contains the manufacturing identification data, general device and system configuration information and access restriction control values for the slots within the Data zone. It is organized as four blocks of 32 bytes each. The values of these bytes can always be obtained using the `Read` command.

System designers should take great care to ensure that the access restrictions are appropriate for the information that is stored within the slot. Particular notice should be taken for those slots that are indirectly referenced – as a read/write/authorization parent, as the target of a copy function (typically calculated as KeyID | 1) or the keys used for IO protection, GPIO, CounterMatch or Secure boot.

The bytes of this zone are arranged as shown the table below:

Table 2-4. Configuration Zone

Byte	Name	Description	Write	Read
0 → 3	SN[0:3]	Part of the serial number value. See Section 2.2.2	Never	Always
4 → 7	RevNum	Device revision number. See Section 2.2.3	Never	Always
8 → 12	SN[4:8]	Part of the serial number value. See Section 2.2.2	Never	Always
13	AES_Enable	Bit 0: `0` = The AES command and AES mode of the KDF command are disabled. `1` = The AES operations are enabled. Bits 1-7: Set by Microchip and cannot be changed. The value in these bits will vary and software should not depend on any particular state.	Never	Always
14	I2C_Enable	Bit 0: `0` = The device operates in Single-Wire Interface mode. `1` = The device operates in I2C interface mode. Bits 1-7: Set by Microchip and cannot be changed. The value in these bits will vary and software should not depend on any particular state.	Never	Always
15	Reserved	Set by Microchip.	Never	Always
16	I2C_Address	When I2C_Enable:0 is one, this field is the I2C_Address with a default value of `0xC0`. See Section 2.2.1. When I2C_Enable:0 is zero the chip operates in single wire mode and this this field controls the GPIO function. See Section 2.2.3.	If Config Unlocked	Always
17	Reserved	Reserved. Must be zero.		
18	CountMatch	Counter match function control byte, See Section 2.2.1 Bit 0: Counter match enable. If 0, counter match function is disabled. Bit 1-3: Must be zero. Bits 4-7: CountMatchKey, slot to be used for counter match.	If Config Unlocked	Always

Byte	Name	Description	Write	Read
19	ChipMode	Bit 0: I2C Address UserExtraAdd Mode. See Section 2.2.1. `0`= I2C address is stored in byte 16 `1`= I2C address is in byte 85 if that is not zero Bit 1: TTLenable. See Figure 9-5 and Figure 9-6 `0` = Input levels use a fixed reference. `1` = Input levels are V_{CC} referenced. Bit 2: Watchdog Duration. Microchip recommends this be set to zero for the best security `0` = $t_{WATCHDOG}$ is 1.3s, nominal. `1` = $t_{WATCHDOG}$ is 10.0s, nominal. Bits 3-7: Clock divider. Only 00000, 01101 (0x0D) and 00101 (0x05) are legal. See Sections 0 and Section Table 9-4	If Config Unlocked	Always
20 → 51	SlotConfig	Two bytes of access and usage permissions and controls for each slot of the Data zone. See Section 2.2.10, SlotConfig (Bytes 20 to 51).	If Config Unlocked	Always
52 → 59	Counter[0]	Monotonic counter that can optionally be connected to keys via the SlotConfig.LimitedUse bit. Can count to a value of 2,097,151 and can never be decremented.	If Config Unlocked	Always
60 → 67	Counter[1]	Second monotonic counter, not connected to any keys.	If Config Unlocked	Always
68	UseLock	Configuration for transport locking. See Section 2.2.4 Bits 0-3: UseLockEnable. Bits 4-7: UseLockKey	If Config Unlocked	Always
69	VolatileKey Permission	Volatile Key Permission configuration. See Section 2.2.5. Bits 0-3: VolatileKeyPermitSlot Bits 4-6: Must be zero Bit 7: VolatileKeyPermitEnable.	If Config Unlocked	Always
70 → 71	SecureBoot	This byte controls the special secure boot functionality of the device. See Sections 2.2.6 and 4.2.	If Config Unlocked	Always
72	KdfIvLoc	Index within the KDF(HKDF) input string where the two bytes stored below (KdfIvStr) should be found. See Section 2.2.7	If Config Unlocked	Always
73 → 74	KdfIvStr	Two byte KDF IV string that must be found in the KDF message for the KDF(HKDF) special IV mode. See Section 2.2.7	If Config Unlocked	Always
75 → 83	Reserved	Must be zero	If Config Unlocked	Always
84	UserExtra	One byte value that can be modified via the `UpdateExtra` command after the Data zone has been locked. Can be written via UpdateExtra only if it has a value of zero.	Via Update Extra Cmd Only	Always
85	UserExtraAdd	If non-zero, the I2C address to which this device will respond on the bus. Can be written via UpdateExtra only if it has a value of zero.	Via Update Extra Cmd Only	Always
86	LockValue	Controls the ability to write the OTP and Data zones. `0x55` = unlocked; `0x00` = locked. See Section 2.4, EEPROM Locking.	Via Lock Command Only	Always

Byte	Name	Description	Write	Read
87	LockConfig	Controls the ability to modify the Configuration zone. `0x55` = unlocked; `0x00` = locked. See Section 2.4, EEPROM Locking.	Via Lock Command Only	Always
88 – 89	SlotLocked	A single bit for each slot. If the bit corresponding to a particular slot is zero, the contents of the slot *cannot* be modified under any circumstances. See Section 2.4.3	If Config Unlocked, Via Lock Command	Always
90 – 91	ChipOptions	Bit 0: Power On Self Test. If 1, the SHA, AES and RNG self tests will be automatically executed on wake or power-on. Bit 1: IO Protection Key Enable. See Section 2.2.8 Bit 2: KDF AES Enable. If 1, the AES mode of KDF is enabled. If 0, the KDF command can only be run with the PRF and HKDF algorithms. Bits 3-7: Must be zero Bits 8-9: ECDH Protection Bits. See Section 2.2.8 Bits 10-11: KDF Protection Bits. See Sections 2.2.7 and 2.2.8 Bits 12-15: IO Protection Key. See Section 2.2.8	If Config Unlocked	Always
92 – 95	X509format	X.509 certificate validation formatting. See Section 2.2.9	If Config Unlocked	Always
96 – 127	KeyConfig	Two bytes of additional access and usage permissions and controls for each slot of the Data zone. See Section 2.2.11, KeyConfig.	If Config Unlocked	Always

2.2.2 Serial Number (Bytes 0-2 and 8-12)

Nine bytes (SN[0:8]) that together form a unique 72 bit value that is never repeated for any device in the CryptoAuthentication family. It can be used for any purpose by external firmware and can be the basis of a diversified key calculation.

The serial number can never be written under any circumstances and can always be read, regardless of the state of the lock bits. The serial number is divided into two groups:

SN[0:1] and SN[8]
: The values of these bits are fixed at manufacturing time in most versions of the ATECC608A. Their default value is `0x01 23 EE`. These 24 bits are usually included in the hash computations that the ATECC608A makes.

SN[2:3] and SN[4:7]
: The values of these bits are programmed by Microchip during the manufacturing process and are different for every die. These 48 bits are optionally included in some cryptographic computations that are made by the ATECC608A.

2.2.3 Revision Number (Bytes 4-7)

Four bytes of information that are used by Microchip to provide manufacturing revision information. These bytes can be freely read but they should never be used by system software because they may be revised by Microchip from time to time.

2.2.1 I2C Address (Byte 16)

When the device is configured to use the I^2C interface then either byte 16 or 85 contains the device address to which this ATECC608A will respond. The selection of which byte to use is contained in the first bit of Chip Mode (Byte 19).

In those situations in which the configuration zone is locked prior to insertion in the end system and there are multiple ATECC608A devices in the system, it may be necessary to be able to select among a set of ATECC608A devices. This can be accomplished by using the UpdateExtra command to write a non-zero value to location 85. Once written to 85, the ATECC608A should be powered off or put to sleep. On wake or power-on, it will then respond to the address programmed in byte 85.

Regardless of the value of ChipMode:Bit 0, the ATECC608A will use the I2C address in byte 18 as long as byte 85 contains a value of zero. Once byte 85 has been written to a value other than 0, it can no longer be written. There is no method to program the ATECC608A to respond to a device address of 0.

Regardless of the source of the address, the LSB of the address byte is always ignored by the ATECC608A.

2.2.2 GPIO Control (Byte 16)

When the device is configured to operate in single wire interface mode, these bits are used to control the GPIO pin. (see Section 8).

Bits 0-1: *GPIO Mode*

00 = Disabled. SCL pin is unused should be tied low on the board.

01 = Authorization Modes, bit 3 determines device operation.

10 = Input. Current value on the SCL pin returned by Info command.

11 = Output. SCL may be driven high or low by Info command.

Bit 2: *GPIO Default*. Default state of SCL pin on power-up when configured as an output.

Bit 3: *GPIO Auth Mode*. Selects between the authorization modes. Must be zero if IO_Mode is not 01.

0 = Authorization Output mode. When an authorization is successfully performed on the slot in SignalKey, the SCL pin is asserted.

1 = Intrusion Detection mode. Persistent latch is set via authorization and cleared if SCL falls.

Bits 4-7: *SignalKey*. If IO_Mode is 01, the slot number for the GPIO authorizing key. For all other modes, must be 0000.

2.2.3 Counter Match Key (Byte 18)

The counter match function in the ATECC608A provides a mechanism of altering the limit to which the first monotonic counter (Counter 0) can be incremented. Key usage can be connected to counter 0 (See section 4.4.5) to prevent their use when the counter is at its limit.

If the counter match enable bit in this byte is zero, then the counter match function is disabled and the counter has to reach its natural limit before the keys are disabled.

The *CounterMatchKey* field in this byte defines the key slot which contains the value to compare with the current counter value. There are a few details to be followed when writing a limit value to this slot:

- The least significant 5 bits of the value in the key slot is ignored. This means that the granularity of a change in the limit is 32 iterations. The limit value should be written as a 32 bit number with the least significant 5 bits as 0 and a net value no greater than the maximum count as defined below in Section 4.4.5.
- To provide reliable operation, this 4 byte (32 bit) value should be repeated in the second 4 bytes of the key slot.
- The *CounterMatchKey* slot must not have the IsSecret bit set. It is still fine to require both encryption and MACs on the write command for this slot.

2.2.4 Use Lock (Byte 68)

This byte controls the transport locking function of the ATECC608A. When this function is enabled, general purpose use of the device is prohibited until the device is cryptographically enabled. When locked, only the Read, Nonce and CheckMac commands are permitted.

To enable the device, the host system should use the Nonce command to generate a nonce value. That nonce should then be combined using SHA following the sequence described in the CheckMac command below. The resulting digest should be sent to the CheckMac command with the source key pointing to UseLockKey. If the ATECC608A can validate the digest, then it will automatically clear UseLockEnable and allow normal use.

The bits in this byte are encoded as follows:

Bits 0-3: *UseLockEnable*. If 0x0A, then only Read, Nonce and CheckMac commands are permitted. If any other value, then this function is not enabled, the UseLockKey field is ignored and all commands complete normally.

Bits 4-7: *UseLockKe*y. The slot ID of the key that can be used with CheckMac to enable full use of the chip

The following two best practices should be observed in order to obtain the appropriate security:

- Set the ReqRandom bit in KeyConfig for UseLockKey. This will require that each device be enabled via a separate transaction with the host system using a random number generated within the ATECC608A chip. If this bit is not set, then the identical sequence can be sent to every device to enable it, leaving open the possibility of an attack on the host enablement system and the loss of that sequence.
- Configure the UseLockKey slot to allow DeriveKey(Roll) without authorization – WriteConfig = 0010. After the device has been unlocked, then the authorizing key can be destroyed via combination with a random number (generated in any manner the host may desire) on the input, preventing any possibility of future use.

2.2.5 Volatile Key Usage Permission (Byte 69)

This byte controls the volatile permission function of the ATECC608A. When this function is enabled for a particular key slot, general purpose use of that slot will be prohibited until cryptographically enabled. The permission status is stored in the persistent latch and is retained during sleep, idle and active modes. See Section 3.5 for more information on the Persistent Latch.

One use of this function is to connect a particular ATECC608A to a particular host MCU on a board. On power-up, the host would enable normal key usage during boot and continue operation as before. If the ATECC608A is removed from the board, then without knowledge of the shared key, the remaining keys on the ATECC608A would be unusable.

The secret used to facilitate the Volatile Key Permission should be a unique value that is stored in both the MCU and the ATECC608A at board manufacturing. One fine option is to share this key with the key used for IO Protection. This key can be generated via the ATECC608A random number generator and stored within a key slot and then slot locked. The same value should be stored in a safe place within the MCU.

This function may also allow factory enablement of keys within systems that contain a battery. If the device is removed from the system, use of the keys will not be permitted.

To enable the keys, the host system should use the Nonce command to generate a nonce value. That nonce should then be combined using SHA following the sequence described in the CheckMac command below. The resulting digest should be sent to the CheckMac command with the source key pointing to VolKeyPermitSlot. Then the Info(Mode=4) command can be run to write the Persistent Latch to a value of 1.

The bits in this byte are encoded as follows:

Bits 0-3: *VolKeyPermitSlot*. The slot containing the key which is used to set or clear the Persistent Latch via the Info command.

Bits 4-7: Must be zero.

Bit 7: *VolKeyPermitEnable*. If 0, then this function is disabled and control of the Persistent Latch may occur via other means. If 0, VolKeyPermitSlot is ignored.

Best practice is to set the ReqRandom bit in KeyConfig for VolKeyPermitSlot. This will require that the keys be enabled via a separate transaction with the host system using a random number generated within the ATECC608A chip. If this bit is not set, then the enable sequence can be stored by an attacker and replayed in a different environment.

2.2.6 Secure Boot Configuration (Bytes 70-71)

These two bytes control the operation of the secure boot functionality of the device. In general, the SecureBoot command makes use of these configuration bits to ensure that the proper sequences are executed. See Section 4.2 for a complete description of the secure boot capability supported by the device.

Bits 0-1: *SecureBootMode*

00: Secure boot functionality disabled.

01: Full Secure boot (FullBoth). Both digest and signature must be passed to the chip on boot.

10: Stored Secure Boot (FullSig). After first successful secure boot, signature is stored and subsequent boots do not require signature to transmitted

11: Stored Secure Boot (FullDig). After first successful secure boot, digest is stored and on subsequent boots the digest is compared without ECC verify

Bits 2: Must be zero.

Bit 3: *SecureBootPersistentEnable*. If 1, then on successful execution of the SecureBoot command, the persistent latch will be set. If key usage is tied to the Persistent Latch via the PersistentDisable bit in KeyConfig, then those keys will not be usable until secure boot has successfully completed.

Bit 4: *SecureBootRandNonce*. If 1, then the input code digest for the SecureBoot command must be encrypted and the nonce used for the digest encryption must use the ATECC608A random number generator. When the input is encrypted, the output always includes a MAC. If 0, input encryption is optional and is controlled solely via the mode bit of the SecureBoot command. If 0, use of the ATECC608A random number generator is not required.

Bits 5-7: Must be zero.

Bits 8-11: *SecureBootSigDig*. Slot number containing the stored digest or signature to be used during secure boot if a previous secure boot has validated the digest or signature.

Bits 12-15: *SecureBootPubKey*. Slot number containing the public key used for validation of the code signature.

2.2.7 Special KDF Initialization Vector Function (Bytes 72-74)

The result of the KDF command computation can be prevented from being sent to the host MCU in the clear via the IO Protection field (below) if so configured. While this is a secure method for use it is not optimal for the HKDF initialization vector values. This is important when using the on-chip AES engine because that command requires that the mode handling for any mode other than ECB be performed externally in the host MCU.

This special function allows the KDF command to bypass that the IO protection field and allow the KDF result to be placed in the output buffer in the clear if all of the following conditions are met:

1. The KDF algorithm must be HKDF and the mode should specify that the output is in the clear
2. The length of the input string must be at least as long as config.KdfIvLoc + 2
3. The two bytes starting at index config.KdfIvLoc must match the two bytes stored at config.KdfIvStr

This function can be disabled by setting the index value (config.KdfIvLoc) to 0xF0, which exceeds the size of the input buffer and will never result in a legal match.

2.2.8 IO Protection (ChipOptions : Bytes 90-91)

The ATECC608A provides a method to protect the IO transmissions between it and the host MCU for the ECDH, KDF, Verify and SecureBoot commands. Other commands provide protection for the result as described elsewhere.

The remaining bits within the Chip Options word are used for other functions, this section includes only those bits which pertain to the IO protection function.

It is expected that a typical implementation would involve generation of a random number at the first power-up of the board containing the ATECC608A and the host. This shared secret would be written into both the IO Protection key slot and a relatively secure place within the host MCU. Once written it would be slot locked on the ATECC608A to prevent any changes. In such an implementation, every board would have a different secret protecting the transmissions between the ATECC608A and host. Due to this, any possible attack on one system could not be directly translated to a second.

For the ECDH and KDF commands, IO protection involves encrypting the data transmitted from the ATECC608A to the host MCU. For the Verify command, IO protection involves returning a validation MAC with the output to ensure that a man-in-the-middle has not modified the result from the chip. The SecureBoot operation involves both encryption and authentication.

Bit 1: *IO Protection Enable*. If 1, then the protection functions are enabled via the secret key stored in the IO Protection key slot. If 0, this function is disabled and the three fields below are ignored by the chip.

Bits 8-9: *ECDH Prot*. Restrictions on the way the ECDH command result can be used:

00 = Output in the clear is OK, though encryption can still be indicated in the mode parameter

01 = Output is OK, but the result must be encrypted. The state of the encryption bit in the mode parameter will be ignored by the ECDH command.

10 = Result must be stored in TempKey or and EEPROM key slot, output on the bus is forbidden.

11 = Do not use.

Bits 10-11: *KDF Prot*. Restrictions on the way the KDF command result can be used, coded identically as for ECDH above.

Bits 12-15: *IO Protection key*. KeyID to be used for IO protection. SlotConfig.NoMac must be set to 0 for this key.

2.2.9 X.509 Certificate Validation Format (Bytes 92-95)

Most X.509 certificates are too large to store in the slots of the ATECC608A. The device does, however, provide a method to store the public keys contained within an X.509 certificate and to prevent their use until the certificate has been validated. This sequence is controlled via the SHA(Public) and Verify(ValidateExternal) commands.

The four individual bytes within this format section are associated with the X.509 certificate formatting appropriate for specific of public keys stored within the device. If the value of the byte associated with a particular public key is zero, then these formatting restrictions are ignored and that public key can be validated with `Verify(Validate)`. Unused bytes within this array must be zero, otherwise, the formatting must be as follows:

Bits 0 – 3: PublicPosition. The SHA block number in which the public key must be inserted in the SHA sequence for the Verify(ValidateExternal) command to properly validate a public key.

Bits 4 – 7: TemplateLength. The total number of blocks in the entire SHA sequence which are required for the Verify(ValidateExternal) command to properly validate a public key.

2.2.10 SlotConfig (Bytes 20 to 51)

The 16 SlotConfig elements are used to configure the access protections for each of the 16 slots within the ATECC608A device. Each configuration element consists of 16 bits, which control the usage and access for that particular slot or key. The SlotConfig field is interpreted according to the following table when the Data zone is locked. When the Data zone is unlocked, *these restrictions generally do not apply*, and those slots not configured to contain private keys may freely be written and none may be read.

Table 2-5. SlotConfig Bits (Per Slot)

Bit	Name	Description
0 → 3	ReadKey	Use this keyID to encrypt data being read from this slot using the Read command. See more information in the description for bit 6 in Table 2-6. 0 = Then this slot can be the source for the CheckMac copy operation. See Section 4.4.6, Password Checking. ▶ Do not use zero as a default. Do not set this field to zero unless the **CheckMac** copy operation is explicitly desired, regardless of any other read/write restrictions. Slots containing private keys can *never* be read and this field has a different meaning: Bit 0: External signatures of arbitrary messages are enabled. Bit 1: Internal signatures of messages generated by GenDig or GenKey are enabled. Bit 2: ECDH operation is permitted for this key. Bit 3: If clear, then ECDH master secret will be output in the clear. If set, then master secret will be written into slot N|1. Ignored if Bit 2 is zero. For slots containing public keys that can be validated (PubInfo is one, see Section 2.2.11, KeyConfig), this field stored the ID of the key that should be used to perform the validation.
4	NoMac	1 = The key stored in the slot is intended for verification usage and cannot be used by the MAC command. When this key is used to generate or modify TempKey, then that value may not be used by the MAC command. 0 = The key stored in the slot can be used by all commands.
5	LimitedUse	1 = The key stored in the slot is "Limited Use" and its use is controlled by Counter0. See Section 4.4.5, High Endurance Monotonic Counters. 0 = There are no usage limitations.
6	EncryptRead	1 = Reads from this slot will be encrypted using the procedure specified in the **Read** command using ReadKey (bits 0 – 3 in this table) to generate the encryption key. No input MAC is required. If this bit is set, then IsSecret must also be set (in addition, see the following Table 2-6). 0 = Clear text reads may be permitted.
7	IsSecret	1 = The contents of this slot are secret – Clear text reads are prohibited and both 4-byte reads and writes are prohibited. This bit must be set if EncryptRead is a one or if WriteConfig has any value other than *Always* to ensure proper operation of the device. 0 = The contents of this slot should contain neither confidential data nor keys. The GenKey and Sign commands will fail if IsSecret is set to zero for any ECC private key. See Table 2-6 for additional information.
8 → 11	WriteKey	Use this key to validate and encrypt data written to this slot
12 → 15	WriteConfig	Controls the ability to modify the data in this slot. See Table 2-7, Table 2-8, Table 2-10, and 11.23.

2.2.10.2 Read Permissions

Read operations for most data slots are controlled by the state of IsSecret and EncryptRead, according to the following table. ECC private keys can never be read under any circumstances.

Table 2-6. Read Operation Permission

IsSecret	EncryptRead	Description
0	0	Clear text reads are always permitted from this slot. Slots set to this state should *never* be used as key storage. Either 4 or 32 bytes may be read at a time.
0	1	Prohibited. No security is guaranteed for slots using this code.
1	0	Reads are never permitted from this slot. Slots set to this state can still be used for key storage.
1	1	Reads from this slot are encrypted using the encryption algorithm documented in the Read command section below. The encryption key is in the slot specified by ReadKey. 4-byte reads and writes are prohibited.

2.2.10.3 Write Permissions

The 4-bit WriteConfig field is interpreted by the `Write`, `DeriveKey`, and `GenKey` commands as shown in Table 2-7, Table 2-8, and Table 2-10 where “X” means don't care.

The tables overlap: for example, a code of `0110` indicates a slot which can be written in encrypted form using the `Write` command and can also be the target of an unauthorized `DeriveKey` command with the target as the source.

KeyType in the KeyConfig field (see Table 2-7) indicates whether the `GenKey` or `DeriveKey` commands can be used on a particular slot; with `GenKey` for ECC keys only, and `DeriveKey` for SHA-256 keys.

See Section 2.2.10.4, Writing ECC Private Keys for special information regarding the writing of ECC private keys. ECC public keys are treated as normal data, and Write permissions for those slots are described in this section.

Table 2-7. Write Configuration Bits: `Write` Command

Bit 15	Bit 14	Bit 13	Bit 12	Mode Name	Description
0	0	0	0	Always	Clear text writes are always permitted on this slot. Slots set to always should never be used as key storage. Either 4 or 32 bytes may be written to this slot.
0	0	0	1	PubInvalid	If a validated public key is stored in the slot, writes are prohibited. Use `Verify(Invalidate)` to invalidate prior to writing. Do not use this mode unless slot contains a public key. See Section 4.3
0	0	1	X	Never	Writes are never permitted on this slot using the `Write` command. Slots set to never can still be used as key storage.
1	0	X	X	Never	Writes are never permitted on this slot using the `Write` command. Slots set to never can still be used as key storage.
X	1	X	X	Encrypt	Writes to this slot require a properly computed MAC, and the input data must be encrypted by the system with WriteKey using the encryption algorithm documented in the `Write` command description. 4-byte writes to this slot are prohibited.

Table 2-8. Write Configuration Bits: `DeriveKey` Command

Bit 15	Bit 14	Bit 13	Bit 12	Source Key[1]	Description
0	X	1	0	Target	`DeriveKey` command can be run without authorizing MAC. (Roll)
1	X	1	0	Target	Authorizing MAC required for `DeriveKey` command. (Roll)
0	X	1	1	Parent	`DeriveKey` command can be run without authorizing MAC. (Create)
1	X	1	1	Parent	Authorizing MAC required for `DeriveKey` command. (Create)
X	X	0	X	—	Slots with this value in the WriteConfig field may *not* be used as the target of the `DeriveKey` command.

Note 1. The source key for the computation performed by the `DeriveKey` command can either be the key directly specified in Param2 (Target) or the key at slotConfig[Param2].WriteKey (Parent).

The IsSecret bit controls internal circuitry necessary for proper security for slots in which reads and/or writes must be encrypted or are prohibited altogether. It must also be set for all slots that are to be used as keys, including those created or modified with the `DeriveKey` command. Specifically, to enable proper device operation, this bit must be set unless WriteConfig is *Always*. An exception is that the CounterMatch slot must have IsSecret set to 0, but may have any legal WriteConfig value. Four byte accesses are generally prohibited to and from slots in which this bit is set.

Slots used to store key values should always have IsSecret set to one and EncryptRead set to zero (reads prohibited) for maximum security. For fixed key values, WriteConfig should be set to *Never*. When configured in this way, after the Data zone is locked, there is no way to read or write the key; and it may only be used for crypto operations.

Some security policies require that secrets be updated from time to time. The ATECC608A supports this capability in the following way: WriteConfig for the particular slot should be set to *Encrypt* and SlotConfig.WriteKey should point back to the same slot by setting WriteKey to the slot ID. A standard `Write` command can then be used to write a new value to this slot, provided that the authentication MAC is computed using the old (i.e. current) key value.

2.2.10.4 Writing ECC Private Keys

ECC private keys are designated via the appropriate contents of KeyConfig.KeyType and KeyConfig.PubInfo. They can never be written with the `Write` and/or `DeriveKey` commands. Instead, `GenKey` and `PrivWrite` can be used to modify these slots. It is always an error to attempt to execute `GenKey` or `PrivWrite` on a slot that is not configured to contain an ECC private key. SlotConfig.WriteConfig has the following interpretations for these commands:

Table 2-9. Write Configuration Bits: `GenKey` Command

Bit 15	Bit 14	Bit 13	Bit 12	Description
X	X	0	X	`GenKey` may not be used to write random keys into this slot.
X	X	1	X	`GenKey` may be used to write random keys into this slot.

Table 2-10. Write Configuration Bits: `PrivWrite` Command

Bit 15	Bit 14	Bit 13	Bit 12	Mode Name	Description
X	0	X	X	Forbidden	`PrivWrite` will return an error if the target key slot has this value.
X	1	X	X	Encrypt	Writes to this slot require a properly computed MAC and the input data must be encrypted by the system with SlotConfig.WriteKey using the encryption algorithm documented in the `PrivWrite` command description.

2.2.11 KeyConfig (Bytes 96 thru 127)

The 16 KeyConfig elements are used in addition to SlotConfig to restrict the actions that can be performed using information stored in a particular slot. The KeyConfig element is interpreted according to the table below when the Data zone is locked. When the Data zone is unlocked, these restrictions do not apply, with the exception that slots configured to contain private keys can be written only with the `PrivWrite` command.

Table 2-11. KeyConfig Bits (Per Slot)

Bit	Name	Description
0	Private	1 = The key slot contains an ECC private key and can be accessed only with the Sign, GenKey, and PrivWrite commands. 0 = The key slot does not contain an ECC private key and cannot be accessed with the Sign, GenKey, and PrivWrite commands. It may contain an ECC public key, a SHA key, or data.
1	PubInfo	If Private indicates this slot contains an ECC private key: 0 = The public version of this key can *never* be generated. Use this mode for the highest security. 1 = The public version of this key can always be generated. If Private indicates that this slot does *not* contain an ECC private key, then this bit may be used to control validity of public keys. If so configured, the Verify command will only use a stored public key to verify a signature if it has been validated. The Sign and Info commands are used to report the validity state. The public key validity feature is ignored by all other commands and applies only to Slots 8 – 15. 0 = The public key in this slot can be used by the Verify command without being validated. 1 = The public key in this slot can be used by the Verify command only if the public key in the slot has been validated. When this slot is written for any reason, the most significant four bits of byte 0 of block 0 will be set to 0xA to invalidate the slot. The Verify command can be used to write those bits to 0x05 to validate the slot. If this slot contains a key of type Data or AES, then the PubInfo bit controls whether or not the KDF command write data into this slot. If 1, then writes by KDF are allowed. If 0, KDF may not write to this slot.
2 → 4	KeyType	If the slot contains an ECC public or private key, then the key type field below must be set to 4. If the slot contains any other kind of data, key, or secret, then this field *must* be set to another value for proper operation, as listed below. 0-3 = RFU (reserved for future use) 4 = P256 NIST ECC key 5 = RFU (reserved for future use) 6 = AES key 7 = SHA key or other data

Bit	Name	Description
5	Lockable	1 = Then this slot can be individually locked using the Lock command. See the SlotLocked field in the Configuration zone to determine whether a slot is currently locked or not. 0 = Then the remaining keyConfig and slotConfig bits control modification permission. Applies to all slots, regardless of whether or not they contain keys. See Section 2.4, EEPROM Locking.
6	ReqRandom	This field controls the requirements for random nonces used by the following commands: `GenKey`, MAC, `CheckMac`, `Verify`, `DeriveKey`, and `GenDig`. `1` = A random nonce is required. `0` = A random nonce is *not* required.
7	ReqAuth	`1` = Before this key must be used, a prior authorization using the key pointed to by AuthKey must be completed successfully prior to cryptographic use of the key. Applies to all key types, both public, secret, and private. See Section 4.4.8, Authorized Keys. `0` = No prior authorization is required.
8 →11	AuthKey	If ReqAuth is one, this field points to the key that must be used for authorization before the key associated with this slot may be used. Must be zero if ReqAuth is zero.
12	PersistentDisable	`1` = Use of this key is prohibited for all commands other than `GenKey` if the PersistentLatch is zero. `GenKey` is permitted regardless of the state of the latch. `0` = Then use of this key is independent of the state of the PersistentLatch.
13	RFU	Must be zero.
14 →15	X509id	The index into the X509format array within the Configuration zone (addresses 92-95) which corresponds to this slot. If the corresponding format byte is zero, then the public key can be validated by any format signature by the parent. If the corresponding format byte is non-zero, then the validating certificate must be of a certain length; the stored public key must be located at a certain place within the message and the `SHA()` commands must be used to generate the digest of the message. Must be zero if the slot does not contain a public key.(1)

More information on select fields is described below.

- **Private**
  This bit indicates that the slot contains an ECC private key and it is used by the device to limit uses of this slot to the appropriate ECC commands.

  If this bit is set, then SlotConfig.ReadKey is used to enable or disable the use of the private key for various operations. ReadKey:0 enables the use of the key for signatures of externally supplied data, while ReadKey:1 enables the use of the key to sign only messages that are stored in TempKey by the `GenKey` or `GenDig` commands. This mechanism permits a remote entity to have the knowledge that a particular key value or slot contents are stored within an ATECC608A device, and it prevents an attacker from creating an external message that would model an internal state that does not exist and create a signature of that state.

- **PubInfo**
  For public keys, this field can be used to walk a certificate chain to validate the key. This feature is implemented using the `Verify` command and the validation is stored in nonvolatile memory alongside the key so that subsequent uses of the public key do not require additional validation. These keys are always invalidated when any part of the slot containing the key is written.

  For private keys, this field can be used to increase security or privacy in some situations by preventing the generation of the public key corresponding to a private key. The presumption is that the public key has been stored elsewhere at the time the private key was generated or written into the device. This field is ignored when a random key is generated. The ATECC608A includes a method of walking either an X.509 certificate chain or a simplified internal format chain. See the `SHA` and `Verify(ValidateExternal)` commands for more details.

  `If KeyType is Data or AES, then PubInfo controls whether or not the KDF command may write data into this slot. If 1, then writes by KDF are allowed, if 0, KDF writes are not permitted.`

- **KeyType**
  The four ECC commands that use ECC keys (i.e. `GenKey`, `Sign, ECDH, and Verify`) will operate only on data slots in which this field is set to one of the legal ECC key types. Any attempt to use any symmetric computation commands (i.e. `CheckMac`, `DeriveKey`, `MAC`, or `AES`) on a slot configured to be an ECC private key will result in an error.

  Keys that will be the source or destination of the symmetric computation commands should be set to a KeyType of 6 (AES) or 7 (Data) as appropriate. Proper operation of the device is *not* guaranteed if these commands are attempted with any other KeyType. The `GenDig` command may operate on any slot type other than for ECC private keys.

- **ReqRandom**
  This field is useful in preventing replays of authorization and/or other cryptographic operations. Keys that control encrypted reads and/or writes should have this field set to one under normal circumstances in order to provide data security.

  If this field is set to one, then prior to the execution of the `CheckMac`, `GenDig`, `DeriveKey`, `Verify` and `MAC` commands, the Random Number Generator (RNG) must have been used by the `Nonce` command to generate the contents of TempKey.

  If `GenKey` is used to generate a public key digest of either a public or private key stored in a Data zone slot, then the ReqRandom field is used to ensure that the nonce in TempKey included the RNG.

- **ReqAuth**
  If this bit is set, then prior authorization of the key at KeyConfig.AuthKey must have been completed prior to execution of any cryptographic command (i.e. `CheckMac`, `DeriveKey`, `GenDig`, `GenKey`, `MAC`, `Sign`, or `Verify`) that uses this key. The `DeriveKey` command checks for usage authorization only for the parent key, and never the target key, unless it is the same as the parent key.

  The `GenKey` command checks for usage authorization even when generating a new key to prevent denial of service attacks.

  The authorization state is stored in two internal volatile registers:

  – AuthComplete.valid
  – AuthComplete.keyId

  These registers are retained as long as power is applied, and the device does *not* enter the Sleep mode.

  These registers are set by means of the execution of a successful `CheckMac` or `Verify` command with the key to be authorized as the target key of the command. `CheckMac` must be run with mode:1 set to one, or `Verify` must be run in Stored mode to set AuthComplete.valid. AuthComplete.keyId is set to the value in the

KeyID parameter to these commands. The `CheckMac` and `Verify` commands do not clear these bits on an unsuccessful authorization attempt unless the keys also happen to be used as the source key.

AuthComplete.valid is cleared under the following situations:

- The device enters sleep mode or power is removed.
- Any command is executed that uses a key requiring prior authorization, regardless of which slot has been authorized and/or which slot was required to be authorized for this key. If there are multiple state or configuration errors preventing the proper execution of the command, then authComplete.valid may or may not be cleared depending upon the specific error conditions encountered.

2.3 EEPROM One Time Programmable (OTP) Zone

The OTP zone of 64 bytes (512 bits) is part of the EEPROM array and can be used for read-only storage. It is organized as two blocks of 32 bytes each. The data cannot be modified and is normally used to store fixed model numbers, calibration information, manufacturing history, or other data that should never change.

Prior to locking the Configuration zone (by using lockConfig), the OTP zone is inaccessible and can be neither read nor written. After configuration locking, but prior to locking of the OTP zone (using lockValue), the entire OTP zone can be written using the `Write` command. Prior to locking the Data/OTP zones using LockValue, this zone cannot be read at all.

Once the OTP zone is locked, the contents can always be read but never written. Attempts to use the `Write` command will always return an error and leave the memory unmodified. All 64 bytes within the OTP zone are always available for reading using either 4 or 32 byte reads.

All OTP bits have a value of one upon shipment from the Microchip factory.

2.4 EEPROM Locking

There are two separate lock states for the device:

- One to lock the Configuration zone (that is controlled by LockConfig, byte 87).
- One to lock both the OTP and Data zones (that are controlled by LockValue, byte 86).

These locks are stored within separate bytes in the Configuration zone, and they can be modified only by means of the `Lock` command. After a memory zone is locked, there is no way to unlock it. Both bytes are set to 0x55 (Unlocked) on shipment from Microchip

In addition to the two device locking options described above, there is an individual slot locking mechanism that can optionally be configured for every slot. This locking operation can take place at any time before or after the chip locking operations have occurred.

The device should be personalized at the system manufacturer's site with the desired configuration information; after which, the Configuration zone should be locked. Then, all necessary writes of public and secret information into the data and OTP zones should be performed by using encrypted writes, if appropriate, and then the data and OTP zones should be locked.

<u>*It is vital that the Data and OTP zones be locked prior to release into the field of the system containing the device.*</u> Failure to lock these zones may permit modification of any secret keys and may lead to other security problems.

Any attempt to read *or* write the Data or OTP zones prior to locking the Configuration zone causes the device to return an error.

Contact Microchip for optional secure personalization services.

2.4.1 Configuration Zone Locking

Certain bytes within the Configuration zone can never be modified in the field regardless of the lock status, per Table Table 2-4. Write permission for most of the remaining bytes within the zone is controlled using the LockConfig byte in the Configuration zone as shown in the table below. Throughout this document, if LockConfig is `0x55`, the Configuration zone is said to be unlocked; otherwise, it is locked.

Table 2-12. Configuration Zone Locking

	Read Access	Write Access
`LockConfig == 0x55` (unlocked)	Read	Write
`LockConfig != 0x55` (locked)	Always	<Never>(1)

Note 1: See Table 2.2 above for limited exceptions.

2.4.2 Data and OTP Zone Locking

Once the configuration zone has been locked, secret and/or read-only data can be written into the slots of the data zone and the OTP zone. Most write access restrictions are ignored when the data zone is unlocked, when the Data/OTP zones are locked, the values in the config zone control read and write access. Throughout this document, if LockValue is `0x55`, then both the OTP and Data zones are said to be unlocked; otherwise, they are locked.

There is neither read nor write access to the OTP and Data zones prior to locking of the Configuration zone.

Table 2-13. Data Zone Access Restrictions

	Read Access	Write Access
`LockValue == 0x55` (unlocked)	<Never>	Always
`LockValue != 0x55` (locked)	Read(1)	Write(1)

Note 1: Once locked, writes to the Data Zone are controlled via the IsSecret, EncryptRead, WriteConfig and SlotLock bits for this particular slot. Prior to locking, these configuration bits are generally ignored.

Table 2-14. OTP Zone Access Restrictions

	Read Access	Write Access
`LockValue == 0x55` (unlocked)	<Never>	Always
`LockValue != 0x55` (locked)	Always	<Never>

2.4.3 Individual Slot Locking

ATECC608A provides a mechanism for one-time locking of any of the 16 data slots. Once a slot is individually locked, the slot can no longer be modified under any circumstances. This mechanism is controlled by the 16 bit field SlotLocked in the Configuration zone and the 16 Lockable bits within the 16 keyConfig words. The SlotLocked and Lockable bits can be freely written using the `Write` command prior to locking of the Configuration zone.

- **SlotLocked Bits**
  After the Configuration zone is locked, if the SlotLocked bit for a particular slot is set to zero, then

modification of that slot via the `PrivWrite`, `Write`, `GenKey`, and/or `DeriveKey` commands is permanently prohibited, regardless of the state of the corresponding Lockable, SlotConfig and/or KeyConfig bits. When SlotLocked is zero, then the corresponding slot cannot be written even if the Data zone is unlocked.

- **Lockable Bits**
  After the Configuration zone is locked, the state of the Lockable bit for a particular slot controls whether or not the `Lock` command will be permitted to change the SlotLocked bit for the corresponding slot, per the table below. If Lockable is one, then the `Lock` command can be used to modify the SlotLocked bit either before or after the Data zone is locked.

Table 2-15. Individual Slot Locking After Configuration Zone is Locked

SlotLocked Bit	Lockable Bit	Command	`PrivWrite`, `Write`, `DeriveKey`, ECDH, KDF and GenKey Commands	Notes
0	0 or 1	No	No	Not writeable.
1	0	No	Yes	Writeable but not lockable.
1	1	Yes	Yes	Writeable and lockable.

Individually lockable slots can contain either secret information or readable data and may be used in one of two ways:

- The Configuration zone and non-lockable data slots should be initialized and locked in the usual manner by the OEM. After the Data zone has been locked, those particular slots marked as lockable can then be modified and individually locked in the field at some point in the future.
- After the Configuration zone is locked, some slots can be personalized and locked by the OEM prior to transfer of the device/component to a second party such as a subcontractor or distributor that personalizes the remaining slots, and then locks the Data zone prior to shipment of the device into the field.

The `Lock` command does not provide a CRC validation mechanism when using the individual slot locking mechanism. If slots are locked prior to locking of the entire Data zone, then the contents may be validated at the time of data/OTP locking. After the Data zone is locked, either the `Read`, `CheckMac`, or `MAC` commands can be used to validate the slot contents prior to individual slot locking.

Validation of a public key via the `Verify` command can occur regardless of the state of the SlotLocked bit for that slot.

3 Static RAM (SRAM) Memory

The device also includes an SRAM array that is used to store the input command or output result, nonces, intermediate computation values, ephemeral keys, the SHA context, etc.. The entire contents of this memory are always invalidated whenever the device goes into sleep mode or the power is removed.

3.1 TempKey

TempKey is the primary storage register in the SRAM array that can be used to store various intermediate values. The contents of this register can never be read from the device (although the device itself can read and use the contents internally).

TempKey is 64 bytes long. The KDF and Nonce commands are capable of writing both 32 byte halves of this register, all other commands can modify only the first (lower) 32 bytes of TempKey. Either the first 32 bytes or all 64 bytes can be valid, the device does not permit the upper 32 bytes to be valid if the lower 32 bytes are invalid.

Along with the data portion of the TempKey register is a set of flags that indicate information about the source of the data and also its validity. The `Info` command can be used to return the value of some of the status/flag bits corresponding to this register as below:

Table 3-1. TempKey Flags

Name	Length	Description
KeyID	4 bits	If TempKey was generated by `GenDig` or `GenKey`, these bits indicate which key was used in its computation. The four bits represent one of the slots of the Data zone.
SourceFlag	1 bit	The source of the randomness in TempKey: `0` = Internally generated random number (*Rand*). `1` = Input (fixed) data only, no internal random generation (*Input*).
Generator	4 bit	`0` = TempKey was not generated by `GenDig`. `1` = The contents of TempKey were generated by `GenDig` using one of the slots in the Data zone (and TempKey.KeyID will be meaningful).
GenKeyData	1 bit	`0` = TempKey.KeyID was *not* generated by `GenKey`. `1` = The contents of TempKey were generated by `GenKey` using one of the slots in the Data zone (and TempKey.KeyID will be meaningful).
NoMacFlag	1 bit	`1` = The contents of TempKey were generated using the value in a slot for which slotConfig.NoMac is one, and therefore cannot be used by the MAC command. If multiple slots were used in the calculation of TempKey, then this bit will be set if slotConfig.NoMac was set for any of those slots.
Valid	1 bit	`0` = The information in TempKey is invalid. `1` = The information in TempKey is valid.

In this specification, these flags are generally referred to as TempKey.SourceFlag, TempKey.GenDigData, and so forth. When TempKey.Valid is 0, then any attempted use of the TempKey register contents will result in an error, regardless of the state of any other flag bits.

The TempKey resigster and all its flags are cleared to zero during power-up, sleep, brown-out, watchdog expiration or tamper detection. The contents of TempKey and the flags are retained when the device enters idle mode.

In general, TempKey.Valid and all the other flags are cleared to zero whenever TempKey is used (read) for any purpose during command execution. When a command that is intended to use TempKey encounters an error, TempKey may or may not be cleared depending on the situation. If a particular command or command mode/configuration does not use TempKey then it will never be cleared. TempKey is never cleared by the KDF or AES commands.

Commands which leave a result in TempKey will set the Valid flag and any other flags which may be appropriate for the operation performed.

3.2 Message Digest Buffer

The Message Digest Buffer is a 64 byte register that is used to convey the input message digest to the Verify and Sign commands when the TempKey register is needed to retain different information. The SHA command can write a digest directly to this register to simplify external host programming.

If a validating MAC is desired with the output of the Verify command, this register is always used to convey the nonce used to compute that MAC. The location of the nonce within the Message Digest buffer depends on whether the signature message digest is being input via TempKey or the Message Digest Buffer.

The Nonce command can write either 32 or 64 bytes of fixed input data to the Message Digest Buffer.

The Message Digest Buffer is generally cleared after the execution of most commands with the exception of the Nonce and SHA commands. It can only be used (read) in a single command without reloading as it is always cleared upon use.

3.3 Alternate Key Buffer

The Alternate Key Buffer is a 32 byte register that can be used by the KDF command to store keys when the TempKey register is needed to retain different information. It can be written to a fixed input value by the Nonce command or to a secret value by the KDF command.

The Alternate Key Buffer is cleared on power-up or wake from sleep and otherwise remains valid once written.

A use for the Alternate Key Buffer is to generate two separate SRAM-based keys from a single root key. One method to accomplish this is to use the KDF command with the input set to the AltKeyBuf and the output set to TempKey(Lo). Then the KDF is run a second time with the output set to TempKey(Hi), resulting in two distinct keys being stored in one location, in this case TempKey. A flow similar to this may be required for TLS 1.3.

3.4 SHA Context Buffer

The SHA command uses a standard three phase flow: Initialize, Update and Finalize. In many situations the Update phase is run many times. Internal SRAM memory is used to store the intermediate state, aka SHA context, between these phases.

This SHA context buffer is neither read nor written by any other ATECC608A command and is therefore not disrupted regardless of the success or failure of the execution of any other commands. Like all SRAM memory in the device, it is cleared when the device goes to sleep or power is removed.

3.5 Persistent Latch

The ATECC608A has a single volatile memory bit known as the Persistent Latch that retains its state as long as Vcc remains above 2.0V. It is always set to zero on power-up and may be manipulated using the operations described below. The current state of the Persistent Latch can always be read via the Info(Mode=4) command, regardless of the way in which its use is configured.

Note that this bit is *not* stored in the SRAM which loses its state when the device goes into sleep mode.

Depending on the way the configuration zone is coded, this latch may have several different functions:

Volatile Key Usage Permission: When configured in this mode, the state of the Persistent Latch is used to enable or disable key slots that have been so configured. The state of the latch can be written to either a 1 or 0 using the Info(Mode=4) command. See Section 2.2.5 for more details.

Secure Boot: This mode is similar to the previous, in that key slots are enabled or disabled with the state of the persistent latch, but the latch is set by the SecureBoot command instead of the Info command.

Authorization Output: When configured in this mode, the Persistent Latch is used to store the value to which the SCL is being driven. The state of the latch can be changed using the Info(Mode=3) command with proper authorization via config.GPIOcontrol.SignalKey. One use for this function is to securely enable or disable external hardware based on some sequence that would run in the host on startup.

Intrusion Detection: When configured in this mode, the state of the Persistent Latch reflects whether the SCL pin has been pulled low at any time during this power cycle. One use for this is to connect to an external tamper switch that detects unauthorized opening of the system case, maintenance door, etc. A typical implementation would connect a battery to VCC and SCL through the switch such that SCL goes to 0V when the switch is open or the tamper wires are broken. The state of the latch can be initially ‘armed’ via the Info(Mode=3) command with proper authorization via config.GPIOcontrol.SignalKey. Keys would be enabled/disabled with this state.

Generally, if the device is configured for more than one of these uses then the behavior may be different than expected. Both Volatile Key Usage Permission and Secure boot modes may be configured to affect the Persistent Latch, however enablement of the keys will no longer be dependent only on a proper secure boot. A careful security analysis of the system must be undertaken in this situation including protection of the VolatilePermission key value held in the host system.

4 Security Information

4.1 Cryptographic Standards

The ATECC608A follows various industry standards for the computation of cryptographic results. These reference documents are described in the sections below.

4.1.1 SHA-256

The ATECC608A computes the SHA-256 digest based upon the algorithm documented in the following site:

http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf

The complete SHA-256 message processed by the ATECC608A is listed in Section 10, for each of the particular commands that use the algorithm. Most standard software implementations of the algorithm automatically add the appropriate number of pad and length bits to this message to match the operation the device performs internally.

The SHA-256 algorithm is also used for encryption by taking the output digest of the hash algorithm and XORing it with the plain text data to produce the ciphertext. Decryption is the reverse operation, in which the ciphertext is XORed with the digest with the result being the plain text.

4.1.2 HMAC/SHA-256

The ATECC608A can compute an HMAC digest based upon SHA-256 using a key stored in the EEPROM as documented below:

http://csrc.nist.gov/publications/fips/fips198-1/FIPS-198-1_final.pdf

4.1.3 TLS V1.2 Pseudo Random Function (PRF)

The ATECC608A `KDF(PRF)` command calculates the key derivation function (KDF) specified for TLSV1.2 (and earlier) at:

https://tools.ietf.org/html/rfc5246

4.1.4 HMAC-based Extract-and-Expand KDF (HKDF)

The ATECC608A `KDF(HKDF)` command calculates the HKDF key derivation function (KDF). This is the KDF currently planned to be used in the TLS1.3 proposal. It is specified at:

https://tools.ietf.org/html/rfc5869

4.1.5 Elliptic Curve Digital Signature Algorithm (ECDSA)

The ATECC608A computes and verifies the Elliptic Curve signatures according to the algorithm documented in:

ANSI X9.62-2005	http://www.ansi.org/
FIPS 186-4 specification	http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf

4.1.6 Elliptic Curve Diffie-Hellman (ECDH)

The ATECC608A executes the ECDH key agreement according to NIST Special Publication 800-56A recommendations:

http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-56Ar2.pdf

For the purposes of compliance to 800-56Ar2, the ATECC608A always treats the input public key as an ephemeral key. For optimal security, no private key should be used for both ECDSA and ECDH. This restriction can be implemented by properly setting the bits within slotConfig[keyId].readKey.

4.1.7 Advanced Encryption Standard (AES)

Symmetric encryption & decryption, if enabled, is implemented via the AES command using only AES-128 per:

http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf

4.2 Secure Boot

The ATECC608A provides a mechanism to support secure boot operations in a connected MCU, which can help to identify situations in which fraudulent code has been installed on the host CPU. On power-up, the boot code within the host MCU sends the code digest and appropriate signature to the ATECC608A. If the signature validates that digest using the pre-approved code validating public key stored in the ATECC608A, then a message is returned to the host MCU indicating success.

Identity and communication keys can optionally be configured to be disabled on power-up. In the event of a secure boot successful validation, these keys will be enabled for use. The state of the secure boot result is stored in a special latch that does not lose its value even when the ATECC608A is put to sleep so that the secure boot operation does not need to be re-run unless power is removed from the ATECC608A. It is expected that the same power rail will be used to supply both the host MCU and the ATECC608A.

If the secure boot validation fails, then the ID/comm keys will remain disabled until a successful digest/signature pair is sent to the ATECC608A.

The public key used to validate the signature of the code digest is stored in the slot indicated by config.SecureBoot.SecureBootPubKey. Generally, slotConfig.writeConfig for this slot should be set to Never, as the code signing key is rarely intended to change. If it is desired that the system have the ability to update the Secure Boot public key, then writeConfig for that slot should be set to PubInvalid (0001) and the public key update mechanism described in Section 4.3 should be employed.

If the code to be validated at boot is relatively small, then the SHA computation engine on the ATECC608A can be used to calculate the code digest by sending the code bytes to the ATECC608A. For many systems this may be too slow, however in some hierarchical boot cases the primary boot code itself can be optimized sufficiently that its size permits use of the ATECC608A.

4.2.1 Secure boot speed optimization

The ATECC608A SecureBoot command includes the option to store the signature and/or digest within the protected boundary of the ATECC608A to reduce the execution time. The signature and/or digest can be updated via a mode switch on the normal secure boot command which performs both a verify of the signature and storage of the signature/digest in a designated slot.

Storing the signature reduces the boot time by limiting the size of the IO block that needs to be sent to the ATECC608A from 96 bytes to only 32 bytes.

If the digest is stored, then the ATECC608A does only a digest compare between the host code digest in the input array and the stored digest in a slot of the ATECC608A. This reduces the boot time by eliminating the computation delay for the ECC verification.

The SecureBootSigDig slot configuration must be set up to ensure that only the SecureBoot command has the capability to write into the designated storage slot:

- The target slot (config.SecureBoot.SigDig) must not be locked via the slotLocked capability
- Write config for the target slot (config.SlotConfig.WriteConfig) should generally be set to "Never" to avoud situations in which fraudulent modification of the slot may be possible
- If the device is configured to store the signature then the designated slot must be from 8-15. Digests can be stored in any slot

4.2.2 Secure boot wire protection

In some applications, it may be necessary to protect the system against an adversary who might cut the wire(s) between the ATECC608A and the host MCU to replace the result of the Verify operation with a fraudulent "success" signal.

If so configured or indicated by the mode parameter to the Secure Boot command, the input code digest can be encrypted via and XOR of the code digest with a digest of a nonce and the IO protection secret.

The config.SecureBoot.SecureBootRandNonce bit can be used to ensure that this input encryption is always performed and that it uses a nonce generated by the ATECC608A random number. Such a flow prevents the replay of a message from an earlier point in time from being used as input to the ATECC608A SecureBoot command.

If so configured or indicated by the mode parameter to the Secure Boot command, the output Boolean can be accompanied by a MAC generated over an input nonce from the host and the IO protection key. Generally, the host nonce would be generated by a counter or random number generator on the MCU, its only requirement is that it be unique vs other secure boot nonces.

These two functions: MAC-protected enablement of specified key slots and MAC validation of the ECC Verify output are available separately in the Volatile Key Permit function (See Section 2.2.5) and the Verify command MAC output if the Secure Boot command is not used.

In addition to prevention of man-in-the-middle attacks, these sequences prevent the ATECC608A from being removed off of one board and used on another or in some other kind of system since the new board would not contain the IO protection secret contained on the MCU of the initial board. While it is expected that with some effort an attacker could extract the IO protection secret from the MCU, that process would have to be repeated again for every board attacked.

4.3 Public Key Update

Keys used to validate code digest or chain of trust roots can be stored on the ATECC608A. Using the device for this storage can prevent some kinds of attacks because rogue software cannot overwrite the Public key with a fraudulent key.

Generally, these root public keys are stored in a slot and the slotConfig.writeConfig field is set to Never. In this case, there is no method for any software to write the public key slot. However, in some environments there may be a requirement to be able to update the root key in the hopefully unlikely event that the corresponding private key is compromised.

The basic flow for public key update is as follows: 1) Use the Parent key to sign the appropriate message to Verify(Invalidate) to clear the validation bits on the root public key. 2) Use the Write command to write a new root public key to the root public key slot, it will be still marked as invalid and cannot yet be used. 3) Use the Parent key to sign the appropriate message to Verify(Validate) to set the validation bits on the root public key and allow for normal use.

The two key slots should be configured as follows:

- The root public key should be stored in a slot with slotConfig.writeConfig set to “PubInvalid” (0001). KeyConfig.KeyType should be set to P256 (4). KeyConfig.PubInfo should be set to 1. KeyConfig.X509id should point to one of the config.X508format bytes that has a value of 0. SlotConfig.ReadKey should point to a parent public key.
- The Parent Public key slot should have slotConfig.writeConfig set to Never, KeyConfig.KeyType should be set to P256 (4) and KeyConfig.PubInfo should be set to 0.

If keyConfig.reqRandom is set on the parent slot, then the update sequence will be required to use the RNG on the ATECC608A, thereby preventing an update package that works on one device from working on another. If reqRandom is zero, then a common update sequence can be used for multiple ATECC608A devices.

This flow is protected against a destructive rogue write (denial of service) because in order to do the write a signature is required from the parent key. Under normal circumstances, such a signature would never exist and the write would never be possible.

4.4 Key Uses and Restrictions

Any slot in the EEPROM Data zone can be used to store a secret or private key. There are a number of ways in which the keys stored within the device can be used and/or their access restricted. See the following sections 4.4.1, Diversified Keys to 4.4.8, Diversified Keys for some of these concepts.

The device should be properly configured to prevent any unwanted read and write access to all key slots, including the setting of the IsSecret bit. Private keys can never be read from the device regardless of the values in the Configuration zone.

With the exception of transport keys documented in Section 4.4.7, Transport Keys, the most significant 12 bits of all KeyID parameters should be zero.

4.4.1 Diversified Keys

If the host or validating entity has a place to securely store secrets, or contains an ATECC608A device, the secret key values stored in the EEPROM slot(s) of the clients can be diversified by using the serial number embedded in the device (SN[0:8]). In this manner, every client device can have a unique key, which can provide extra protection against known plaintext attacks and permit compromised serial numbers to be identified and blacklisted.

To implement this operation, a root secret is externally combined with the device’s serial number during personalization by using some cryptographic algorithm, and the result is written to the ATECC608A key slot.

The ATECC608A `GenDig` and `CheckMac` commands provide a mechanism to securely generate and compare diversified keys, thereby eliminating this requirement from the host system.

Consult the following application note for more details:

http://www.atmel.com/dyn/resources/prod_documents/doc8666.pdf

4.4.2 Rolled Keys

In order to prevent repeated uses of the same secret key value, the ATECC608A supports key rolling. Normally, after a certain number of uses (perhaps as few as one), the current key value is replaced with the SHA-256 digest of its current value combined with some offset, which may either be a constant, something related to the current system (for example, a serial number or model number), or a random number.

This capability is implemented using the `DeriveKey` command. Prior to execution of the `DeriveKey` command, the `Nonce` command must be run to load the offset into TempKey.

One use for this capability is to permanently remove the original key from the device, and replace it with a key that is only useful in a particular environment. After the key is rolled, there is no possible way to retrieve the old key’s value, which improves the security of the system.

Any power interruption during the execution of the `DeriveKey` command in Roll mode may cause the key to have an unknown value. If writing to a slot is enabled using bit 14 of SlotConfig, such keys can be written in encrypted and authenticated form using the `Write` command. Alternatively, multiple copies of the key can be stored in multiple slots so that failure of a single slot does not incapacitate the system.

4.4.3 Created ECC Keys

For the highest security, private ECC keys may be created within the ATECC608A using the internal high quality RNG. When a key is generated internally, the public portion of the key is always returned to the system. The generated key may be either stored in an EEPROM slot or held in TempKey for later use by the ECDH command.

For private keys stored in an EEPROM slot, the public key can optionally be computed from the private key if the slot is so configured.

Internally generated keys are guaranteed to be unique to this device since there is no mechanism for reading the value of an ECC private key from the ATECC608A.

4.4.4 Created Secret Keys

There may be a need to have unique ephemeral symmetric keys on each client; a function also supported by the ATECC608A. With this mechanism, a parent key (that is specified by slotConfig.writeKey) is combined with a fixed or random nonce to create a unique key, which is then used for any cryptographic purpose.

The ability to create unique keys is especially useful if the parent key has usage restrictions (see Section 4.4.5, below). In this mode, the limited use parent can be employed to create an unlimited use child key. Because the child key is useful only for this particular host-client pair, attacks on its value are less valuable.

This capability is also implemented using the `DeriveKey` command. Prior to execution of the `DeriveKey` command, the `Nonce` command must be run to load the nonce value into TempKey.

4.4.5 High Endurance Monotonic Counters

The ATECC608A supports two independent high endurance nonvolatile monotonic counters that can count to a value of 2,097,151. Their value never decreases and the storage elements are protected against count loss if the power is interrupted during an incrementing operation. The current value of the two counters can be read using the `Counter` command, which can also be used to increment the counters. There is no way to reset the counters.

The counters can be used in one of two methods:

- **Cryptographic Counters**:
  In this mode, the `Counter` command is used to increment the value of the counters and the current value can be read via the same command. The two counters are independent.
- **Limited Key Use**:
  Counter[0] can be attached to any key via the slotConfig.limitedUse bit. If this bit is set for any particular key, then any use of the key will cause the counter to increment automatically prior to the operation being performed. If the counter has reached its limit, then the command will return an error code, and no counter change will occur. See Microchip for alternate limit programming.

The `GenKey`, `Read`, and `Write` commands ignore the monotonic counter limited use feature. It is also ignored for the copied slot during `CheckMac(Copy)`.

4.4.6 Password Checking

Many applications require a user to enter a password to enable features, decrypt stored data, or perform some other task. Typically, the expected password has to be stored somewhere in the memory, and therefore is subject to discovery. The ATECC608A can securely store the expected password and perform a number of useful operations upon it. The password is never passed in the clear to the device, and it cannot be read from the device. It is hashed with a random number in the system software before being passed to the device.

The copy capability of the `CheckMac` command enables the following types of password checking options:

1. `CheckMac` does an internal comparison with the expected password and returns a Boolean result to the system to indicate whether the password was correctly entered or not.
2. If the device determines that the correct password has been entered, then the value of the password can optionally be combined with a stored ephemeral value to create a key that can be used by the system for data protection purposes.
3. If the device determines that the correct password has been entered, then the device can use this fact to optionally release a secondary high entropy secret, which can be used for data protection without the risk of an exhaustive dictionary attack.
4. If the password has been lost, then an entity with knowledge of a parent key value can optionally write a new password into the slot. Optionally, the current value can be encrypted with a parent key and read from the device.

To prepare for this `CheckMac/Copy` capability, passwords should be stored in even numbered slots. If the password is to be mapped to a secondary value (using the third option above), then the target slot containing this value is located in the next higher slot number (i.e. the password's slot number plus one); otherwise, the target slot is the same as the password slot. ReadKey for the target slot must be set to zero to enable this capability. In order to prevent fraudulent or unintended usage of this capability, do not set ReadKey for any slot to zero unless this `CheckMac/Copy` capability is specifically required. In particular, do not assume that the other bits in the configuration word for a particular slot will override the enablement of this capability specified by ReadKey = 0.

This capability is only enabled if the mode parameter to `CheckMac` has a value of `0x01`, indicating the following:

- The first 32 bytes of the SHA-256 message are stored in a data slot in the EEPROM (i.e. the password).
- The second 32 bytes of the SHA-256 message must be a randomly generated `Nonce` in the TempKey register.

If the above conditions are met, and the input response matches the internally generated digest, then the contents of the target key are copied to TempKey. The other TempKey register bits are set as follows:

- SourceFlag is set to one (not Random).
- GenDigData is set to zero (not generate by the `GenDigData` command).
- NoMacFlag is set to zero (TempKey is usable by `MAC` and `Read` commands).
- Valid is set to one.

See the Microchip website for application notes with more details on this capability.

4.4.7 Transport Keys

The ATECC608A device includes an internal hardware array of keys that are used for secure personalization (i.e. transport keys). The values of the hardware keys are kept secret and are made available only to qualified customers upon request to Microchip. These keys can be used with the `GenDig` command *only* and are indicated by a KeyID value greater than or equal to `0x8000`.

For `GenDig` and all other commands, KeyID values of less than `0x8000` always reference keys that are stored in the Data zone of the EEPROM. In these cases, only the four least-significant bits of KeyID are used to determine the slot number, while the entire 16 bit KeyID as input is used in any SHA-256 message calculation.

4.4.8 Authorized Keys

The ATECC608A device provides an optional mechanism for restricting the use of any key to those users with knowledge of the appropriate authorization information.

Key authorization is a standard cryptographic requirement in many systems and can be used to prevent fraudulent use of a key if the device containing the key is stolen or lost. For instance, if a key is used as identification for a person, the authorizing value could be a password known only to that person. If the device with the ID is stolen, then the thief cannot use the device to sign fraudulent messages since he or she does not know the password.

The device can use either the `CheckMac` or `Verify` commands to implement this capability. If the validation succeeds, then an internal AuthComplete flag is set and the authorizing slot number is retained. The AuthComplete flag is cleared whenever the device wakes from sleep or is powered on. It is also cleared when any operation is performed on a key which requires authorization. Prior to the authorization check, the `Nonce` command must be run to load TempKey with a nonce.

- `CheckMac`
  The authorization value is stored in any slot configured to contain a secret, and it is validated with a MAC calculated using that secret and the nonce stored in TempKey.
- `Verify`
  The authorizing slot must contain a valid ECC public key. The authorization value should be a signature calculated using the corresponding private key calculated over the nonce stored in TempKey. This signature is then validated.

Depending upon the configuration of the slot containing the authorizing secret, a token can be externally stored, which can be repeatedly used for key authorization. If the authorizing slot is configured to require a random nonce (KeyConfig.ReqRandom is one), then a stored authorizing token will not work, and the authorizing digest or signature will have to be computed on the fly by the authorizing agent using the random nonce generated by the device.

4.5 Security Features

4.5.1 Physical Security

The ATECC608A incorporates a number of physical security features designed to protect the EEPROM contents from unauthorized exposure. Among many others, these security measures include:

- Active Shield Circuitry
- Internal Memory Encryption
- Glitch Protection
- Voltage and Temperature Tamper Detection

Pre-programmed transport keys stored on the ATECC608A are encrypted in such a way as to make retrieval of their values using outside analysis very difficult.

Both the logic clock and logic supply voltage are internally generated, thus preventing any direct attack on these two signals using the pins of the device.

4.5.2 Random Number Generator (RNG)

The ATECC608A includes a high quality cryptographic random number generator implemented using a combination of a non-deterministic noise (entropy) source (NRBG) seeding a deterministic algorithm (DRBG) implemented according to the following NIST standards. The NRBG is used both in the instantiation and each time an RNG number is required.

The NRBG output is evaluated using the methods in NIST SP 800-90B. The DRBG is designed using the AES-128 variant specified within NIST SP 800-90A. The combination of the two to create the final random number generator follows the methods specified in NIST SP 800-90C.

http://csrc.nist.gov/groups/STM/cavp/documents/drbg/DRBGVS.pdf (SP 800-90A)

http://csrc.nist.gov/publications/drafts/800-90/draft-sp800-90b.pdf

http://csrc.nist.gov/publications/drafts/800-90/sp800_90c_second_draft.pdf

The noise source runs continuously when power is applied to the chip unless the chip enters sleep mode. When a random number is requested the following two conditions are met prior to use of the random number:

- Health testing is applied to the output of the noise source per SP 800-90B to ensure that this particular noise source is working properly at the current time under the current environmental conditions.
- If insufficient entropy has been stored within ATECC608A, it will delay for sufficient time to accumulate the required entropy.

See http://csrc.nist.gov/groups/STM/cavp/documents/drbg/drbgval.html for further documentation on NIST CAVP certification of this RNG. Note that as of the time of this document, only SP-800-90A has been released, both 800-90B and 800-90C are in various draft stages.

The system may use this RNG for any purpose. The device provides a special `Random` command for such purposes that does not affect any ephemeral nonce that may currently be stored in one of the SRAM registers.

To simplify system testing, prior to Config Locking the RNG always returns the following value:

ff ff 00 00 ff ff 00 00 …

where ff is the first byte read from the device and the first byte into the SHA message.

5 General I/O Information

Communications to the ATECC608A are through one of two different protocols. The protocols are selected by specifying the part number that is ordered:

- **Single-Wire Interface**
  Uses a single GPIO connection on the system microprocessor that is connected to the SDA pin on the device. It permits the fewest number of pins connected to any removable or replaceable entity. The bit rate is up to 26Kb/s.
- **I²C Interface**
  This mode is compatible with the I²C standard and also with the Microchip AT24C16 Serial EEPROM interface. Two pins, Serial Data (SDA) and Serial Clock (SCL), are required. The I²C interface supports a bit rate of up to 1Mb/s.

  The ATECC608A and AT24C16B have different default I²C addresses. The ATECC608A I²C address can be modified from default by writing a new value into the Configuration zone.

The device implements a failsafe internal watchdog timer that forces it into a very low-power mode after a certain time interval regardless of any current activity. System programming must take this into consideration. See Section 10.6, Watchdog Failsafe.

5.1 Byte and Bit Ordering

CryptoAuthentication uses a common ordering scheme for bytes and also for the way in which numbers and arrays are represented in this datasheet:

- All multi-byte aggregate elements are treated as arrays of bytes and are processed in the order received or transmitted with index #0 first.
- 16 bit (2 byte) integers, typically Param2, SlotConfig or KeyConfig, appear on the bus least-significant byte first.
- ECC keys appear on the bus, and are stored in EEPROM, with the most significant 32-bit word at the lowest address. See Section 5.1.1, ECC Key Formatting for further information on ECC key formatting.

In this document, the most-significant bit or nibble of a byte or 16 bit word appears towards the left hand side of the page.

The bit order is different depending upon the I/O channel used:

- On the one-wire bus, data is transferred to and from the ATECC608A Least Significant bit (LSb) first on the bus.
- On the I²C interface, data is transferred to and from the ATECC608A Most Significant bit (MSb) first on the bus.

5.1.1 ECC Key Formatting

The format for public and private keys depends on the command and key length. In general, the Most Significant bytes (MSB) appear first on the bus and at the lowest address in memory. In the remainder of this section below, the bytes on the left side of the page are the MSBs. Atmel recommends all pad bytes be set to zero for consistency.

- ECC private keys appear to the user only as the input parameter to the `PrivWrite` command. This parameter is always 36 bytes in length and the first four bytes (32 bits) are all pad bits.

ECC public keys appear as the input or output parameters to several commands, and they can also be stored in EEPROM. They are composed of an *X* value first on the bus or in memory, followed by a *Y* value. They are formatted differently depending upon the situation as noted below:

- **The public key is an output of `GenKey` command or an input to `Verify` command**:
  32 bytes of X, then 32 bytes of Y. (36 bytes) There are no pad bytes.
- `Write` **Command**:
  Public keys can be written directly to the EEPROM using `Write` command and are always 72 bytes long, formatted as follows: 4 pad bytes, 32 bytes of X, four pad bytes, then 32 bytes of Y.
- `GenKey` **Command**:
  SHA Message: Public keys can be hashed and placed in TempKey by the `GenKey` command. The SHA message contains various bytes that are independent of the size of the key. These are followed by 25 bytes of pad, followed by 32 bytes of X, then 32 bytes of Y.
- `Verify` **Command**:
  SHA Message: When used to validate a stored public key, the `Verify` command expects an input signature created over a SHA-256 digest of a key stored in memory. Such an inner SHA calculation is always performed over 72 bytes formatted as they are stored in EEPROM as 4 pad bytes, 32 bytes of X, four pad bytes, then 32 bytes of Y.

When a public key is configured to be validated by the `Verify` command, then the most significant four bits of the first byte in memory are used internally by the device to save the validation state. They are always set to the invalid state (`0xA`) by the `Write` command, and then may be set to the valid state (`0x5`) by the `Verify` command.

The lowest levels of the I/O protocols are described below. Above the I/O protocol level, exactly the same bytes are transferred to and from the device to implement the commands and error codes documented below.

6 Single-Wire Interface

In this mode, communications to and from the ATECC608A take place over SDA, a single asynchronously timed wire, and the SCL pin is not used as part of the communications channel. Instead, the SCL pin functions as a GPIO pin.

The sleep current specification values are guaranteed only if SCL pin is held low or left unconnected.

The overall communications structure is a hierarchical format:

Tokens I/O	Tokens implement a single data bit transmitted on the bus, or the wake-up event.
Flags	Flags consist of eight tokens (bits) that convey the direction and meaning of the next group of bits (if any) that may be transmitted.
Groups	Groups of data follow the command and transmit flags. They incorporate both a byte count and a checksum to ensure proper data transmission.
Packets	Packets of bytes form the core of the group (minus the byte count and CRC). They are either the input or output parameters of a CryptoAuthentication command or status information from the ATECC608A.

See the Microchip website for the appropriate application notes for more details on how to use any microprocessor to easily generate the signaling necessary to send these elements to the device, including C source code libraries. Also see Section 13.2, Wiring Configuration for Single-Wire Interface for more information about how to connect the device in the Single-Wire Interface mode.

6.1 I/O Tokens

There are a number of I/O tokens, which may be transmitted over the Single-Wire Interface:

- **Input** (to the ATECC608A):
  - Wake Wake the device up from either the sleep or idle modes, or reset the I/O interface.
  - Zero Send a single bit from the system to the device with a value of zero.
  - One Send a single bit from the system to the device with a value of one.
- **Output** (from the ATECC608A):
  - ZeroOut Send a single bit from the device to the system with a value of zero.
  - OneOut Send a single bit from the device to the system with a value of one.

The waveforms are the same in either direction, however, there are some differences in timing based upon the expectation that the host has a very accurate and consistent clock while the ATECC608A has significant part-to-part variability in its internal clock generator due to normal manufacturing and environmental fluctuations.

The bit timings are designed to permit a standard UART running at 230.4Kbaud to transmit and receive the tokens efficiently. Each byte transmitted or received by the UART corresponds to a single bit received or transmitted by the device.

The Wake token is special since it requires an extra long low pulse on the SDA pin, which cannot be confused with the shorter low pulses that occur during a data token (i.e. Zero, One, ZeroOut, or OneOut). Devices that are either in the idle or sleep mode will ignore all data tokens until they receive a legal wake token. If the processor is out of synchronization with the ATECC608A, it can send an additional Wake token to the device, which will reset the I/O channel hardware on the device.

Note Well: This may result in the loss of data stored in the command output buffer.

6.2 I/O Flags

The system is always the bus master, so before any I/O transaction, the system must send an eight bit flag to the device to indicate the I/O operation that will be subsequently performed.

Table 6-1. IO Flags

Value	Name	Meaning
`0x77`	Command	After this flag, the system starts sending a command group to the device. The first bit of the group can follow immediately after the last bit of the flag.
`0x88`	Transmit	This command tells the device to wait for a bus turnaround time and then to start transmitting its response to the previously transmitted command group.
`0xBB`	Idle	Upon receipt of an idle flag, the device goes into the idle mode and remains there until the next Wake token is received.
`0xCC`	Sleep	Upon receipt of a sleep flag, the device enters the low-power sleep mode until the next Wake token is received.
All other values are reserved and should not be used.		

- **Transmit Flag**
  The transmit flag is used to turn around the bus so that the ATECC608A can send data back to the system. The bytes that the device returns to the system depend on the current state of the device and may include status, error code, or command results.

When the device is busy executing a command, it ignores the SDA pin and any flags that are sent by the system. See Table 10-4, Command Opcodes, Short Descriptions, and Execution Time for each command type's execution delays. The system must observe these delays after sending a command to the device.

- **Idle Flag**
  The idle flag is used to transition the ATECC608A to the idle mode, which causes the input/output buffer to be flushed. It does ***not*** invalidate the contents of the TempKey, MessageDigest Buffer and Alt Key registers. This flag can be sent to the device at any time that it will accept a flag. When the device is in the idle mode, the watchdog timer is disabled.
- **Sleep Flag**
  The sleep flag transitions the ATECC608A to the low-power sleep mode, which causes a complete reset of the device, including invalidation of the contents of the SRAM and all volatile registers. This flag can be sent to the device at any time that it will accept a flag.

6.3 Synchronization

Because the communications protocol is half-duplex, there is the possibility that the system and the ATECC608A will fall out of synchronization with each other. In order to speed recovery, the device implements a timeout that forces it to sleep under certain circumstances.

6.3.1 I/O Timeout

After a leading transition for any data token has been received, the ATECC608A will expect both the completion of the token and the start of the next (if this is not the last token of the group) to be properly received by the device within the $t_{TIMEOUT}$ interval. Failure to send enough bits, or the transmission of an illegal token (e.g. a low pulse exceeding t_{ZLO}), will cause the device to enter the Sleep mode after the $t_{TIMEOUT}$ interval.

The same timeout applies during the transmission of the command group. After the transmission of a legal command flag, the I/O Timeout Circuitry is enabled until the last expected data bit is received.

The Timeout Counter is reset after every legal token; therefore, the total time to transmit the command may exceed the $t_{TIMEOUT}$ interval while the time between bits may not.

The I/O timeout circuitry is disabled when the device is busy executing a command.

6.3.2 Synchronization Procedures

If the device is not busy when the system sends a transmit flag, the device should respond within $t_{TURNAROUND}$. If t_{EXEC} time has not already passed, the device may be busy, and the system should poll or wait until the maximum t_{EXEC} time has elapsed. If the device still does not respond to a second transmit flag within $t_{TURNAROUND}$, it may be out of synchronization. At this point, the system may take the following steps to reestablish communication:

1. Wait $t_{TIMEOUT}$.
2. Send the transmit flag.
3. If the device responds within $t_{TURNAROUND}$, then the system may proceed with more commands.
4. Send a wake token.
5. Wait t_{WHI}.
6. Send the transmit flag.
7. The device should respond with a `0x11` return status within $t_{TURNAROUND}$, after which the system may proceed with more commands.

7 I^2C Interface

The I^2C Interface uses the SDA and SCL pins to indicate various I/O states to the ATECC608A. This interface is designed to be compatible at the protocol level with the Microchip AT24C16 Serial EEPROM operating at 1MHz.

Note Well: There are many differences between the two devices (for example, the ATECC608A and AT24C16 have different default I^2C addresses); therefore, designers should read the respective datasheets carefully.

The SDA pin is normally pulled high with an external pull-up resistor because the ATECC608A includes only an open-drain driver on its output pin. The bus master may either be open-drain or totem pole. In the latter case, it should be tri-stated when the ATECC608A is driving results on the bus. The SCL pin is an input and must be driven both high and low at all times by an external device or resistor.

7.1 I/O Conditions

The device responds to the following I/O conditions:

7.1.1 Device is Asleep

When the device is asleep, it ignores all but the Wake condition.

- **Wake —** If SDA is held low for a period of greater than t_{WLO}, the device will exit low-power mode and after a delay of t_{WHI}, it will be ready to receive I^2C commands. The device ignores any levels or transitions on the SCL pin when the device is idle or asleep and during t_{WLO}. At some point during t_{WHI} the SCL pin is enabled and the conditions listed in Section 7.1.2, Device is Awake are honored.

The Wake condition requires that either the system processor manually drives the SDA pin low for t_{WLO}, or a data byte of `0x00` be transmitted at a clock rate sufficiently slow so that SDA is low for a minimum period of t_{WLO}. When the device is awake, the normal processor I^2C hardware and/or software can be used for device communications up to and including the I/O sequence required, thus putting the device back into low-power (i.e. sleep) mode.

When there are multiple ATECC608A devices on the bus, and the I^2C interface is run at 133KHz or slower, the transmission of certain data patterns (such as `0x00`) will cause all the ATECC608A devices on the bus to wake-up. Because subsequent device addresses transmitted along the bus will only match the desired devices, the unused devices will remain idle and not cause any bus conflicts.

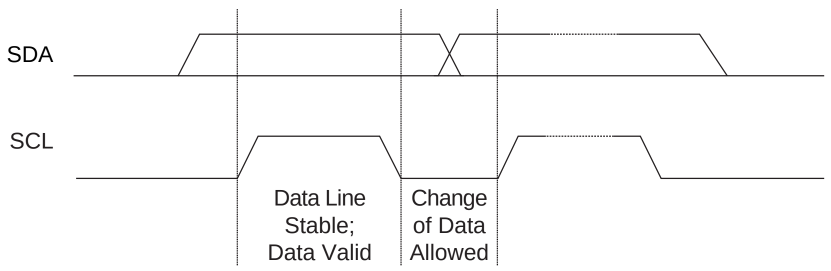
In the I^2C mode, the device will ignore a wake sequence that is sent when the device is already awake.

7.1.2 Device is Awake

When the device is awake, it honors the conditions listed below:

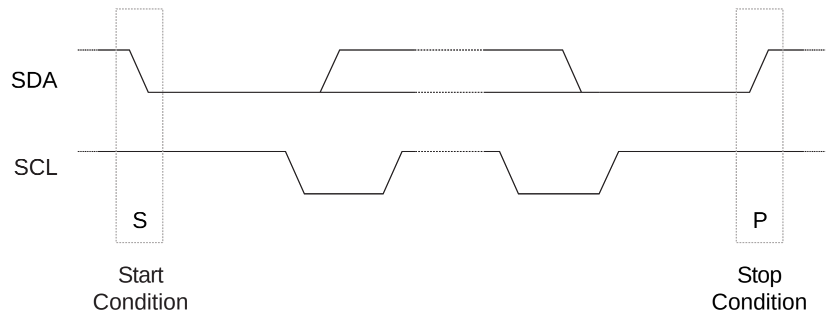
- **DATA Zero:** If SDA is low and stable while SCL goes from low to high to low, then a zero bit is being transferred on the bus. SDA can change while SCL is low.
- **DATA One:** If SDA is high and stable while SCL goes from low to high to low, then a one bit is being transferred on the bus. SDA can change while SCL is low.

Figure 7-1. Data Bit Transfer on I²C Interface



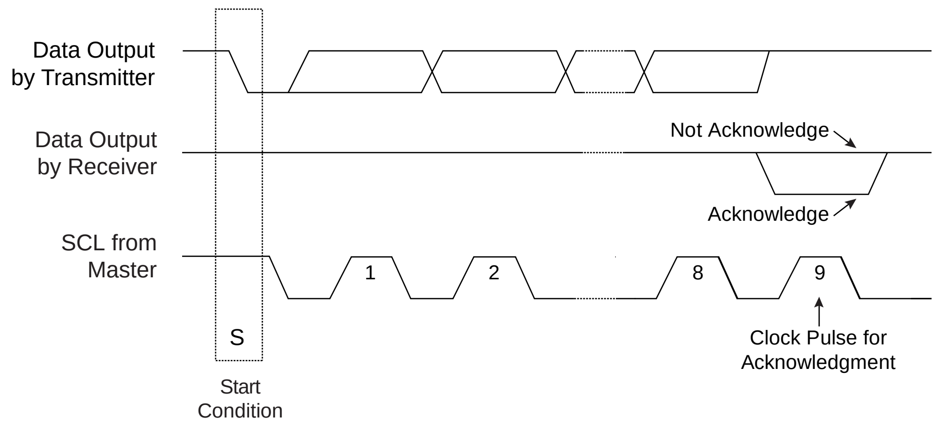
- **Start Condition:** A high-to-low transition of SDA with SCL high is a Start condition which must precede all commands.
- **Stop Condition:** A low-to-high transition of SDA with SCL high is a Stop condition. After this condition is received by the device, the current I/O transaction ends. On input, if the device has sufficient bytes to execute a command, the device transitions to the busy state and begins execution. The Stop condition should always be sent at the end of any packet sent to the device.

Figure 7-2. Start and Stop Conditions on I²C Interface



- **Acknowledge (ACK):** On the ninth clock cycle after every address or data byte is transferred, the receiver will pull the SDA pin low to acknowledge proper reception of the byte.
- **Not Acknowledge (NOT ACK):** Alternatively, on the ninth clock cycle after every address or data byte is transferred, the receiver can leave the SDA pin high to indicate that there was a problem with the reception of the byte or that this byte completes the group transfer.

Figure 7-3. NOT ACK and ACK Conditions on I²C Interface



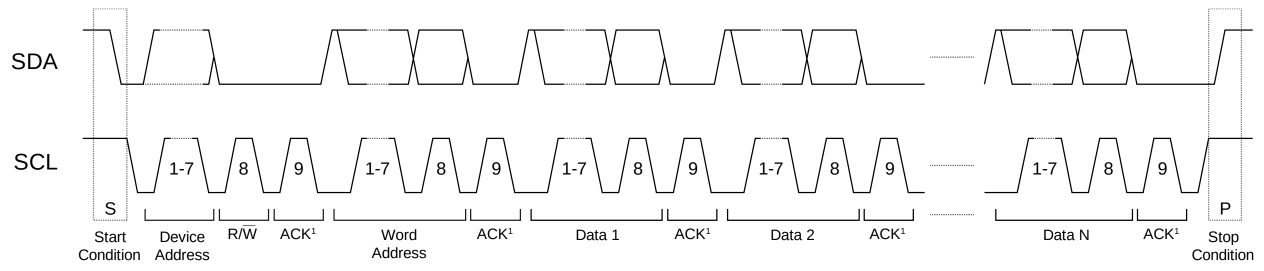
Multiple ATECC608A devices can easily share the same I^{2}C interface signals if the I2C_Address byte in the Configuration zone is programmed differently for each device on the bus. Because all seven of the bits of the device address are programmable, ATECC608A can also share the I^{2}C interface with any I^{2}C device, including any Serial EEPROM.

7.2 I^{2}C Transmission to ATECC608A

The transmission of data from the system to the ATECC608A is summarized in the table below. The order of transmission is as follows:

- Start Condition
- Device Address Byte
- Word Address Byte
- Optional Data Bytes (1 through N)
- Stop Condition

Figure 7-4. Normal I^{2}C Transmission to ATECC608A



SDA is driven low by ATECC608A ACK periods.

The tables below label the bytes of the I/O transaction. The column labeled "I^{2}C Name" provides the name of the byte as described in the AT24C16 datasheet.

Table 7-1. I^{2}C Transmission to ATECC608A

Name	I^{2}C Name	Description
Device Address	Device Address	This byte selects a particular device on the I^{2}C interface. ATECC608A is selected if bits 1 thru 7 of this byte match bits 1 thru 7 of the I2C_Address byte in the Configuration zone. Bit 0 of this byte is the standard I^{2}C R/W bit, and should be zero to indicate a write operation (the bytes following the device address travel from the master to the slave).
Word Address	Word Address	This byte should have a value of `0x03` for normal operation. See Sections 7.2.1, Word Address Values and 7.6, Address Counter for more information.
Command	Data1,N	The command group, consisting of the count, command packet, and the two byte CRC. The CRC is calculated over the size and packet bytes. See Section 10.1, I/O Groups.

Because the device treats the command input buffer as a FIFO, the input group can be sent to the device in one or many I^{2}C command groups. The first byte sent to the device is the count, so after the device receives that number of bytes, it will ignore any subsequently received bytes until execution is finished.

The system ***must*** send a Stop condition after the last command byte to ensure that ATECC608A will start the computation of the command. Failure to send a Stop condition may eventually result in a loss of synchronization; see Section 7.8, I2C Synchronization for recovery procedures.

7.2.1 Word Address Values

During an I^{2}C write packet, the ATECC608A interprets the second byte sent as the word address, which indicates the packet function as it is described in the table below:

Table 7-2. Word Address Values

Name	Value	Description
Reset	`0x00`	Reset the address counter. The next I^{2}C read or write transaction will start with the beginning of the I/O buffer.
Sleep (Low-power)	`0x01`	The ATECC608A goes into the low power sleep mode and ignores all subsequent I/O transitions until the next wake flag. The entire volatile state of the device is reset.
Idle	`0x02`	The ATECC608A goes into the idle mode and ignores all subsequent I/O transitions until the next wake flag. The contents of TempKey, MessageDigestBuffer, and Alternate Key registers are retained.
Command	`0x03`	Write subsequent bytes to sequential addresses in the input command buffer that follow previous writes. This is the normal operation.
Reserved	`0x04 – 0xFF`	These addresses should not be sent to the device.

7.2.2 Command Completion Polling

After a complete command has been sent to the ATECC608A, the device will be busy until the command computation completes. The system has two options for this delay as noted below:

- **Polling:**
  The system should wait t_{EXEC} (typical) and then send a read sequence (see Section 7.5, I2C Transmission from the ATECC608A). If the device NOT ACKs the device address, then it is still busy. The system may delay for some time or immediately send another read sequence, again looping on NOT ACK. After a total delay of t_{EXEC} (max), the device will have completed the computation and return the results.
- **Single Delay:**
  The system should wait t_{EXEC} (max) after which the device will have completed execution, and the result can be read from the device using a normal read sequence.

7.3 Sleep Sequence

Upon completion of the use of the ATECC608A by the system, the system should issue a sleep sequence to put the device into low-power mode. This sequence consists of the proper device address followed by the value of `0x01` as the word address followed by a Stop condition. This transition to the low-power state causes a complete reset of the device's internal command engine and input/output buffer. It can be sent to the device at any time when it is awake and not busy.

7.4 Idle Sequence

If the total sequence of required commands exceeds $t_{WATCHDOG}$, then the device will automatically go to sleep and lose any information stored in the volatile registers. This action can be prevented by putting the device into the idle mode prior to completion of the watchdog interval. When the device receives the Wake token, it will then restart the watchdog timer and execution can be continued.

The idle sequence consists of the proper device address followed by the value of `0x02` as the word address followed by a Stop condition. It can be sent to the device at any time when it is awake and not busy.

7.5 I²C Transmission from the ATECC608A

When the ATECC608A is awake and not busy, the bus master can retrieve the current buffer contents from the device using an I²C Read. If valid command results are available, the size of the group returned is determined by the particular command which has been run. Otherwise, the size of the group (and the first byte returned) will always be four: count, status/error, and 2-byte CRC. The bus timing is shown in Figure 9-4, I2C Synchronous Data Timing.

Table 7-3. I²C Transmission from the ATECC608A

Name	I²C Name	Direction	Description
Device Address	Device Address	To slave	This byte selects a particular device on the I²C interface and ATECC608A will be selected if bits 1 thru 7 of this byte match bits 1 thru 7 of the I2C_Address byte in the Configuration zone. Bit 0 of this byte is the standard I²C R/W pin, and should be one to indicate that the bytes following the device address travel from the slave to the master (Read).
Data	Data1,N	To master	The output group, consisting of the count, status/error byte or the output packet followed by the two byte CRC per Section 10.1, I/O Groups.

The status, error, or command outputs can be read repeatedly by the master. Each time a `Read` command is sent to the ATECC608A along the I²C interface, the device transmits the next sequential byte in the output buffer. See the following section for details on how the device handles the address counter.

If the ATECC608A is busy, idle, or asleep, it will NOT ACK the device address on a read sequence. If a partial command has been sent to the device and a read sequence [Start + DeviceAddress(R/W == R)] is sent to the device, then the ATECC608A will *not* ACK the device address to indicate that no data is available to be read.

7.6 Address Counter

Writes to and/or reads from the ATECC608A I/O Buffer over the I²C interface are treated as if the device were a FIFO. Either the I²C byte or page write/read protocols can be used. The number of bytes transferred with each page sequence does not affect the operation of the device.

The first byte transmitted to the device is treated as the size byte. Any attempt to send more than this number of bytes, or any attempts to write beyond the end of the I/O Buffer (71 bytes) will cause the ATECC608A to *not* ACK those bytes.

After the host writes a single command byte to the input buffer, reads are prohibited until after the device completes command execution. Attempts to read from the device prior to the last command byte being sent will result in an ACK of the device address but all ones (`0xFF`) on the bus during the data intervals because the device is still waiting for the completion of the command transmission. If the host attempts to send a read byte after the last byte of the command has been transmitted, the device will be executing the command and will NOT ACK the device address.

Data may be read from the device under the following three conditions:

- On power-up, the single byte `0x11` (Section 10.3, Status/Error Codes) can be read inside a four byte group.
- If a complete block has been received by the device, but there are any errors in parsing or executing the command, a single byte of error code is available (also inside a four byte group).

- Upon completion of a command execution from 1 to 32 bytes of command, results are available to be read inside a group of 4 to 35 bytes.

Any attempt to read beyond the end of the valid output buffer returns `0xFF` to the system, and the address counter does ***not*** wrap around to the beginning of the buffer.

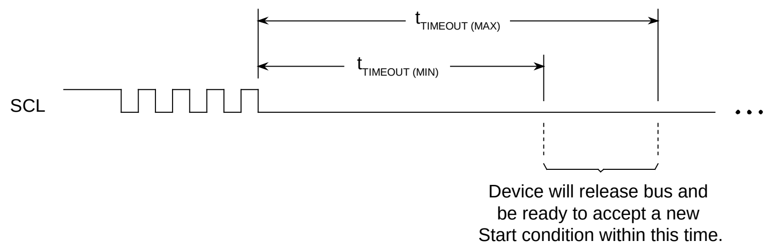
There may be situations where the system may wish to re-read the output buffer, for example when the CRC check reveals an error. In this case, the host should send a two-byte sequence to the ATECC608A consisting of the correct device address and a word address of `0x00` (Reset, per Table 7-2, Word Address Values), followed by a Stop condition. This causes the address counter to be reset to zero and permits the data to be rewritten (or re-read) to (or from) the device. This address reset sequence does not prohibit subsequent read operations if data were available for reading in the I/O Buffer prior to the sequence execution.

After one or more read operations to retrieve the results of a command execution, the first write operation resets the address counter to the beginning of the I/O Buffer.

7.7 SMBus Timeout

The ATECC608A supports the SMBus Timeout feature in which the ATECC608A will reset its serial interface and release the SMBus (i.e. stop driving the bus and let SDA float high) if the SCL pin is held low for more than the minimum $t_{TIMEOUT}$ specification. The ATECC608A will be ready to accept a new Start condition before $t_{TIMEOUT}$ maximum has elapsed.

Figure 7-5. SMBus Timeout



$t_{TIMEOUT\ (MAX)}$
$t_{TIMEOUT\ (MIN)}$
SCL
Device will release bus and be ready to accept a new Start condition within this time.

7.8 I²C Synchronization

It is possible for the system to lose synchronization with the I/O port on the ATECC608A, perhaps due a system reset, I/O noise, or other condition. Under this circumstance, the ATECC608A may not respond as expected, may be asleep, or may be transmitting data during an interval when the system is expecting to send data. To resynchronize, the following procedure should be followed:

1. To ensure an I/O channel reset, the system should send the standard I^{2}C software reset sequence, as follows:
   - A Start bit condition.
   - Nine cycles of SCL, with SDA held high.
   - Another Start bit condition.
   - A Stop bit condition.

   It should then be possible to send a read sequence, and if synchronization has completed properly, the ATECC608A will ACK the device address. The device may return data or may leave the bus floating (which the system will interpret as a data value of `0xFF`) during the data periods.

   If the device does ACK the device address, the system should reset the internal address counter to force the ATECC608A to ignore any partial input command that may have been sent. This can be accomplished by sending a write sequence to word address `0x00` (Reset), followed by a Stop condition.

2. If the device does ***not*** respond to the device address with an ACK, then it may be asleep. In this case, the system should send a complete Wake token and wait t_{WHI} after the rising edge. The system may then send another read sequence, and if synchronization has completed, the device will ACK the device address.
3. If the device still does not respond to the device address with an ACK, then it may be busy executing a command. The system should wait the longest t_{EXEC} (max) and then send the read sequence, which will be acknowledged by the device.

8 General Purpose I/O Pin

When the Single-Wire interfaces is enabled, the SCL pin is available to be used as a GPIO pin. It may be used to drive one or two LEDs or can be connected to an external tamper detection switch or connected in many other ways. When configured as an output, it may be used as an enable pin for some external component in the system which may require cryptographic validation prior to assertion.

On initial power-up, the pin is always temporarily configured as an input. During the device initialization, which occurs with the very first wake operation, the contents of the I2C_Address field are read, and the GPIO pin will be driven to the state. The direction (input or output) and state (if an output) of the GPIO pin will remain unchanged during sleep and idle states. The actions of this pin are controlled by the I2C_Address byte in the Configuration zone, and the GPIO mode of the `Info` command as described in the table below:

Table 8-1. GPIO Mode

Bit 3	Bit 2	Bit 1	Bit 0	Name	Power-Up State	Meaning
x	x	0	0	Disable	Input	The SCL pin is unused and should be tied to GND. Any attempt to execute the GPIO mode of the `Info` command will result in an error code being returned to the system firmware. The GPIO mode of the `Info` command will also return an error code if the part is configured for I^{2}C operation.
0	0	0	1	Auth0	Low	The SCL pin will be permanently configured as an output and will be driven to a zero (default) state when the first wake operation after power-up occurs. The pin can then be driven to the opposite ('1') state by the `Info` command if a prior authorization has been performed using the SignalKey slot. The GPIO output mode of the `Info` command can be used to reset the pin back to the default value without authorization. The GPIO retains its state so long as V_{CC} remains above 2V.
0	1	0	1	Auth1	High	As Auth0; however, the default state after power-up is one.
1	x	0	1	Intrusion Detection	Input	The SCL pin will be permanently configured as an input. The Persistent latch is set via authorization and is cleared if SCL falls below 1.8V. Any falling edge on the SCL pin resets the Persistent latch to zero regardless of whether or not the ATECC608A is in wake or sleep mode. A GPIO read via the `Info`(mode=3) command returns the value of the Persistent latch; not the current state of the pin.
x	x	1	0	Input	Input	The SCL pin will remain permanently configured as an input. Execution of the `Info` command will permit the current state on the pin to be returned to the system firmware.
x	0	1	1	Output0	Low	The SCL pin will be configured as an output and will be driven to a zero state when the first wake operation occurs. Subsequent `Info` commands can be executed to drive the pin high or low. Alternatively, the `Info` command can be used to change the GPIO pin to an input.
x	1	1	1	Output1	High	As Output0; however, the default state after power-up is one.

The GPIO pin has active drivers for both the high and low output states to enable connection to two different LEDs, which may be connected to V_{CC} and GND respectively. If an LED is connected to a supply voltage higher than V_{CC}, it may not turn off completely when the GPIO pin is high. In this case, the GPIO pin should be transitioned to an input to completely turn off the LED.

9 Electrical Characteristics

9.1 Absolute Maximum Ratings*

Operating Temperature	-40°C to 85°C
Storage Temperature	-65°C to 150°C
Maximum Operating Voltage	6.0V
DC Output Current	5mA
Voltage on any pin	-0.5V to (V_{CC} + 0.5V)

*Notice: Stresses beyond those listed under “Absolute Maximum Ratings” may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions beyond those indicated in the operational sections of this specification are not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

9.2 Reliability

The ATECC608A is fabricated with the Microchip high reliability of the CMOS EEPROM manufacturing technology.

Table 9-1. EEPROM Reliability

Parameter	Min	Typical	Max	Units
Write Endurance at 85°C (Each Byte)	400,000			Write Cycles
Data Retention at 55°C	10			Years
Data Retention at 35°C	30	50		Years
Read Endurance	Unlimited			Read Cycles

9.3 AC Parameters: All I/O Interfaces

Figure 9-1. AC Timing Diagram: All Interfaces

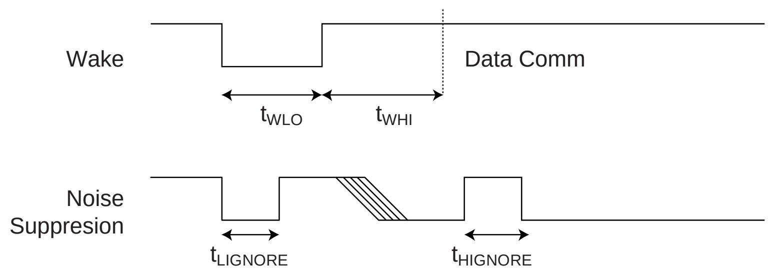


Figure 9-2. AC Parameters: All I/O Interfaces

Parameter	Symbol	Direction	Min	Typ	Max	Unit	Notes
Power-Up Delay[2]	t_{PU}	To Crypto Authentication	100		—	µs	Minimum time between V_{CC} > V_{CC} min prior to measurement of t_{WLO}.
Wake Low Duration	t_{WLO}	To Crypto Authentication	60		—	µs	
Wake High Delay to Data Comm.	t_{WHI}	To Crypto Authentication	1500			µs	SDA should be stable high for this entire duration.
High Side Glitch Filter at Active	$t_{HIGNORE\_A}$	To Crypto Authentication	45[1]			ns	Pulses shorter than this in width will be ignored by the device, regardless of its state when active.
Low Side Glitch Filter at Active	$t_{LIGNORE\_A}$	To Crypto Authentication	45[1]			ns	Pulses shorter than this in width will be ignored by the device, regardless of its state when active.
Low Side Glitch Filter at Sleep	$t_{LIGNORE\_S}$	To Crypto Authentication	15[1]			µs	Pulses shorter than this in width will be ignored by the device when in sleep mode.
Watchdog Timeout	$t_{WATCHDOG}$	To Crypto Authentication	0.7	1.3	1.7	s	Time from wake until device is forced into sleep mode if Config.ChipMode.Bit2 is 0. See Section 10.6, Watchdog Failsafe.
			7.6	10	13.1	s	Watchdog time : Config.ChipMode.Bit2 is 0.

Note 1. These parameters are guaranteed through characterization, but not tested.
2. The power-up delay will be significantly longer if Power-On self test is enabled in the configuration zone.

9.3.1 AC Parameters: Single-Wire Interface

Figure 9-3. AC Timing Diagram: Single-Wire Interface

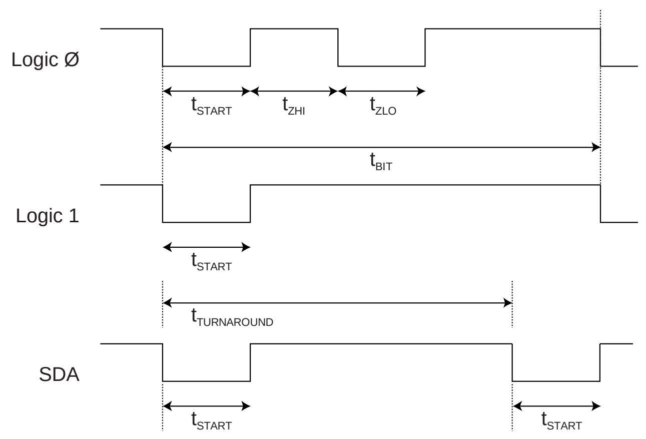


Table 9-2. AC Parameters: Single-Wire Interface

Applicable from T_A = -40°C to +85°C, V_{CC} = +2.0V to +5.5V, CL =100pF (unless otherwise noted).

Parameter	Symbol	Direction	Min	Typ	Max	Unit	Notes
Start Pulse Duration	t_{START}	To Crypto Authentication	4.10	4.34	4.56	µs	
		From Crypto Authentication	4.60	6	8.60	µs	
Zero Transmission High Pulse	t_{ZHI}	To Crypto Authentication	4.10	4.34	4.56	µs	
		From Crypto Authentication	4.60	6	8.60	µs	
Zero Transmission Low Pulse	t_{ZLO}	To Crypto Authentication	4.10	4.34	4.56	µs	
		From Crypto Authentication	4.60	6	8.60	µs	
Bit Time(1)	t_{BIT}	To Crypto Authentication	37	39	—	µs	If the bit time exceeds $t_{TIMEOUT}$ then ATECC608A may enter the sleep mode. See Section 6.3.1, I/O Timeout.
		From Crypto Authentication	41	54	78	µs	
Turn Around Delay	$t_{TURNAROUND}$	From Crypto Authentication	64	96	131	µs	ATECC608A will initiate the first low going transition after this time interval following the initial falling edge of the start pulse of the last bit of the transmit flag.
		To Crypto Authentication	93			µs	After ATECC608A transmits the last bit of a group, system must wait this interval before sending the first bit of a flag. It is measured from the falling edge of the start pulse of the last bit transmitted by ATECC608A.
IO Timeout	$t_{TIMEOUT}$	To Crypto Authentication	45	65	85	ms	ATECC608A may transition to the sleep mode if the bus is inactive longer than this duration. See Section 6.3.1.

Note 1. START, ZLO, ZHI, and BIT are designed to be compatible with a standard UART running at 230.4Kbaud for both transmit and receive. The UART should be set to seven data bits, no parity and one Stop bit.

9.3.2 AC Parameters: I²C Interface

Figure 9-4. I²C Synchronous Data Timing

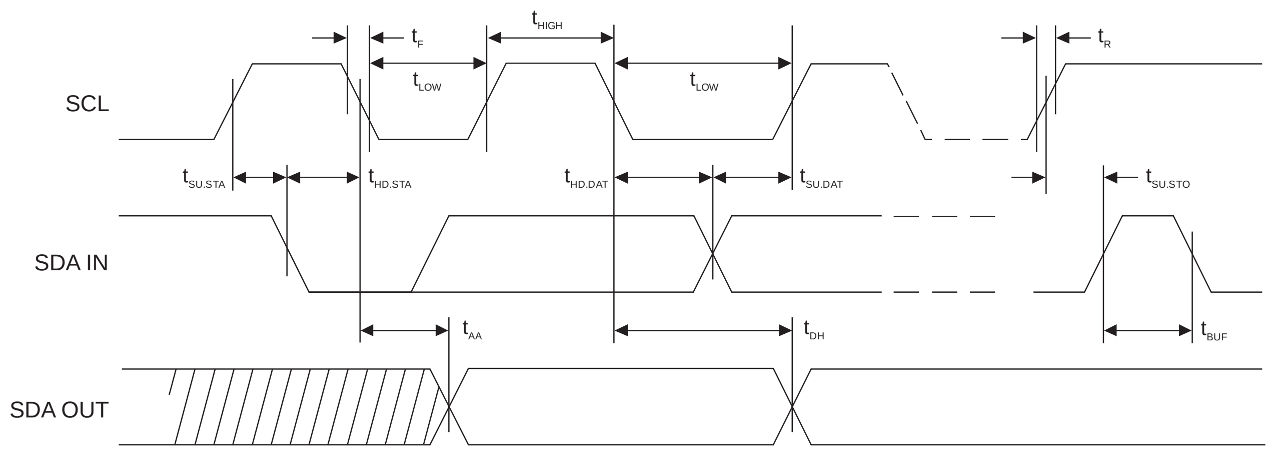


Table 9-3. AC Characteristics of I²C Interface

Applicable over recommended operating range from TA = -40°C to + 85°C, V_{CC} = +2.0V to +5.5V, CL = 1 TTL Gate and 100pF (unless otherwise noted).

Symbol	Parameter	Min	Max	Units
f_{SCK}	SCK Clock Frequency	0	1	MHz
t_{HIGH}	SCK High Time	400		ns
t_{LOW}	SCK Low Time	400		ns
$t_{SU.STA}$	Start Setup Time	250		ns
$t_{HD.STA}$	Start Hold Time	250		ns
$t_{SU.STO}$	Stop Setup Time	250		ns
$t_{SU.DAT}$	Data In Setup Time	100		ns
$t_{HD.DAT}$	Data In Hold Time	0		ns
t_R	Input Rise Time(1)		300	ns
t_F	Input Fall Time(1)		100	ns
t_{AA}	Clock Low to Data Out Valid	50	550	ns
t_{DH}	Data Out Hold Time	50		ns
$t_{TIMEOUT}$	SMBus Timeout Delay	25	75	ms
t_{BUF}	Time bus must be free before a new transmission can start. (1)	500		ns

Note
1. Values are based on characterization and are not tested.
2. AC measurement conditions:
   - RL (connects between SDA and V_{CC}): 1.2k (for V_{CC} +2.0V to +5.0V)
   - Input pulse voltages: $0.3V_{CC}$ to $0.7V_{CC}$
   - Input rise and fall times: ≤ 50ns
   - Input and output timing reference voltage: $0.5V_{CC}$

9.4 DC Parameters: All I/O Interfaces

Table 9-4. DC Parameters on All I/O Interfaces

Parameter	Symbol	Min	Typ	Max	Unit	Notes
Ambient Operating Temperature	T_A	-40		85	°C	
Power Supply Voltage	V_{CC}	2.0		5.5	V	
Active Power Supply Current	I_{CC}		2	3	mA	Waiting for I/O during I/O transfers or execution of non-ECC commands. Independent of Clock Divider value.
			—	14	mA	During ECC command execution. Clock divider = 0x0
				6	mA	During ECC command execution. Clock divider = 0x5
				3	mA	During ECC command execution. Clock divider = 0xD
Idle Power Supply Current	I_{IDLE}		800		µA	When device is in idle mode, V_{SDA} and V_{SCL} < 0.4V or > V_{CC} – 0.4
Sleep Current	I_{SLEEP}		30	150	nA	When device is in sleep mode, $V_{CC} \leq 3.6V$, V_{SDA} and V_{SCL} < 0.4V or > V_{CC} – 0.4, $T_A \leq 55°C$
				2	µA	When device is in sleep mode.
Output Low Voltage	V_{OL}			0.4	V	When device is in active mode, V_{CC} = 2.5 – 5.5V
Output Low Current	I_{OL}			4	mA	When device is in active mode, V_{CC} = 2.5 – 5.5V, V_{OL} = 0.4V
Theta JA	Θ_{JA}		166		°C/W	SOIC (SSH)
			173		°C/W	UDFN (MAH)
			146		°C/W	RBH

9.4.1 V_{IH} and V_{IL} Specifications

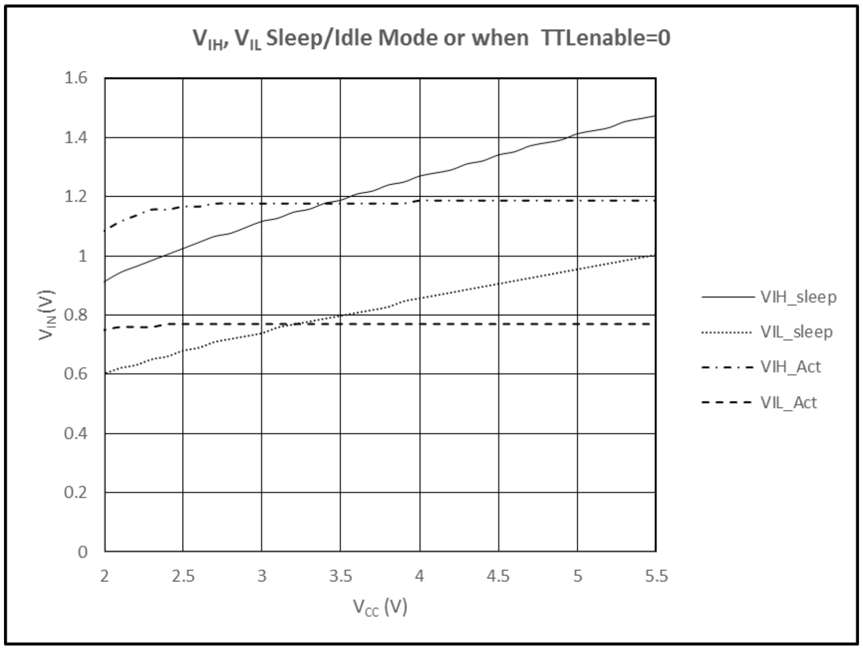
The input levels of the device will vary dependent on the mode and voltage of the device. The input voltage thresholds when in sleep or idle mode are dependent on the V_{CC} level as shown in Figure 8-5. When in sleep or idle mode the TTLenable bit has no effect.

When the device is active (i.e. not in sleep or idle mode), the input voltage thresholds are different depending upon the state of TTLenable (bit 1) within the ChipMode byte in the Configuration zone of the EEPROM. If the voltage supplied to the V_{CC} pin of the ATECC608A is different than the system voltage to which the input pull-up resistor is connected, then the system designer may choose to set TTLenable to zero, which enables a fixed input threshold also shown in Figure 8.5 Table 8-5 also shows the guaranteed levels of operation when operating in this mode.Table 8-5 only applies when the device is active:

Table 9-5. V_{IL}, V_{IH} on All I/O Interfaces (TTLenable=0)

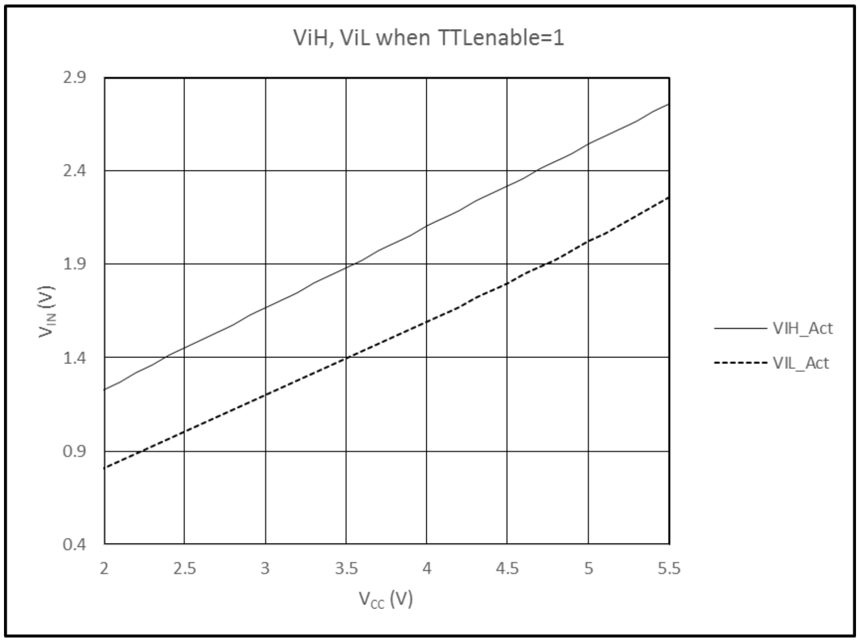
Parameter	Symbol	Min	Typ	Max	Unit	Notes
Input Low Voltage	V_{IL}	-0.5		0.5	V	When device is active and TTLenable bit in configuration memory is zero; otherwise see above.
Input High Voltage	V_{IH}	1.5		V_{CC} + 0.5	V	When device is active and TTLenable bit in configuration memory is zero; otherwise see above.

Figure 9-5. V_{IH} and V_{IL} in Sleep and Idle Mode or When TTLenable = 0 on All I/O Interfaces



When a common voltage is used for the ATECC608A V_{CC} pin and the input pull-up resistor, then the TTLenable bit should be set to a one, which permits the input thresholds to track the supply as shown below.

Figure 9-6. V_{IH} and V_{IL} When Active and TTLenable = 1 on All I/O Interfaces



10 General Command Information

10.1 I/O Groups

Regardless of the I/O protocol being used (i.e. either Single-Wire Interface or I^2C); security commands are sent to the device and responses received from the device within a group that is constructed in the following way:

Table 10-1. I/O Groups

Byte	Name	Meaning
0	Count	Number of bytes to be transferred to (or from) the device in the group, including count byte, packet bytes, and checksum bytes. The count byte should therefore always have a value of (N+1), where N is equal to the number of bytes in the packet plus the two checksum bytes. For a group with one count byte, 50 packet bytes, and two checksum bytes, the count byte should be set to 53. The maximum size group (and value of count) is 155 bytes, and the minimum size group is four bytes. Values outside this range will cause the device to return an I/O Error.
1 to (N-2)	Packet	Command, parameters and data, or response. See below for more details.
N-1, N	Checksum	CRC-16 verification of the count and packet bytes. The CRC polynomial is `0x8005`. The initial register value should be zero and after the last bit of the count and packet have been transmitted; the internal CRC register should have a value that matches the checksum bytes in the block. The first CRC byte transmitted (N-1) is the least-significant byte of the CRC value, so the last byte of the group is the most-significant byte of the CRC.

The ATECC608A is designed in such a way that the count value in the input group should be consistent with the size requirements that are specified in the command parameters. If the count value is inconsistent with the command opcode and/or parameters within the packet, then the ATECC608A will respond in different ways depending upon the specific command. The response may either include an error indication or some input bytes may be silently ignored.

10.2 Command Packets

The command packet is broken down as shown in Table 10-2, below:

Table 10-2. Command Packets

Byte	Name	Meaning
0	Opcode	The command code. See Section 10.4, Command Summary and Execution Times.
1	Param1	The first parameter; always present.
2 – 3	Param2	The second parameter; always present.
4+	Data	Optional remaining input data.

After the ATECC608A receives all the bytes in a group, the device transitions to the busy state and attempts to execute the command. Neither status nor results can be read from the device when it is busy. During this time, the I/O interface of the device ignores all SDA transitions regardless of the I/O interface selected. The command execution delays are listed in Section 10.4.

If insufficient bytes are sent to the device when it is in Single-Wire mode, the device automatically transitions to the low-power sleep mode after the $t_{TIMEOUT}$ interval. In I^2C mode, the device continues to wait for the remaining bytes until the watchdog timer limit $t_{WATCHDOG}$ is reached, or a Start/Stop condition is received by the device.

10.3 Status/Error Codes

The device does not have a dedicated status register, so the output FIFO is shared among status, error, and command results. All outputs from the device are returned to the system as complete groups which are formatted identically to input groups:

- Count
- Packet
- Two byte CRC

After the device receives the first byte of an input command group, the system cannot read anything from the device until the system has sent all the bytes to the device.

After wake and after execution of a command, there will be error, status, or result bytes in the device's output register that can be retrieved by the system. When the length of that group is four bytes, the codes returned are detailed in Table 10-3, below. Some commands return more than four bytes when they execute successfully. The resulting packet description is listed in the Command section that follows.

CRC errors are always returned before any other type of error. They indicate that some sort of I/O error occurred, and that the command may be resent to the device. No particular precedence is enforced among the remaining errors if more than one occurs.

Table 10-3. Status/Error Codes in Four byte Groups

State Description	Error/Status	Description
Successful Command Execution	`0x00`	Command executed successfully.
Checkmac or Verify Miscompare	`0x01`	The `CheckMac` or `Verify` command was properly sent to the device, but the input client response did not match the expected value.
Parse Error	`0x03`	Command was properly received but the length, command opcode, or parameters are illegal regardless of the state (volatile and/or EEPROM configuration) of the ATECC608A. Changes in the value of the command bits must be made before it is re-attempted.
ECC Fault	`0x05`	A computation error occurred during ECC processing that caused the result to be invalid. Retrying the command may result in a successful execution.
Self Test Error	`0x07`	There was a self test error and the chip is in failure mode waiting for the failure to be cleared.
Execution Error	`0x0F`	Command was properly received but could not be executed by the device in its current state. Changes in the device state or the value of the command bits must be made before it is re-attempted.
After Wake, Prior to First Command	`0x11`	Indication that ATECC608A has received a proper Wake token.
Watchdog About to Expire	`0xEE`	There is insufficient time to execute the given command before the watchdog timer will expire. The system must reset the watchdog timer by entering the idle or sleep modes.
CRC or Other Communications Error	`0xFF`	Command was ***not*** properly received by AT88SHA204 and should be re-transmitted by the I/O driver in the system. No attempt was made to parse or execute the command.

10.4 Command Summary and Execution Times

10.4.1 Command Summary

The following commands are implemented by the ATECC608A:

Table 10-4. Command Opcodes, Short Descriptions, and Execution Time

Command	Opcode	Description
`AES`	`0x51`	Execute the AES-ECB Encrypt or Decrypt functions. Calculate a Galois Field Multiply.
`CheckMac`	`0x28`	Verify a MAC calculated on another CryptoAuthentication device.
`Counter`	`0x24`	Read or increment one of the monotonic counters
`DeriveKey`	`0x1C`	Derive a target key value from the target or parent key.
`ECDH`	`0x43`	Generate an ECDH master secret using stored private key and input public key.
`GenDig`	`0x15`	Generate a data digest from a random or input seed and a key.
`GenKey`	`0x40`	Generate an ECC public key. Optionally generate an ECC private key.
`Info`	`0x30`	Return device state information.
`KDF`	`0x56`	Implement the PRF or HKDF key derivation functions
`Lock`	`0x17`	Prevent further modifications to a zone or slot of the device.
`MAC`	`0x08`	Calculate response from key and other internal data using SHA-256.
`Nonce`	`0x16`	Generate a 32-byte random number and an internally stored Nonce.
`PrivWrite`	`0x46`	Write an ECC private key into a slot in the Data zone.
`Random`	`0x1B`	Generate a random number.
`Read`	`0x02`	Read four bytes from the device, with or without authentication and encryption.
`SecureBoot`	`0x80`	Validate code signature or code digest on power-up
`SelfTest`	`0x77`	Test the various internal cryptographic computation elements
`Sign`	`0x41`	ECDSA signature calculation.
`SHA`	`0x47`	Computes a SHA-256 or HMAC digest for general purpose use by the system.
`UpdateExtra`	`0x20`	Update bytes 84 or 85 within the Configuration zone after the Configuration zone is locked.
`Verify`	`0x45`	ECDSA verify calculation.
`Write`	`0x12`	Write 4 or 32 bytes to the device, with or without authentication and encryption.

10.4.2 Command Execution times

During execution of a properly received command, the device will be busy and not respond to transitions on the pins. The interval during which the device will be busy varies depending upon the command parameters, device state, the environmental conditions, and other factors.

The following table shows representative times for the duration to execute the command assuming no error conditions, typical mode setting and chip state. Some modes may be faster, while others may be slower.

Table 10-5. Typical Execution Time in ms

Command	Condition	Clock Divider 0	Clock Divider 5	Clock Divider D
AES	Mode = Encrypt	1	1	1
CheckMac		8	8	8
Counter	Mode = Read	0.5	0.5	0.5
DeriveKey		15	15	15
ECDH	Target = EEPROM	28	94	370
GenDig		11	11	11
GenKey (First)[1]	Mode = Create	59	115	350
GenKey (Subsequent)[1]	Mode = Create	59	115	350
Info	DevRev	0.5	0.5	0.5
KDF	Mode = PRF	99	99	99
Lock	Mode = Config, check CRC	15	15	15
MAC		7	7	7
Nonce (First)[1]	Mode = Random	17	17	17
Nonce (Subsequent)[1]	Mode = Random	11	11	11
PrivWrite		29	29	29
Random (First)[1]		15	15	15
Random (Subsequent)[1]		15	15	15
Read	32 bytes	0.8	0.8	0.8
Secure Boot	Stored Digest. No encrypt or MAC	0.9	0.9	0.9
SelfTest	Mode = 0xFF (All algorithms)	110	350	1400
Sign (First)[1]	Mode = External	64	124	366
Sign (Subsequent)[1]	Mode = External	60	113	378
SHA	Update, message = 64 bytes	1	1	1
UpdateExtra		8	8	8
Verify	Mode = Fixed	27	90	354
Write	32 bytes	8	8	8

In a typical polling configuration, the host software should delay for the interval in the table above and then start polling to determine actual command completion. The three columns in this table correspond to the legal values of the clock divider field of the chip mode configuration byte. In most but not all cases, failing commands will return relatively quickly.

Some commands noted with (^{1}) in the table above use the random number generator (RNG). The first time the RNG is called in any wake/power cycle there will be additional time for the NIST instantiation sequence. Subsequent use of the RNG for any purpose does not require this to be re-run. (First) indicates that this delay is appropriate for the first execution of this command after power-up or wake but before any random numbers have been generated. (Subsequent) indicates that this delay is appropriate for situation in which a random number has already been generated.

Do not use the numbers in this table for software timeouts that drive errors. These delays represent *typical* times only and the actual execution time may be shorter or significantly longer depending on the mode, chip state and/or configuration. Timeout intervals should be at least 50ms more than the typical values listed in the table above when configured with the clock divider of 00. Other clock divider values will require additional extra time.

The following situations can cause commands to take longer to execute than the typical times above:

- Some modes indicate extra computation, for instance an output validation MAC, that takes extra time vs the non-MAC version of the command
- In some cases the current state of the chip may require additional operations to be performed which can increase the execution time
- The first random number generated after a wake/power-on will take extra time for the DRBG instantiation operation, as indicated in the table above
- Some commands may write to the EEPROM instead of an internal SRAM register. Depending on the number of pages and/or internal elements to be written, this will take extra time
- The limited use function for each slot requires that the internal monotonic counter be incremented prior to the use of the key. Depending on the current count, the additional execution time can vary significantly. The counter command may also increment one of the monotonic counters
- The various internal security sequences include random delays. On a statistically infrequent basis, the cumulative computation delay may be much longer than typical

Commands may take substantially more time to execute under these and other conditions. The following table shows sample execution times for a few commands where the modes, chip state and parametric conditions have been configured to be close to the longest delay possible. This data was gathered with a clock divider value of 00.

Table 10-6. Example Long Execution Time in ms

Command	Example Long Execution Time (ms)
Sign	75
GenKey	90
Verify	67
Nonce	20
SecureBoot(Full Store)	82

10.5 Address Encoding

The `Read` and `Write` commands include a single 16 bit address in Param2, which indicates the memory location to be accessed. In all cases, the least significant two bytes of the byte address should be dropped from the parameter that is passed to the device to create a word address.

The `Read` and `Write` commands support either 4 or 32 byte accesses. When 32 bytes are being accessed, the offset (i.e. the least significant three bits of the word address) must be present in the parameter, but their value in the parameter is ignored, and the operation proceeds assuming they are zero (i.e. all 32 byte accesses are block aligned).

10.5.1 Zone Encoding

The value in Param1 controls which zone the command accesses. See Section 2.4.1, Configuration Zone Locking to obtain more information on what controls the locked and unlocked states for each zone. All other zone values are reserved and should not be used.

Table 10-7. Zone Encoding (Param1)

Zone	Param1 Value	Size	Read	Write
Config	0	1024 bits 128 bytes 4 Blocks	Always available.	Partially, when unlocked. Never when locked. Never encrypted.
OTP	1	512 bits 64 bytes 2 Blocks	Never when unlocked. Always when locked.	All writeable when unlocked using `Write`. When locked, no writes are permitted.
Data	2	9664 bits 1208 bytes 16 Slots	Never when unlocked; otherwise, controlled by IsSecret and EncryptRead.	All writeable when unlocked. When locked, writes controlled by WriteConfig.

Table 10-8. Address Encoding for Config and OTP Zones (Param2)

Zone	Byte 1	Byte 0		
	Unused	Unused	Block	Offset
Config	Bits 0 → 7	Bits 5 → 7	Bits 3 → 4	Bits 0 → 2
OTP	Bits 0 → 7	Bits 4 → 7	Bit 3	Bits 0 → 2

Table 10-9. Address Encoding for Data Zone (Param2)

Zone	Byte 1		Byte 0		
	Unused	Block	Unused	Slot	Offset
Data Slots 0 – 7	Bits 1 → 7	Bit 0	Bit 7	Bits 3 → 6	Bits 0 → 2
Data Slot 8	Bits 4 → 7	Bits 0 → 3	Bit 7	Bits 3 → 6	Bits 0 → 2
Data Slots 9 – 15	Bits 2 → 7	Bits 0 and 1	Bit 7	Bits 3 →6	Bits 0 → 2

Within each zone, there are various access restrictions as noted in the table below:

Table 10-10. Legal Block/Slot Values

Zone Name	Legal Block	Legal Slot	Notes
Config	0 – 3	—	Addresses below 16 (Block 0, Offset 16) can never be written. Addresses from 84-87 cannot be written using the `Write` command. Both 4-byte and 32-byte reads/writes are permitted.
OTP	0 – 1	—	When OTP is unlocked, all offsets in both blocks are available to use with 4 or 32 byte reads. If OTP zone is locked, no writes to OTP are permitted.
Data	0 – 1	0 – 7	All offsets in all slots available for both `Read` and `Write`. A 4-byte access is permitted on a particular slot only if SlotConfig.IsSecret is zero.
	0 – 12	8	
	0 – 2	9 – 15	

In the following table, address is the value to be passed to the `Read` and/or `Write` commands as the address parameter to access data in the specific blocks using a 32 byte Read or Write. Size is the number of implemented EPROM bytes within that particular block.

To use a four byte `Read` or `Write` command to access the first word in a block, use the addresses shown in Table 10-11; otherwise, the least significant three bits of the address field should include the word address to be accessed. The 32 byte access is permitted in blocks that contain less than 32 implemented memory bytes. The extra bytes will be returned as zero on a read and ignored on a Write.

Table 10-11. Data Zone Address Values

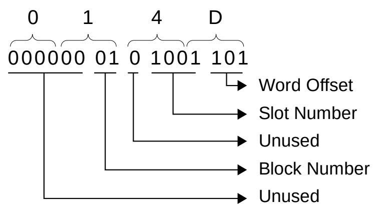
	Block 0		Block 1		Block 2		Block 3	
Slot	Address	Size	Address	Size	Address	Size	Address	Size
0	`0x0000`	32	`0x0100`	4				
1	`0x0008`	32	`0x0108`	4				
2	`0x0010`	32	`0x0110`	4				
3	`0x0018`	32	`0x0118`	4				
4	`0x0020`	32	`0x0120`	4				
5	`0x0028`	32	`0x0128`	4				
6	`0x0030`	32	`0x0130`	4				
7	`0x0038`	32	`0x0138`	4				
8	`0x0040`	32	`0x0140`	32	`0x0240`	32	`0x0340`	32
9	`0x0048`	32	`0x0148`	32	`0x0248`	8		
10	`0x0050`	32	`0x0150`	32	`0x0250`	8		
11	`0x0058`	32	`0x0158`	32	`0x0258`	8		
12	`0x0060`	32	`0x0160`	32	`0x0260`	8		
13	`0x0068`	32	`0x0168`	32	`0x0268`	8		
14	`0x0070`	32	`0x0170`	32	`0x0270`	8		
15	`0x0078`	32	`0x0178`	32	`0x0278`	8		

Example: To complete a four byte read of the 53rd through 56th byte of Slot 9, the word address would be:

The 53rd byte is the 21st byte in Block 1 (53 divided by 32 is 1, 53 minus 32 is 21).

The 21st byte is located at byte offset `0x14`, which is at word offset `0x05` (`0x14` divided by 4 is `0x05`).

Per Table 10-9, the address parameter to the `Read` command is `000000 01 0 1001 101` or `0x014`.



10.6 Watchdog Failsafe

After the ATECC608A receives a Wake token, a watchdog counter starts within the device after $t_{WATCHDOG}$, the device enters sleep mode regardless of whether some I/O transmission or command execution is in progress. There is no way to reset the counter other than to put the device into sleep or idle mode and then wake it up again.

The watchdog timer is implemented as a fail-safe mechanism where no matter what happens on either the system side or inside the device, including any I/O synchronization issue, power consumption will fall to the ultra-low sleep level automatically.

The device resets the values stored in the SRAM and internal status registers when it transitions to the sleep mode; however, if the device is explicitly put into the idle mode through the appropriate I/O sequence, then the device retains the contents of the SRAM registers.

Normally, all command sequences must complete within $t_{WATCHDOG}$ if they require a state that is stored in the SRAM registers. If there is a need to implement a command sequence that is longer than the watchdog interval, the system software can use this idle mode mechanism to ensure that the sequence can complete successfully..

If a command is attempted when insufficient time remains prior to watchdog timer execution, the device will return the Watchdog Timeout error code without attempting to execute the command. This feature prevents situations in which the command may only be partially executed at the time the watchdog timer resets the device. In particular, the limited use counter is always decremented prior to execution of the crypto computation; therefore, an aborted command might result in fewer counts remaining without the result being available to the system. The device will never be left in an unusable state after an aborted command.

Due to the extended nature of some networking sequences, the ATECC608A provides an extended watchdog interval to be enabled via the configuration zone ChipMode byte. For highest security, Microchip always recommends the shorter watchdog interval. See Section 2.2 for more details.

11 Detailed Command Descriptions

11.1 AES Command

The AES command executes the AES algorithm in one of two modes:

Encrypt-ECB: 16 bytes of cleartext are expected on input, the AES encrypt operation is performed on the input block and the output 16 bytes are the encrypted result.

Decrypt-ECB: 16 bytes of ciphertext are expected on input, the AES decrypted operation is performed on the input block and the output 16 bytes are the cleartext result.

Galois Field Multiply (GFM): Take the first 16 bytes input after Param2 as H and the second 16 bytes as the input data and calculate GFM to be used in the AES-GCM AEAD functionality. This mode does not involve secrets or anything stored on the chip. If this mode is selected the remaining mode bits are ignored.

The AES key can be located in any of 4 possible locations if the input has 64 valid bytes or either of the 2 possible locations if the input has 32 valid bytes. This selection is controlled by bits 6:7 of the mode byte.

The AES command mode is enabled via bit 0 of byte 13 in the configuration area. If 1, the command is enabled, if 0 it is disabled. This configuration byte cannot be written in the field.

Table 11-1. Input Parameters

	Name	Size	Notes
Opcode	AES	1	0x51
Param1	Mode	1	Bit 0-2: If 000, AES-ECB Encrypt If 001, AES-ECB Decrypt If 011, calculate Galois Field Multiply (GFM) on the input data All other values reserved, do not use Bits 3-5: Reserved for future use, must be zero. Bits 6-7: Use this 16 byte AES block within the source field as the AES key. An error is returned if the source field does not have 64 valid bytes and one of the upper keys is selected.
Param2	KeyID	2	The key slot in the EEPROM to be used as the secret key in the AES calculation if Mode:0 is 0. If 0xFFFF, then TempKey is the source of the AES key.
Data	Input	16	The input to the AES or GCM operation.

Table 11-2. Output Parameter

Name	Size	Notes
Response	1 or 16	If the AES operation was successful the output is 16 bytes, either the AES ECB or GCM output. Otherwise if any error has occurred, the error code.

11.2 `CheckMac` Command

The `CheckMAC` command calculates a MAC response that would have been generated on a different ATECC608A, ATECC508A, or ATSHA204A device and then compares the result with the input value. It returns a Boolean result to indicate the success or failure of the comparison.

Prior to running this command, the `Nonce` and/or `GenDig` commands may have been optionally run to create a key or nonce value in TempKey. The input mode parameter determines the source of the key (the first 32 bytes of SHA message) and challenge/nonce (the second 32 bytes of SHA message). If KeyConfig[KeyID].ReqRandom is one, the RNG must have been used during the execution of the `Nonce` command, or else this command will return an error.

If authorization is required by the KeyConfig before use of a key, this authorization function can be accomplished by executing this command with mode[1] set to zero. TempKey should have been previously loaded with a nonce via the `Nonce` command. If KeyConfig[KeyID].ReqRandom is one, the RNG should have been used during the execution of that `Nonce` command. If the `CheckMac` succeeds, then an internal authComplete flag will be set and KeyID retained internally. See Section 4.4.8, Authorized Keys for more details.

If the comparison matches, then the target EEPROM slot value ***may*** be copied into TempKey. If KeyID is even, then the target slot is KeyID+1, or else the target slot is KeyID. For the copy to take place the mode parameter to `CheckMac` must have a value of `0x01` or `0x05` and slotConfig.readKey for the target key must be zero. This copy will take place regardless of the value of SlotConfig.LimitedUse for the target slot.

Table 11-3. Input Parameters

	Name	Size	Notes
Opcode	CheckMac	1	`0x28`
Param1	Mode	1	Bit 0: `0` = The second 32 bytes of the SHA message are taken from the input ClientChal parameter. `1` = The second 32 bytes of the message are taken from TempKey. Bit 1: `0` = Use key[KeyID] in first SHA block. `1` = Use TempKey. Bit 2: If Mode:0 or Mode:1 are set, then the value of this bit must match the value in TempKey.sourceFlag or the command will return an error. Bits 3 – 7: Must be zero.
Param2	KeyID	2	The internal key is to be used to generate the response. All except bits 0:3 of KeyID are ignored.
Data1	ClientChal	32	Challenge sent to client. If Mode:0 is one, then the value of this parameter will be ignored. (These 32 bytes ***must*** still appear in the input stream).
Data2	ClientResp	32	Response generated by the client.
Data3	OtherData	13	Remaining constant data needed for response calculation.

Table 11-4. Output Parameter

Name	Size	Notes
Result	1	Returns a single byte with a value of zero if ClientResp matches the internally computed digest; value of one if there is a mismatch.

The message that will be hashed with the SHA-256 algorithm consists of the following information:

32 bytes	key[KeyID] or TempKey	(depending on mode)
32 bytes	ClientChal or TempKey	(depending on mode)
4 bytes	OtherData[0:3]	
8 bytes	Zeros	
3 bytes	OtherData[4:6]	
1 byte	SN[8]	
4 bytes	OtherData[7:10]	
2 bytes	SN[0:1]	
2 bytes	OtherData[11:12]	

11.3 `Counter` Command

The `Counter` command reads the binary count value from one of the two monotonic counters located on the device within the Configuration zone. The maximum value that the counter may have is 2,097,151. Any attempt to count beyond this value will result in an error code. The counter is designed to never lose counts even if the power is interrupted during the counting operation. In some power loss conditions, the counter may increment by a value of more than one.

Counter[0] may be attached to some keys to limit their use; counter [1] is never attached to any key. When counter[0] is attached to a key, the counter will be incremented with each use of the key until the counter has reached its maximum value at which point use of the key will no longer be permitted.

The Counter Match function allows Counter[0] incrementing to stop when the value in the counter matches the value in the Config:CountMatchKey EEPROM key slot. CounterMatch is not supported for Counter[1].

The number of legal uses for a key can also be controlled by initializing the counter[0] to a non-zero value at configuration time. Contact Microchip for details.

Table 11-5. Input Parameters

	Name	Size	Notes
Opcode	Counter	1	`0x24`
Param1	Mode	1	Bit 0: `0` = Read the value of the specified counter. `1` = Increment the value of the specified counter. Bit 1-7: Must be zero.
Param2	KeyID	2	The counter to be incremented. Only zero and one are legal values.

Table 11-6. Output Parameter

Name	Size	Notes
Count	4 or 1	Generally, this will be the current binary value of the counter. If KeyID points to in invalid counter ID, a Parse Error will be returned. If a requested increment fails, then an Exec error will be returned.

11.4 `DeriveKey` Command

The device combines the current value of a key with the nonce stored in TempKey using SHA-256 and places the result into the target key slot. SlotConfig[TargetKey].Bit13 must be set or `DeriveKey` will return an error. `DeriveKey` always returns an error if KeyConfig indicates that the slot contains an ECC private key, if the Configuration zone has not been locked, or if the TargetKey slot is individually locked using SlotLocked.

If SlotConfig[TargetKey].Bit12 is zero, the source key that will be combined with TempKey is the target key as specified in the command line (Roll Key operation). If SlotConfig[TargetKey].Bit12 is one, the source key is the parent key of the target key, which is found in SlotConfig[TargetKey].WriteKey (Create Key operation).

Prior to execution of this command, the `Nonce` command must have been run to create a valid nonce in TempKey. If KeyConfig.ReqRandom is one for the source key, this nonce must have been created with the internal RNG or an error will be returned. In all cases, Mode:2 must match the state of TempKey.SourceFlag or the command will return an error.

If SlotConfig[TargetKey].Bit15 is set, an input MAC must be present and have been computed as follows:

SHA-256(ParentKey, Opcode, Param1, Param2, SN[8], SN[0:1])
where the ParentKey ID is always SlotConfig[TargetKey].WriteKey.

If performing a Roll Key operation and KeyConfig[TargetKey].ReqAuth is one, then the appropriate authorization must have been performed using KeyConfig[TargetKey].AuthKey prior to the execution of `DeriveKey`. If performing a Create Key operation and KeyConfig[ParentKey].ReqAuth is one, then the appropriate authorization must have been performed using KeyConfig[ParentKey].AuthKey prior to the execution of `DeriveKey`.

If an input MAC is required and KeyConfig[ParentKey].ReqAuth is one, then the appropriate authorization must have been performed using KeyConfig[ParentKey].AuthKey prior to the execution of `DeriveKey`.

If a parent key is involved in the operation (either SlotConfig[TargetKey].Bit12 or SlotConfig[TargetKey].Bit15 are set) and SlotConfig[ParentKey].LimitedUse is also set, `DeriveKey` returns an error if Counter[0] has reached its limit. `DeriveKey` always ignores LimitedUse for the target key.

If the source and target key are the same, then there is a risk of permanent loss of the key value if power is interrupted during the write operation. If the configuration bits permit it, then the key value may be recovered using an authenticated and encrypted Write based on the parent key.

Table 11-7. Input Parameters

	Name	Size	Notes
Opcode	DeriveKey	1	`0x1C`
Param1	Mode	1	Bit 2: The value of this bit must match the value in TempKey.SourceFlag or the command will return an error. Bits 0:1, 3:7: Must be zero.
Param2	TargetKey	2	Key slot to be written.
Data	MAC	0 or 32	Optional MAC used to validate operation.

Table 11-8. Output Parameter

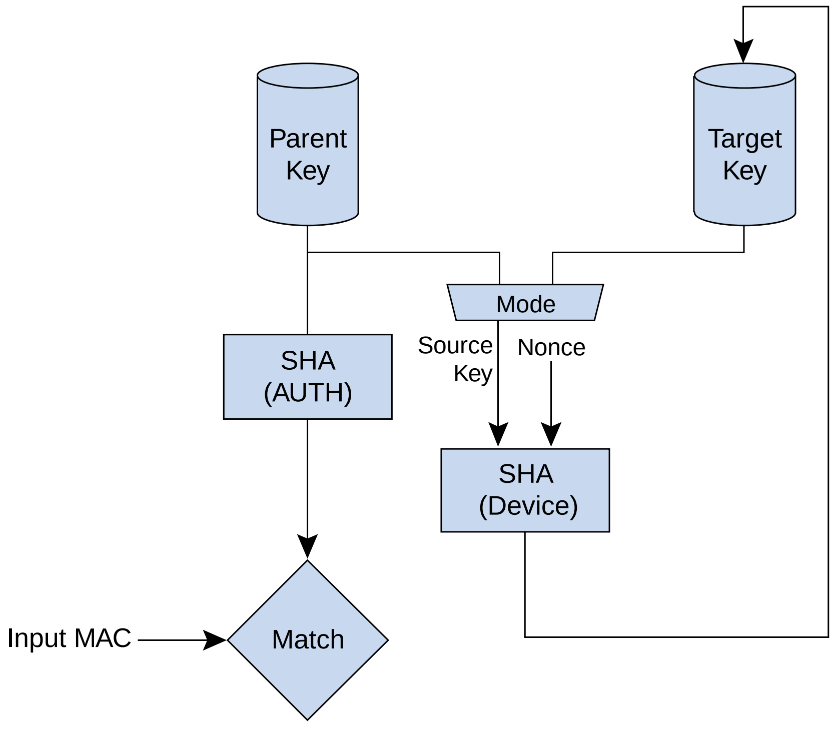
Name	Size	Notes
Success	1	Upon successful completion, the ATECC608A returns a value of zero.

The key written to the target slot is the result of SHA-256 of the following message:

32 bytes	Target or Parent key	(depending upon SlotConfig:Bit 12)
1 byte	Opcode	
1 byte	Param1	
2 bytes	Param2	
1 byte	SN[8]	
2 bytes	SN[0:1]	
25 bytes	Zeros	
32 bytes	TempKey.value	

The data flow for this command is illustrated below.

Figure 11-1. Data Flow for `DeriveKey` Command



11.5 ECDH Command

The ECDH command on the ATECC508A computes the ECDH algorithm to create a shared secret.

There are two possible sources for the private key:

TempKey: The GenKey command must have been previously executed to generate and load a new key into TempKey. TempKey will be cleared after its use on the input but may be re-validated if it is the output target.

EEPROM key slot: The type of the slot must be an ECC private key. Bit 2 in SlotConfig.SlotConfig.ReadKey for the private key must be set to enable the ECDH operation or this command will return an error.

There are three possible destinations for the result, controlled by the mode parameter.

Output Buffer: The output may be encrypted, see below.

TempKey: Any previous value in TempKey will be invalidated. The resulting TempKey can be used by the KDF or AES commands only.

EEPROM key slot: The target slot must NOT be an ECC private key and must have the WriteConfig field set to ALWAYS. The target slot must not be SlotLocked. The host processor may encrypt the premaster secret by executing an encrypted read of the target slot.

If the target field of the mode parameter is COPY (00), then the chip operates in a manner compatible with the ATECC508A: the result is place in either the output buffer or in the KeyID | 1 slot depending on the value of the ReadKey field for the source key. In this case, the WriteConfig check is suppressed. This is the only method to allow both the input and output to be the EEPROM.

If the mode byte indicates that the result should go to the output buffer then depending on the IO Protection configuration, various options are possible:

- If config.chipOption.ioProtEnable is 0, then the result will be placed in the output buffer in the clear, regardless of the state of Mode:Bit 1 (output encryption)
- If config.chipOption.ioProtEnable is 1, then the following possibilities occur:
    - If config.chipOption.ECDHprot is 2 (STORE) then an error will be returned as the configuration prohibits the result on the output port
    - If config.chipOption.ECDHprot is 1 (ENCRYPT) then the output will be encrypted regardless of the state of Mode:Bit 1 (output encryption)
    - If config.chipOption.ECDHprot is 0 (CLEAR) then the output will be encrypted if Mode:Bit 1 is set, otherwise it will be returned to the host in the clear.

Table 11-9. Input Parameters

	Name	Size	Notes
Opcode	ECDH	1	`0x43`
Param1	Mode	1	Bit 0: If 0, source key is EEPROM slot. If 1, TempKey contains the private key Bit 1: If 0, Output is in the clear if permitted. If 1, force output encryption Bit 2-3: 00 – Compatibility copy mode. Result goes to either the output buffer or KeyID+1 depending on the state of config.slotConfig[KeyId].ReadKey:3 01 – Result goes to EEPROM slot specified by KeyID. Config.SlotConfig[KeyID].WriteConfig must be ALWAYS (0000) 10 – Result goes to TempKey 11 – Result goes to the output buffer Bits 4-7: Reserved for future use, must be zero.
Param2	KeyID	2	The private key to be used in the ECDH calculation if Mode:0 is 0
Data1	X	32	The X component of the public key to be used for ECDH calculation.
Data2	Y	32	The Y component of the public key to be used for ECDH calculation.

Table 11-10. Output Parameter

Name	Size	Notes
Response	1 or 32	If the ECDH operation was successful and the destination is the output buffer, then this field contains the shared secret (pre-master secret). Otherwise a success code of `0x00` or if any error has occurred, the error code.
OutNonce	0 or 32	If output is encrypted, the nonce used for the encryption

Output buffer encryption is implemented in the following manner:

1. The first 32 bytes of the IO Protection key slot (config.chipOption[bits 12-15]) are copied to a SHA256 buffer.
2. The internal RNG generates a 32 byte random number (nonce) and appends only the first 16 bytes to the SHA256 buffer
3. This SHA256 buffer is hashed and the digest is XOR'd with the cleartext pre-master secret.
4. The output buffer consists of the encrypted pre-master secret followed by the 32 byte nonce from step 2. Note that the second 32 bytes of the nonce are ignored by the chip and are returned to the host for compatibility with the KDF encryption method only.

11.6 GenDig Command

The GenDig command uses SHA-256 to combine a stored or input value with the contents of TempKey, which must have been valid prior to the execution of this command. The stored value can come from one of the data slots, the Configuration zone, either of the OTP pages, the monotonic counters, or be retrieved from the hardware transport key array. The resulting digest is retained in TempKey, and can be used in one of four ways:

1. It can be included as part of the message used by the MAC, Sign or CheckMac commands. Because the MAC response output incorporates both the data used in the GenDig calculation and the secret key from the MAC command, it serves to authenticate the data stored in the data and/or OTP zones.
2. A subsequent Read or Write command can use the digest to provide authentication and/or confidentiality for the data, in which case it is known as a data protection digest.
3. The command can be used for secure personalization by using a value from the transport key array. The resulting data protection digest would then be used by Write.
4. The input value, typically a nonce from a remote device, is combined with the current TempKey value to create a shared nonce in which both devices can attest to the inclusion of the RNG.

If zone is two (i.e. Data), and KeyID is less than 0x8000, then the GenDig command sets TempKey.GenDigData to one, and TempKey.KeyID to the input KeyID; otherwise, TempKey.GenDigData is set to zero. If KeyConfig.ReqRandom is set for KeyID, and the Data zone is locked, then the value in the TempKey register must have been originally computed using a random number via the Nonce command; otherwise, GenDig will fail. Regardless of how the resulting digest is computed, it can never be read from the device.

If TempKey.Valid is invalid, this command returns an error. Upon command completion, the TempKey.Valid bit is set indicating that a digest has been loaded and is ready for use. The TempKey.Valid bit is cleared when the next command is executed. See Section 3.1 for more details.

For all KeyID values less than 0x8000, the device uses the least-significant four bits of KeyID to determine the slot number from which to retrieve the key value from the Data zone of the EEPROM. KeyID values above 0x8000 reference keys stored in the masks of the design. These keys can only be used if the nonce value stored in TempKey has been generated using the on-board RNG. In any event, all 16 bits of the KeyID as input to the device are used as Param2 in the SHA-256 calculation.

When the key specified on input to GenDig has the NoMac bit set, GenDig can be used to generate ephemeral keys matching those generated on client CryptoAuthentication devices using the DeriveKey command. Keys which have the NoMac bit set represent situations in which the device is acting as a host. In this case, the opcode and parameter bytes that would normally be included in the SHA calculation are replaced with bytes from the input stream.

Table 11-11. Input Parameters

	Name	Size	Notes
Opcode	GenDig	1	`0x15`
Param1	Zone	1	If `0x00` (Config), then use KeyID to specify any of the four 256-bit blocks of the Configuration zone. If KeyID has a value greater than three, the command will return an error. If `0x01` (OTP), use KeyID to specify either the first or second 256-bit block of the OTP zone. If `0x02` (Data), then KeyID specifies a slot in the Data zone or a transport key in the hardware array. If `0x03` (Shared Nonce), then KeyID specifies the location of the input value in the message generation. If `0x04` (Counter), then KeyID specifies the monotonic counter ID to be included in the message generation. If `0x05` (Key Config), then KeyID specifies the slot for which the configuration information is to be included in the message generation. All other values are reserved and must not be used.
Param2	KeyID	2	Identification number of the key to be used, selection of which OTP block or message order for Shared Nonce mode.
Data1	OtherData	32 or 4 or 0	Four bytes of data for SHA calculation when using a NoMac key, 32 bytes for “Shared Nonce” mode, otherwise ignored

Table 11-12. Output Parameter

Name	Size	Notes
Success	1	Upon successful execution, ATECC608A returns a value of zero.

If zone is “Shared Nonce” and KeyID:15 is zero then the SHA-256 message body used to create the resulting new TempKey consists of the following bytes:

32 bytes	Input OtherData Parameter	
1 byte	Opcode	(always `0x15`)
1 byte	Mode	
1 byte	LSB of KeyID	
1 byte	Zero	
1 byte	SN[8]	
2 bytes	SN[0:1]	
25 bytes	Zeros	
32 bytes	TempKey.value	

If zone is “Shared Nonce” and KeyID:15 is one then the SHA-256 message body used to create the resulting new TempKey consists of the following bytes:

32 bytes	TempKey.value	
1 byte	Opcode	(always `0x15`)
1 byte	Mode	
1 byte	LSB of KeyID	
1 byte	Zero	
1 byte	SN[8]	
2 bytes	SN[0:1]	
25 bytes	Zeros	
32 bytes	Input OtherData Parameter	

If zone is Data and SlotConfig[KeyID].NoMac is one, then the SHA-256 message body used to create the resulting new TempKey consists of the following bytes:

32 bytes	Data.slot[KeyID]
4 bytes	OtherData
1 byte	SN[8]
2 bytes	SN[0:1]
25 bytes	Zeros
32 bytes	TempKey.value

If zone is Counter, then the SHA-256 message body used to create the resulting new TempKey consists of the following bytes. Counter mode is supported only on the ATECC608A.

32 bytes	Zeros	
1 byte	Opcode	
1 byte	Param1	
2 bytes	Param2	
1 byte	SN[8]	
2 bytes	SN[0:1]	
1 byte	Zero	
4 bytes	Counter[KeyID]	(binary value as reported by `Counter` command)
20 bytes	Zeros	
32 bytes	TempKey.value	

If zone is “Key Config” (`0x05`), then the SHA-256 message body used to create the resulting new TempKey consists of the following bytes:

32 bytes	TempKey
1 byte	Opcode
1 byte	Mode
2 bytes	Param2
1 byte	SN[8]
2 bytes	SN[0:1]
1 byte	Zero
2 bytes	SlotConfig[KeyId]
2 bytes	KeyConfig[KeyId]
1 byte	SlotLocked:KeyId
19 bytes	Zeros
1 byte	`0x00`

In all other cases, the message use to create TempKey is as follows:

32 bytes	OTP[KeyID] or Data.slot[KeyID] or TransportKey[KeyID]
1 byte	Opcode
1 byte	Param1
2 bytes	Param2
1 byte	SN[8]
2 bytes	SN[0:1]
25 bytes	Zeros
32 bytes	TempKey.value

11.7 GenKey Command

The GenKey command performs one or more of the following three operations:

1. **Private Key Creation:**
   Creates a new random private key. The private key will be written to TempKey if the KeyID param is 0xFFFF. In this case, any previous contents of TempKey are destroyed. The resulting private key in TempKey can be used only by the ECDH command. Otherwise the GenKey command writes the generated key into the slot specified by the KeyID parameter.
2. **Public Key Computation:**
   Generates an ECC public key based upon the private key stored in the slot defined by the KeyID parameter. This mode of the command may be used to avoid storing the public key on the device at the expense of the time required to regenerate it.
3. **Digest Calculation:**
   GenKey can also combine a public key referenced by the KeyID parameter with the current value stored in TempKey, calculate a SHA-256 digest of the resulting message, and place that digest back into TempKey. This digest can be used as the message for an internal signature, or as a component of a MAC computation. TempKey must be valid prior to digest calculation. If KeyConfig.ReqRandom is set, then TempKey must have been created using the internal RNG.

The digest calculation operation can be performed by using either a public key computed from a private key in a slot or by using a public key already stored in a slot. In the latter case, the appropriate checks for prior authorization and limited use will be performed on the public key slot, and the remaining checks indicated below will not be performed. When GenKey is used to calculate a digest on a public key slot, it ignores the validity status of the public key.

Excluding the digest generation operation described above, the slot indicated by this command must be configured by means of KeyConfig.Private to contain an ECC private key, and SlotConfig.IsSecret must be set to one, or else this command will fail. If the KeyConfig.KeyType does not indicate an ECC curve supported by this device, then this command will also return an error. Prior to the Configuration zone being locked, this command will always return an error.

Once the Data zone has been locked, the following additional restrictions are enforced:

- Private Key Creation:
  - Bit 13 of the corresponding SlotConfig must be set to one.
  - If KeyConfig.ReqAuth is set to one, then a prior authorization using KeyConfig.AuthKey must have been performed.
- Public Key Generation:
  - KeyConfig.GenPub must be set to one.
  - If KeyConfig.ReqAuth is set to one, then a prior authorization using KeyConfig.AuthKey must have been performed.

The following applies to all private key creation operations regardless of whether or not the Data zone has been locked:

- This command writes only those bytes necessary to create a private key of the type specified. The remaining bytes within the slot are unaffected by this command.
- When creating and writing a random key into the Data zone, the GenKey command always returns the public key regardless of the value of the GenPub bit within the KeyConfig area.
- If the corresponding SlotLocked bit is zero, then this command returns an error.
- There is a small statistical probability that the generated key will be unacceptable, in which case this command will return a single byte containing the ECC fault code (see Table 10-3, Status/Error Codes in Four byte Groups). In this circumstance the command should be re-run and will usually generate a key correctly in the subsequent iteration.

Table 11-13. Input Parameters

	Name	Size	Notes
Opcode	GenKey	1	`0x40`
Param1	Mode	1	See below.
Param2	KeyID	2	
Data	OtherData	3	If KeyID points to a public key, then these bytes replace Param1 and Param2 in the message calculation.

Table 11-14. Output Parameter

Name	Size	Notes
Response	1 or 64	Public Key X and Y coordinates. ECC fault code if generated private key was unacceptable.

Table 11-15. Mode Encoding

Bits	Meaning
0 – 1	Must be zero.
2	`1`: A random private key is generated and stored in the Slot specified by KeyID. KeyType must indicate an ECC key in the KeyConfig area for this KeyID or an error will be returned. `0`: A the private key currently stored in the slot is used to generate the public key.
3	`1`: The device creates a PubKey digest based on the private key in KeyID and places it in TempKey. `0`: No PubKey digest is created.
4	`1`: KeyID must point to a public key, and GenKey only creates the digest in TempKey without any public key generation operation. Bit 2 and bit 3 of the mode byte are ignored if this bit is set. `0`: KeyID points to a private key, and mode:2 and mode:3 control device operation.
5 – 7	Must be zero.

When a PubKey digest is to be calculated by the `GenKey` command, the following message is used as the input to the SHA-256 algorithm:

32 bytes	TempKey
1 byte	Opcode
1 byte	Param1
2 bytes	Param2
1 byte	SN[8]
2 bytes	SN[0:1]
25 bytes	Zeros
64 bytes	X and Y coordinates of the public key

11.8 `Info` Command

`Info` command accesses some static or dynamic four byte information from the device depending upon the value of mode. Illegal values of the mode parameter will result in a parse error response.

Table 11-16. Mode Encoding

Param1	Mode	Notes
0	Revision	A single 4-byte word representing the revision number of the device is returned. Software should not depend on this value as it may change from time to time.
1	KeyValid	Returns a value of one if an ECC private or public key stored in the key slot specified by param is valid and zero if the key is not valid. For public keys in slots where PubInfo is zero, the information returned by this command is not useful. This information is not meaningful for slots in which KeyType does not indicate a supported ECC curve.
2	State	Returns various dynamic state information as follows: First byte on the bus: Bits 0 – 3 TempKey.KeyId Bit 4 TempKey.SourceFlag Bit 5 TempKey.GenDigData Bit 6 TempKey.GenKeyData Bit 7 TempKey.NoMacFlag Second byte on the bus: Bit 0 0 Bit 1 0 Bit 2 AuthValid: A valid authorization sequence has been performed. Bit 3 – 6 AuthKey: The slot ID on which an authorization was performed. Bit 7 TempKey.Valid The third and fourth bytes on the bus are all zeros.
3	GPIO	Accesses the GPIO pin when the device is in either of the Single-Wire Interface modes. The specific operation is controlled by Param2 as follows: Bit 0 State to which output is to be driven. Ignored if bit 1 is zero. Bit 1 Driver state; Input (0) or Output (1). Bits 2-15 Must be zero. Always return the current state in the first byte followed by three bytes of `0x00`.
4	Volatile Key Permit	If Param2:Bit 1 is 0, then return the current state of the persistent latch. If Param2:Bit 1 is 1 and Config.VolKeyPermit:Bit 7 is set, then write Param2:Bit 0 to the persistent latch only if a valid authorization operation has been previously executed using the key specified at Config.VolKeyPermit:Bits 0-3

Table 11-17. Input Parameters

	Name	Size	Notes
Opcode	INFO	1	`0x30`
Param1	Mode	1	See above.
Param2	Param	2	Use depends on mode.
Data	—	0	Ignored.

Table 11-18. Output Parameters

Name	Size	Notes
Response	4	The information specified by mode or an error code.

Further information on the GPIO mode is as follows:

- If the IO_Mode field within Config. I2C_Address is set to disabled, or if Config:I2C_Enable:Bit 0 is set to I^{2}C mode, then the GPIO mode always returns an error code to the system firmware.
- If the IO_Mode field within Config.I2C_Address is set to Authorization modes, then the operation depends on I2C_Address:Bit 3. If this bit is zero, then the device is in Authorization Output mode. If one, then the device is in Intrusion Detection mode.
  - **Authorization Output Mode**:
    Regardless of the state of Param2:Bit1, on a successful execution the `Info` command returns the current state of the output pin. If Param2:1 indicates output and Param2:0 matches the default output state (I2C_Address:2), then set the output to the default; otherwise, if AuthValid is one and AuthKey matches SignalKey, then set the output to the opposite of the default state.
  - **Intrusion Detection Mode**:
    Regardless of the state of Param2:Bit1 on a successful execution the `Info` command returns the current state of the Persistent latch. If Param2:1 indicates output, and if AuthValid is one and AuthKey matches SignalKey, then set the Persistent latch to Param2:Bit 0.
- If the IO_Mode field within Config.I2C_Address is set to Input, then the current state of the GPIO pin is returned to the system firmware without changing the direction of the GPIO pin. This command will return an error if Param2:Bit 1 (driver state) is set to one (output).
- If the IO_Mode field within Config.I2C_Address is set to Output, then the direction of the GPIO driver will be set to match Param2:Bit 1 to zero for input and one for output. If configured as an Output, then the value in Param2:Bit 0 will be driven to the pin. Regardless of the value in Param2, the current state of the GPIO pin will be returned to the system in the output response parameter.

11.9 KDF Command

The KDF command implements one of a number of Key Derivation Functions (KDF). Generally this function combines a source key with an input string and creates a result key/digest/array. Three algorithms are currently supported: PRF, HKDF and AES.

PRF: The KDF specified in TLS 1.2 and earlier used for session establishment. The chip supports multiple variations including the methods used for master secret generation, session validation (finished messages) and key material generation (including AEAD suites)

AES: As used in some protocols. Calculates AES on a single block of input data, result is the key. Upper 16 bytes of the output are padded to 0 so the result is always 32 bytes.

HKDF: The KDF currently specified for TLS 1.3.

The input key may be either TempKey, the Alternate Key Buffer or an EEPROM slot. In either case, the length of the input storage location may be 32 or 64 bytes, though the actual KDF mode may also use only 16 or 48 for the cryptographic key depending on the algorithm.

The output result of the KDF, which could be 32 or 64 bytes, can be returned to the system in the output buffer, written to TempKey or the Alternate Key Buffer, or stored in an EEPROM slot. A 32 byte KDF result can be written to the upper 32 byte area of TempKey only if the lower 32 byte area is already valid.

If the output is returned to the system in the output buffer, then it may be encrypted depending on the mode parameter. If the PRF_Prot field in Config.Chip_Options is ENCRYPT, then the output will be encrypted regardless of the input mode. If the PRF_Prot field in Config.Chip_Options is STORE, then the output must go to either TempKey or the EEPROM. If an output is encrypted, it is always followed by a 32 bytes nonce.

The target can be written to the EEPROM only if KeyConfig[KeyID].type is data and KeyConfig[KeyID].PubInfo is 1 and SlotConfig[KeyID].WriteConfig is “Always”.

Param1 and Param2 control the source and destination of the KDF operation and are common for all three methods. Since there are different computation methods for each method, there is a third Param3 (Details) which drives the inner computation phase. Following the Details parameter is any data necessary for the calculation.

Table 11-19. Input Parameters

	Name	Size	Notes
Opcode	KDF	1	0x56
Param1	Mode	1	Controls the key source and target location for the KDF function.
Param2	KeyID	2	Slot locations in EEPROM if source or target is in the EEPROM
Param3	Details	4	Further information about the computation, depending on the algorithm. See below.
Data	Message	<128	Input value from system

Table 11-20. Output Parameter

Name	Size	Notes
OutData	1, 32, 64	The output of the KDF function or a success code if the result remains within the device.
OutNonce	0, 32	If the output is encrypted, a 32 byte random nonce generated by ATECC608A

Table 11-21. Mode Encoding

Bits	Meaning
0 – 1	Source for the KDF input key: 0: Get the KDF key from TempKey, length will be 32 or 64 bytes 1: Get the KDF key from the upper block of TempKey, length will be 32 bytes 2: Get the KDF key from an EEPROM slot. Use the LS byte of the KeyID input parameter 3: Get the KDF key from the Alternate Key Buffer.
2 – 4	Target for the KDF result: 0: Write the KDF result to TempKey 1: Write the result to the upper block of TempKey, do not affect the lower block 2: Write the KDF result to an EEPROM slot. Use the MS byte of the KeyID input parameter 3: Write the result to the Alternate Key Buffer 4: Place the KDF result in the output buffer. It will be in the clear if Config.ChipOptions permits 5: Place the KDF result in the output buffer encrypted with the IO protection key
5-6	Algorithm: 0 is the TLS PRF. 1 is AES. 2 is HKDF. 3 is reserved.
7	Must be 0

Table 11-22. Detail Parameter Encoding for PRF

Bits	Name	Description
0 – 1	KeyLen	The length of the source key in 16 byte blocks. 0 = 16 bytes, 1 = 32 bytes, 2 = 48 bytes, 3 = 64 bytes
2 – 7	Zero	Must be zero
8	TargetLen	The number of 32 byte blocks to be placed in the target location. 0 = 32 bytes, 1 = 64 bytes
9-10	Aead	If 00, no special AEAD processing. If 10, generate 96 bytes, shift down and ignore first 32 bytes. Result is (TargetLen+1) * 32 of the bytes that remain If 11, generate 96 bytes, shift down 32 bytes. First 32 remaining go to target, second 32 remaining go to output, never encrypted. Do not use this mode if output destination is the output buffer, use 10 instead.
11-23	Zero	Must be zero
24-31	DataLen	Length in bytes of the input parameter (Label | Seed in TLS lingo)

Table 11-23. Detail Parameter Encoding for AES

Bits	Name	Description
0 – 1	KeyLoc	The AES key is located at Src[KeyLoc*16] within the source key material. If the source key is only 32 bytes then KeyLoc must be either 0 or 1.
2 – 31	Zero	Must be zero

The HKDF function within the KDF command is intended to support the necessary key derivation operations as specified in TLS 1.3 and other protocols.

If the special IV function is indicated in the MsgLoc field of the Detail parameter, then the message in the input buffer will be checked to ensure that it contains the appropriate string to be considered an IV calculation. If the two bytes starting at index config.kdfIvLoc match the two stored at config.KdfIvStr, this mode allows the output restrictions to be ignored and the result to be returned in the clear. Also see Section 2.2.7.

It is possible to specify that all three of the input key, the message and the output key point to TempKey. This combination is forbidden and the results may be unpredictable.

Table 11-24. Detail Parameter Encoding for HKDF

Bits	Name	Description
0 – 1	MsgLoc	The location of the message. 0 = EEPROM slot, 1 = TempKey, 2 = Input Parameter, 3 = Special Initialization Vector function
2	ZeroKey	If 1, the key is 32 bytes of 0
3– 7	Zero	Must be zero
8-11	msgKey	The key slot for the message if in EEPROM
12-24	Zero	Must be zero
25-31	DataLen	Length in bytes of the HKDF message. If this value is zero, then the message will be 32 bytes of 0.

For all algorithms, output buffer encryption is implemented in a manner similar to the ECDH command, as below. If the IO protection function is not enabled (config.chipOption[bit 1]) then ATECC608A will return an error if output encryption is indicated in the mode parameter.

1. The first 32 bytes of the IO Protection key slot (config.chipOption[bits 12-15]) are copied to a SHA256 buffer.
2. The internal RNG generates a 32 byte random number and appends the first 16 bytes of that nonce to the SHA256 buffer
3. The SHA256 buffer is hashed and the digest is XOR'd with the first 32 bytes of the cleartext KDF result. If there are only 16 bytes in the result, then the output buffer will contain only those 16 bytes and the second 16 bytes of the SHA digest will be ignored.
4. If there are more than 32 bytes in the output, then a new digest is created via the SHA256 hash of the IO protection key (32 bytes) followed by the second 16 bytes of the random nonce from step #2. The resulting digest is XOR'd with the next 32 bytes of the result.

5. The output buffer consists of the encrypted KDF result followed by the 32 byte nonce. All 32 bytes of the nonce are output even if only the first 16 have been used.

11.10 Lock Command

The Lock command prevents future modifications of the Configuration and/or Data and OTP zones. If the device is so configured, then this command can be used to lock individual data slots. This command fails if the designated area is already locked.

Prior to locking the configuration and/or Data and OTP zones, the ATECC608A can optionally use the CRC-16 algorithm to verify the contents of the designated zone(s). The calculation uses the same algorithm as the CRC computed over the input and output groups. This summary digest (CRC) is always ignored when locking an individual slot.

- **Configuration Zone**:
  The CRC is calculated over all 128 bytes within the Configuration zone using the current value of the LockConfig at address 87. If the compare succeeds, then LockConfig will be set to a value of 00.
- **Data and OTP Zone**:
  The slot contents are concatenated in numerical order to create the input to the CRC algorithm. Slots that are configured to contain an ECC private key are never included in the summary CRC calculation. The OTP zone is then concatenated after the last Data slot and the CRC value is calculated. If the compare succeeds, then LockValue will be set to a value of 00.

If Mode:7 is zero and the input summary does not match that computed on the device, then an error is returned and the personalization process should be repeated.

For slots containing public keys that must be validated, the most significant four bits are modified by the device when being written and/or when being validated. The summary CRC is calculated using the current values.

Table 11-25. Input Parameters

	Name	Size	Notes
Opcode	LOCK	1	0x17
Param1	Mode	1	See Table 11-27.
Param2	Summary	2	Summary (CRC) of the designated zones, or should be 0x0000 if mode:7 is set.
Data	Ignored	0	—

Table 11-26. Output Parameter

Name	Size	Notes
Success	1	Upon successful execution, ATECC608A returns a value of zero.

Table 11-27. Mode Encoding

Bits	Meaning
0 – 1	0: The Configuration zone is to be locked. 1: The Data and OTP zones are to be locked. 2: A single slot in the Data zone is to be locked. 3: Illegal value, the device will return an error.
2 – 5	The slot number to be locked if bits0:1 have a value of two; otherwise, these bits must be zero.
6	Unused, must be zero.
7	Summary check bit. This bit is ignored when locking individual data slots. 0: The summary value is verified before the zone is locked.

Bits	Meaning
	`1`: Check of the zone summary is ignored and the zone is locked regardless of the contents of the zone. Microchip does not recommend using this mode.

11.11 `MAC` Command

The `MAC` command computes a SHA-256 digest of a key stored in the device, a challenge, and other information on the device. The output of this command is the digest of this message. If the message includes the serial number of the device, the response is said to be “diversified”.

The normal command flow to use this command is as follows:

1. Run `Nonce` command to load input challenge and optionally combine it with a generated random number. The result of this operation is a nonce stored internally on the device.
2. Optionally, run `GenDig` command to combine one or more stored EEPROM locations in the device with the nonce. The result is stored internally in the device. This capability permits two or more keys to be used as part of the response generation.
3. Run this `MAC` command to combine the output of step one (and step two if desired) with an EEPROM key to generate an output response (i.e. digest).

Alternatively, data in any slot (which does not have to necessarily even be secret) can be accumulated into the response through the same GenDig mechanism. This has the effect of authenticating the value stored in that location.

Table 11-28. Input Parameters

	Name	Size	Notes
Opcode	MAC	1	`0x08`
Param1	Mode	1	Controls which fields within the device are used in the message.
Param2	KeyID	2	The internal key is to be used to generate the response. Bits 0:3 only are used to select a slot; however, all 16 bits are used in the SHA-256 message.
Data	Challenge	0 or 32	Input portion of message to be digested, ignored if Mode:0 is one.

Table 11-29. Output Parameter

Name	Size	Notes
Response	32	SHA-256 digest

The message that will be hashed with the SHA-256 algorithm consists of the following information:

32 bytes	key[KeyID] or TempKey	(see Mode Encoding)
32 bytes	Challenge or TempKey	(see Mode Encoding)
1 byte	Opcode	(always `0x08`)
1 byte	Mode	
2 bytes	Param2	
8 bytes	Zeros	
3 bytes	Zeros	
1 byte	SN[8]	(*never* zeroed out)
4 bytes	SN[4:7]	(or zeros, see Mode Encoding)

2 bytes	SN[0:1]	(*never* zeroed out)
2 bytes	SN[2:3]	(or zeros, see Mode Encoding)

Table 11-30. Mode Encoding

Bits	Meaning
0	0: The second 32 bytes of the SHA message are taken from the input challenge parameter. 1: The second 32 bytes are filled with the value in TempKey. This mode is recommended for all use.
1	0: The first 32 bytes of the SHA message are loaded from one of the data slots. 1: The first 32 bytes are filled with TempKey
2	If either Mode:0 or Mode:1 are set, Mode:2 must match the value in TempKey.SourceFlag or the command will return an error.
3 – 5	Must be zero.
6	1: Include the 48 bits SN[2:3] and SN[4:7] in the message; otherwise, the corresponding message bits are set to zero.
7	Must be zero.

11.12 `Nonce` Command

The `Nonce` command generates a nonce for use by a subsequent command by combining an internally generated random number with an input value from the system. The resulting nonce is stored internally in TempKey and the generated random number is returned to the system.

The input value is designed to prevent replay attacks against the host, and it must be externally generated by the system and passed into the device using this command. It may be any value that changes consistently, such as a nonvolatile counter, current real time of day, and so forth, or it can be an externally generated random number.

To provide a nonce value for subsequent crypto commands, the input number and output random number are hashed together according to the information listed below. The resulting digest (i.e. nonce) is always stored in the TempKey register, TempKey.Valid is set, and TempKey.SourceFlag is set to *Rand*. The nonce can then be used by a subsequent ATECC608A command. Where the actual nonce value is required to be known by an external system, software will typically be needed to externally compute this digest value and store it externally to complete the execution of those commands.

In order to simplify the system code for some usage models, the device provides a mechanism for a host device to compute the nonce generated on a client device. In this calculation mode, the current value in TempKey is combined with the input parameters using SHA and the result is written back into TempKey. The new TempKey value is also returned to the system as the output parameter. Mode:0-1 must have a value of zero or one to enable this feature. TempKey.SourceFlag is not modified by the device in this mode.

Alternatively, this command can also be run in a pass-through mode if a fixed nonce is required for subsequent commands. In this case, the input value (either 32 or 64 bytes) is written directly to TempKey, the Alternate Key Buffer or the Message Digest Buffer without modification. No SHA-256 calculation is performed and TempKey.SourceFlag is set to Input (if writing to TempKey). If operated in this mode and with a repeated input number value, the device provides no protection against replay attacks.

Prior to the Configuration zone being locked, the RNG produces a value of `0xFF FF 00 00 FF FF 00 00` to facilitate testing. This test value is combined with the input value in the manner described above.

Table 11-31. Input Parameters

	Name	Size	Notes
Opcode	NONCE	1	`0x16`
Param1	Mode	1	Controls the mechanism of the internal RNG or fixed write as below.
Param2	Zero	2	Bits 0-14: Must be zero. Bit 15: `1` = RandOut is replaced by TempKey in both the hash calculation input (message) and the command output parameter. `0` = OutData is either the output of the RNG or a single byte of zero.
Data	NumIn	20, 32, 64	Input value from system.

Table 11-32. Output Parameter

Name	Size	Notes
OutData	1 or 32	The output of the RNG, calculated nonce or a single byte with a value of zero if Mode[0:1] is three.

If Mode[0:1] is zero or one and Param2:15 is zero, then the input NumIn parameter must be 20 bytes long and the SHA-256 message body used to create the nonce stored internally in TempKey consists of the following. Upon completion of the command, TempKey.SourceFlag is set to Rand.

32 bytes	RandOut from Random Number Generator	
20 bytes	NumIn from input stream	
1 byte	Opcode	(always `0x16`)
1 byte	Mode	
1 byte	LSB of Param2	(should always be `0x00`)

If Mode[0:1] is zero or one and Param2:15 is one, then the input NumIn parameter must be 20 bytes long and the SHA-256 message body used to create the nonce stored internally in TempKey consists of the following. TempKey must be valid prior to execution of this command and the values of the remaining TempKey flags remain unchanged.

32 bytes	TempKey	
20 bytes	NumIn from input stream	
1 byte	Opcode	(always `0x16`)
1 byte	Mode	
1 byte	LSB of Param2	(should always be `0x00`)

If Mode[0:1] is three, then this command operates in pass-through mode and TempKey is loaded with NumIn. Mode[5] determines whether 32 or 64 bytes are written to TempKey. No SHA-256 calculation is performed, no data is returned to the system, and TempKey.SourceFlag is set to Input.

Table 11-33. Mode Encoding

Bits	Meaning
0 – 1	`0 or 1`: Combine new random number with NumIn, store in TempKey. Mode[5-7] are ignored. `2`: Invalid. `3`: Operate in pass-through mode and write target with NumIn.
2 – 4	Must be zero.
5	If 0, then 32 bytes are written to the target.. If 1, then 64 bytes from the input will be written to the target which must be either TempKey or the Message Digest Buffer. This bit is ignored unless Mode[0:1] is Pass-Through
6-7	Target for the write. If 00, target is TempKey. If 01, target is the Message Digest Buffer. If 10, target is the Alternate Key buffer. This field is ignored unless Mode[0:1] is Pass-Through

11.13 `PrivWrite` Command

The `PrivWrite` command is used to write externally generated ECC private keys into the device.

For best security, Microchip recommends that the `PrivWrite` command not be used, and that private keys be internally generated from the RNG using the `GenKey(Create)` command.

The slot indicated by this command must be configured via KeyConfig.Private to contain an ECC private key, and SlotConfig.IsSecret must be set to one, or else this command will return an error. If the slot is individually locked using SlotLocked, then this command will also return an error.

The private key data is always sent to the device as a 36 byte integer. It is passed to the device MSB first. The first four bytes (32 bits) should be zero.

Prior to the Data zone being locked, this command can be used to write the slot contents without regards to the slotConfig value and/or the method by which TempKey was generated. The input data may or may not be encrypted based on the zone byte; if the input data is plain text then the MAC is ignored, but if it is encrypted then the MAC must be present and be properly computed. Prior to the Configuration zone being locked, this command will always return an error.

Once the Data zone is locked, the following is necessary for the write to complete:

- SlotConfig.IsSecret must be one.
- SlotConfig.WriteConfig must be set to *Encrypt* to indicate that writes require encryption. It is not possible to write to a slot for which WriteConfig is set to any other value.
- TempKey must be valid, its contents must have been generated using the `GenDig` command, and the KeyId used during the `GenDig` execution must match SlotConfig.WriteKey.
- Zone:6 must be set to indicate that the input data has been encrypted as follows:
  - The first 32 input bytes should be externally encrypted by XORing their value with the current value in TempKey. The next four bytes should be externally encrypted by XORing their value with the first four bytes of SHA-256(TempKey).
- An input authenticating MAC must be computed as follows:
  - SHA-256(TempKey, Opcode, Param1, Param2, SN[8], SN[0:1], <21 bytes of zeros>, 36 bytes of PlainTextData)

KeyConfig.ReqRandom, KeyConfig.ReqAuth and KeyConfig.AuthKey are ignored by this command because they will have been checked by the GenDig command for the parent encrypting key.

Table 11-34. Input Parameters

	Name	Size	Notes
Opcode	PRIVWRITE	1	`0x46`
Param1	Zone	1	Bits 0 – 5, 7: Must be zero. Bit 6: `1` = The input data is encrypted using TempKey. `0` = The input data is not encrypted: legal only when the Data zone is unlocked.
Param2	KeyID	2	Key slot to be written.
Data_1	Value	36	Information to be written to the slot may be encrypted. Must be 36 bytes long regardless of the size of the key.
Data_2	MAC	32	Message Authentication Code to validate EEPROM Write operation.

Table 11-35. Output Parameter

Name	Size	Notes
Success	1	Upon successful completion, ATECC608A returns a value of zero.

11.14 `Random` Command

The `Random` command generates a random number for use by the system.

Random numbers are generated via the internal NIST 800-90 A/B/C random number generator.

Prior to the Configuration zone being locked, the RNG produces a value of `0xFF`, `0xFF`, `0x00`, `0x00`, `0xFF`, `0xFF`, `0x00`, `0x00` to facilitate testing.

Table 11-36. Input Parameters

	Name	Size	Notes
Opcode	RANDOM	1	`0x1B`
Param1	Mode	1	Bit 0: Ignored by the device Bits 1-7: Must be zero
Param2	Zero	2	Must be `0x0000`.
Data	Ignored	0	—

Table 11-37. Output Parameter

Name	Size	Notes
RandOut	32	The output of the RNG.

11.15 Read Command

The Read command reads words (one four byte word or an 8-word block of 32 bytes) from one of the memory zones of the device. The data may optionally be encrypted before being returned to the system. See Section 10.5 for Data zone byte and word addressing information.

If reading from a slot in which SlotConfig.EncryptRead is set, the GenDig command must have been run prior to the execution of this command to generate the key that will be used for encryption. The key specified in SlotConfig.ReadKey must have been used in the GenDig calculation. The device encrypts data to be read by XORing each byte read from the EEPROM with the corresponding byte from TempKey. Encrypted reads of the configuration and/or OTP zones are not permitted.

KeyConfig.ReqRandom, KeyConfig.ReqAuth, and KeyConfig.AuthKey are ignored by this command because they will have been checked by the GenDig command for the parent encrypting key.

The byte addresses to be read should be divided by four (drop the least-significant two bits) before being passed to the device. If 32 bytes are being read, then the least-significant three bits of the input address are ignored. Addresses beyond the end of the specified zone result in an error.

The following restrictions apply to the three zones:

- **Configuration Zone**:
  The words within this zone are always readable using this command, regardless of the value of LockConfig.
- **OTP Zone**:
  If the OTP zone is unlocked this command returns an error. Once locked, the entire OTP zone may be read using 4 or 32 byte reads.
- **Data Zone**:
  If the Data zone is unlocked, this command returns an error; otherwise, the values within the corresponding SlotConfig word control access to the data slot. If SlotConfig.IsSecret is set and a four byte read is attempted, the device returns an error. If EncryptRead is set, this command encrypts the data as specified above. If IsSecret is set and EncryptRead is clear, this command returns an error. If IsSecret is clear and EncryptRead is clear, this command returns the desired slot in the clear.

Partial data blocks are always zero extended to 32 bytes before being encrypted.

Table 11-38. Input Parameters

	Name	Size	Notes
Opcode	READ	1	0x02
Param1	Zone	1	Bits 0 and 1: Select among Configuration, OTP, or Data. See Section 10.5. Bits 2-6: Must be zero. Bit 7: 1 = 32 bytes are read; otherwise four bytes are read.
Param2	Address	2	Address of first word to be read within the zone. See Section 10.5.
Data	—	0	—

Table 11-39. Output Parameter

Name	Size	Notes
Contents	4 or 32	The contents of the specified memory location.

If reading a Data zone and the EncryptRead bit is set in the corresponding SlotConfig word, the following actions are taken to encrypt the data:

- All of the TempKey register bits must be properly set as follows or else this command returns an error:

```
TempKey.Valid == 1
TempKey.GenDigData == 1
TempKey.KeyID == SlotConfig.ReadKey
TempKey.SourceFlag == “Rand”
```

- XOR the data from the memory zone with TempKey. Return as *Contents*.

11.16 `SecureBoot` Command

The SecureBoot command provides support for secure boot of an external MCU or MPU. The general approach is that the boot code within the system will use the device to assist in validating the application code that is to be subsequently executed.

- The digest of all the code to be executed is sent to the chip along with a signature. This digest/signature pair is verified with a public key stored in the device and a Boolean returned indicating success or failure.
- There is an option (mode == FullStore) to allow either the signature or verified digest to be stored in a slot within the chip to speed the boot time by limiting the IO and/or computation times. This mode is enabled by setting config.sboot.method to 010 or 011. To use this mode, the host should initially execute a special FullCopy validation mode where the device receives both the digest and the signature. If the signature is verified, then either the signature or digest is stored in the slot indicated by config.sboot.sigDig and the FullStore mode is then enabled.
- The slot to be written by FullCopy can be configured to require prior authorization to prevent certain kinds of denial of service attacks.

The return code can be MAC'd with a nonce from the host using the IO Protection secret to prevent tampering with the wire between the two devices.

If so configured by setting *SecureBootPersistentEnable* to one, the device can keep various keys disabled on power-up until a proper secure boot operation has completed, after which they will continue to be usable until the next time power is lost even if the device goes to sleep during the power cycle. This is accomplished via the PersistentDisable bit in KeyConfig and the Persistent Latch which maintains its state even when ATECC608A enters sleep mode.

If the device is configured to disable keys on power-up until a proper secure boot operation has completed but a successful secure boot has not yet completed during this power cycle, the SecureBoot command can be locked out until the next power cycle by setting bit 6 in the mode byte. This bit is ignored if a successful secure boot has *completed* or if *SecureBootPersistentEnable* is zero.

Table 11-40. Input Parameters

	Name	Size	Notes
Opcode	SecureBoot	1	`0x80`
Param1	Mode	1	Bits 0-2: See table below. All values not listed in the table are forbidden. Bits 3-5: Must be zero Bit 6: If 1, then the Secure Boot function may be prohibited until the next power cycle. Bit 7 : If 1, the input digest should be encrypted and a validating MAC will be appended to the output. If config.sboot.randNonce is set, then this bit is required for modes Validate, Full, FullCopy and FullStore. It must not be set for the remaining modes.
Param2	Param2	2	Must be zero
Data	Data	32, 96	Depends on mode: Full, FullCopy: 32 byte digest of entire code image followed by 64 byte signature to be verified by public key at config.sboot.pubKey. Code digest is encrypted if MAC is requested. FullStore: 32 byte digest of entire code image. Code digest is encrypted if MAC is requested.

Table 11-41. Output Parameters

Name	Size	Notes
ErrorCode	1	If 0, the comparison completed successfully and resulted in a match. If 1, the computation completed, but the value submitted was a mismatch. Other error codes indicate other issues with the command encoding, math computation, etc.
MAC	0 or 32	If an output MAC is indicated then MAC calculated as below if Match is 1 (validation succeeded). If the input mode parameter specifies that a mac is not required, then this parameter is zero length.

Table 11-42. Mode Encoding

Value	Name	Meaning
0-4	Reserved	Reserved, do not use these values
5	Full	Verify an input code digest of the entire code image using a signature in the input stream and a public key indicated by config.sboot.pubKey. A MAC may optionally replace the output on success.
6	FullStore	Verify an input code digest of the entire code image using a signature or digest stored in a key slot and a public key (if required) indicated by config.sboot.pubKey. A MAC may optionally replace the output on success.
7	FullCopy	Identical to Full, except that either the digest or signature is copied to the target slot on successful validation

If a MAC is allowed by the mode parameter and the MAC request bit is set, there are no parse or execution errors and the digest is accepted via either a signature verify or saved digest match, then neither the error code nor the match byte will be returned, instead the 32 byte MAC will be in the output parameters. The host software should make sure to check the count of returned bytes as the first byte of the MAC may be any value from 0x00 through 0xFF and this will not, by itself, indicate any status, as below.

If the mode indicates that a MAC is to be calculated, then the input code digest (aka message) will be encrypted by XORing with the SHA256 digest of the IO protection key concatenated with the nonce stored in TempKey. If config.SecureBoot.SecureBootRandNonce is set, then TempKey must have been generate by the Nonce command using the random mode. This IO Protection Key/TempKey digest is then re-used as part of the output MAC calculation as indicated below.

If a validation MAC is to be computed by the device in Full, FullCopy or FullStore(Sig) code modes, the Nonce command must have been run in advance to ensure that a nonce is available in TempKey. If the Config.SecureBoot.RandNonce bit is set, then the Nonce command must have been run in a mode that requires the inclusion of the ATECC608A RNG. The MAC will be computed as the SHA-256 digest of the following message:

32 bytes	Contents of the IO protection key
32 bytes	Nonce. First 32 bytes of TempKey

Compute SHA256 digest of above two elements, XOR digest with input encrypted code digest (message) to create plaintext code digest

32 bytes	Digest of IO Protection key and Nonce, as previous line
32 bytes	Plaintext Message. First 32 bytes of Input Buffer
64 byte	Signature as passed input buffer or from slot, per mode
4 bytes	Input parameters (Opcode, Mode, Param2)

If a validation MAC is to be computed by the device in FullStore(Dig), the Nonce command must have been run in advance to ensure that a nonce is available in TempKey. The MAC will be computed as the SHA-256 digest of the following message:

32 bytes	Contents of the IO protection key
32 bytes	Nonce. First 32 bytes of TempKey

Compute SHA256 digest of above two elements, XOR digest with input encrypted code digest (message) to create plaintext code digest

32 bytes	Digest of IO Protection key and Nonce, as previous line
32 bytes	Plaintext Message. First 32 bytes of input Buffer
4 bytes	Input parameters (Opcode, Mode, Param2)

11.17 `SelfTest` Command

Perform a test of one or more of the cryptographic engines within the ATECC608A chip. Some or all of the algorithms will be tested depending on the input mode parameter.

The config.chipOption.powerUpSelfTest bit can be set in the EEPROM in which case the SHA, AES and RNG tests will always be run on a power-up or wake from sleep. The chip cannot be configured to automatically run the ECC functions automatically on wake/power-up.

If any self test fails, whether called automatically on wake or via this command, the chip will enter a failure state in which chip operation is limited. The stored failure state is always cleared upon a wake or power cycle.

When in the failure state, the following operations are allowed:

- Reads of the configuration zone
- This self test command. If a particular test is re-run and passes on the subsequent attempt, that bit in the failure register will be cleared. If all bits are cleared, then ATECC608A resumes normal command operation.
- The current state of the failure register can be read by calling this self test command with a mode parameter of 0.
- Any other command, or reads of any other zone, will return an error code of 0x07. Use SelfTest(0) to determine the cause of the failure

Table 11-43. Input Parameters

	Name	Size	Notes
Opcode	SelfTest	1	`0x77`
Param1	Mode	1	Bit 0: If 1, test the RNG DRBG function Bit 1: If 1, test the ECDSA verify function Bit 2: If 1, test the ECDSA sign function Bit 3: If 1, test the ECDH function Bit 4: If 1, test the AES function – encrypt only Bit 5: If 1, test the SHA function Bits 6-7: Must be zero More than 1 bit may be set. To test all algorithms, the mode parameter should be 0x3F. If mode is 0, then return the current value of any stored failures this power cycle.
Param2	Param2	2	Must be zero.

Table 11-44. Output Parameter

Name	Size	Notes
Response	1	If 0, then all the requested tests were run without failure. If non-zero, then a bit map of the failing tests encoded in the same manner as the mode parameter where a 1 indicates that the test failed.

11.19 SHA Command

Computes a SHA-256 or HMAC/SHA digest for general purpose use by the host system.

The SHA computation is performed in a special section of internal ATECC508A memory that is not read or written by any other commands. Any arbitrary command can be interspersed between the various phases of the SHA command without problems. This stored SHA context is invalidated on power-up. In most cases, if an error occurs during the execution of the SHA command, the context is retained without change.

Calculation of a SHA/HMAC digest occurs as follows:

- **Start**:
  Initialization of the SHA-256 calculation engine and initialization of the SHA context in memory. This mode does not accept any message bytes.
- **HMAC_Start**:
  Initialization of the SHA context in memory and initial HMAC calculation using a stored 32-byte key. The key must be configured to be legal for use with MAC operations and is specified in the KeyID parameter. If KeyID is 0xFFFF, then the HMAC key will be taken from TempKey. This mode does not accept any message bytes. This mode does not support key lengths other than 32 bytes.
- **Update**:
  The command can be called a variable number of times with this mode to add bytes to the SHA or HMAC message. Each iteration of this mode must include a message of 1-64 bytes. There is a variation on Update which adds 64 bytes of a stored public key to the context.
- **End**:
  The SHA-256 or HMAC calculation is completed and the resulting digest is placed into the output buffer. The stored SHA context is invalidated on a successful "End" command. If so specified in the mode parameter, the digest and also be copied into either TempKey or the Message Digest buffer. From 0 through 64 bytes may be passed into the command as the final message bytes to be used before the digest is calculated.

  If the SHA context was started with HMAC_Start, then the HMAC calculation is completed using the key ID passed to the HMACstart phase and the resulting digest is placed into the output buffer. The key is re-loaded from the ECC608A memory during this calculation and the result will be incorrect if any changes to the key storage location occurred between the HMAC_Start and End phases.

The stored SHA context can be offloaded from the chip (Read_Context) and stored in host memory to permit multiple SHA digests to be calculated concurrently. The context stored in host memory can later be written back to the ATECC608A multiple times or just once. The context can be read after the Init or Update modes, but can no longer be read once the End phase has been executed. The host should NOT manipulate the stored context in any way or the results may be incorrect.

- **Read_Context**:
  Reads a variable length context from the ATECC608A while leaving the context valid within the chip. The total length of the output data parameter is always from 40 to 99 bytes and can be determined either from the length field in the output packet or computed as 40 plus the least significant six bits of the first byte in the output. This mode is not permitted for HMAC contexts, i.e. if the context was initialized with HMAC_Start or if a public key has been added to the context via the Public mode.

- **Write_Context**:
  Write a SHA256 context from the host to the ATECC608A to allow subsequent Update operations to be completed. This context should have previously been read from the chip with the Read_Context mode. ATECC608A determines the size of the context from the first 4 bytes of the data parameter, which should have been generated by the Read_Context command.

This command can also optionally generate a digest to be used by `Verify(ValidateExternal)` to validate a stored public key, which allows speed-up for future signature validations when X.509 format signatures are used. To implement this, a SHA(Public) iteration can be inserted prior to SHA(End) iteration, and the 64 bytes of the public key stored in the slot designated by Param2 will be added to the message. This mode will fail if the designated slot does not contain a public key or if slotConfig.noMac is set for the public key slot.

`Verify(ValidateExternal)` will only successfully validate the stored public key if the SHA(Public) iteration occurs at the at the block number indicated by config.publicPosition (X509format:0-3) within the sequence of `SHA(Update)` commands, and if the total number of blocks passed to the `SHA` command matches the value in config.TemplateLength (X509format:4-7). This restriction prevents a generation of non-standard X.509 templates, which may push the inserted public key into an unchecked area of the template.

For implementation reasons, this SHA command is limited to a total message length of 2^{28} bytes.

Table 11-45. Input Parameters

	Name	Size	Notes
Opcode	SHA	1	`0x47`
Param1	Mode	1	Bits 0-2: See table below. All values not listed in the table are forbidden. Bits 3-5: Must be zero Bits 6-7: Ignored except for “Finalize”, when it is interpreted as: 00 : Place resulting digest in both output buffer and TempKey 01 : Place resulting digest in both output buffer and Message Digest Buffer 10 : Do not use 11 : Place resulting digest in output buffer only
Param2	Length	2	KeySlot for the HMAC key if Mode is “HMAC_start” or Public Key if Mode is “Public”
Data	Message	0–64, 40-99	Up to 64 bytes of data to be included into the Update or Finalize mode operation. If “Write_Context”, up to 99 bytes should be provided.

Table 11-46. Output Parameter

Name	Size	Notes
Response	1 or 32 or 40-99	The SHA256 digest if mode is one of the finalize modes, otherwise zero for success or an error code. From 40 to 99 bytes if Read_Context

Table 11-47. SHA Mode Encoding

Name	Bits 0-2	Notes
Initialize	0	Initialize the internal SHA256 context memory. No message bytes are accepted (Length must be zero).
Update	1	Add 1-64 bytes in the message parameter to the current SHA context in memory
Finalize	2	Complete the SHA-256 or HMAC computation and place the digest into the output buffer. Optionally include 1-64 bytes in the context before the finalize.
Public	3	Add 64 bytes of a public key from an EEPROM slot into the context
HMAC_Init	4	Load the internal SHA context with the initialization value for HMAC(SHA-256), read the HMAC key and hash that into the context. Length field specifies the key to be used for the HMAC calculation, 0xFFFF means use TempKey. No message bytes are accepted (Length must be zero).
Read_Context	6	Read the current context out of the device to be stored in host memory.
Write_Context	7	Write (restore) a context into the ATECC508A context memory. Bytes written should have come from a Read_Context command output.

11.20 `Sign` Command

The `Sign` command generates a signature using the ECDSA algorithm. The ECC private key in the slot specified by KeyID is used to generate the signature.

The message may be externally or internally generated, as noted below:

- **External Message Generation (Mode:7 is `1`)**
  - The system should externally compile the information to be signed and compute the digest of that information using an external hash algorithm. This digest should then be loaded into TempKey or the Message Digest Buffer using the `Nonce` command.
  - External signatures must be enabled using SlotConfig.ReadKey:0 or else this command will return an error.
- **Internal Message Generation (Mode:7 is `0`)**
  - The message to be signed is internally generated. A typical use for this mode is to sign an internally generated random key.
  - The message is comprised of the output of the `GenDig` or `GenKey` commands (stored in TempKey), plus various other state information according to the description below. If TempKey is invalid or if it was not generated using `GenDig` or `GenKey`, then this command will return an error.
  - Internal signatures must be enabled using SlotConfig.ReadKey:1 or this command will return an error.
  - If the Data zone has not been locked, then internal signatures will always generate an error code.

The slot indicated by this command must be configured via KeyConfig.Private to contain an ECC private key, and SlotConfig.IsSecret must be set to one or else this command will fail.

The chip ignores KeyConfig.ReqRandom when executing this command. If the mode indicates that the message was internally generated, the GenDig or GenKey commands will have interrogated the corresponding ReqRandom bit and returned an error if TempKey as input to those commands was not derived from the RNG source.

There is a small statistical probability that the device will fail to properly generate the ephemeral key, in which case this command will return a single byte containing the ECC fault code (see Table 10-3, Status/Error Codes in Four byte Groups). The command will generally succeed if resubmitted.

Table 11-48. Input Parameters

	Name	Size	Notes
Opcode	Sign	1	`0x41`
Param1	Mode	1	See Below.
Param2	KeyID	2	The internally-stored private key to be used to generate the signature. The curve specified in KeyConfig[KeyID].KeyType will be used.

Table 11-49. Output Parameter

Name	Size	Notes
Response	64	The signature composed of R and S, or an error code.

Table 11-50. Mode Encoding

Bits	Meaning
0	Set to `0` if the resulting signature is intended to be used by `Verify(Validate)` or one if it is to be used by `Verify(Invalidate)`. In all other situations, this bit must be zero.
1 – 4	Must be `0`.
5-7[1]	00x: Generate an internal message to be signed from the current value in TempKey in addition to key slot configuration information as below. Set the SN bits to 0 01x: Generate an internal message as below and include the stored 48 bit serial number (SN[2:3] and SN[4:7]) 1x0: The message to be signed is in TempKey 1x1: The message to be signed is in the Message Digest Buffer

Note 1: The 'x' in this field indicates that the mode bit can be either 0 or 1

Internal signatures are generated over a 55 byte message consisting of the information listed below. The first element of the message is the public key digest information placed in TempKey by `GenKey` or `GenDig`. The message also includes configuration information regarding the key used for the TempKey calculation. This 55 byte message is created as follows:

> If multiple `GenKey` or `GenDig` commands have been run between the `Nonce` and `Sign` commands, only the configuration for the last key used will be signed.

The bit within the SlotLocked field corresponding to the last key used in the TempKey computation is in the LSB of the byte listed below, regardless of whether or not the slot is individually lockable or not.

If the slot contains a public key corresponding to a supported curve, and if PubInfo indicates this key must be validated before being used by `Verify`, and if the validity bits have a value of `0x05`, then the PubKey Valid byte will be `0x01`. In all other cases, it will be `0`.

32 bytes	TempKey	(must have been generated by `GenKey` or `GenDig`)
1 byte	Opcode	
1 byte	Mode	
2 bytes	Param2	
2 bytes	SlotConfig[TempKeyFlags.keyId]	
2 bytes	KeyConfig[TempKeyFlags.keyId]	
1 byte	TempKeyFlags	(b0-3: keyId, b4: sourceFlag, b5: GenDigData,b6: GenKeyData, b7: NoMacFlag)
2 byte	Zeroes	
1 byte	SN[8]	(*never* zeroed out)
4 bytes	SN[4:7]	(or zeros, see Mode)
2 bytes	SN[0:1]	(*never* zeroed out)
2 bytes	SN[2:3]	(or zeros, see Mode)
1 byte	SlotLocked:TempKeyFlags.keyId	
1 byte	PubKey Valid	(or zero, see above)
1 byte	`0x00`	

This message is then hashed using the SHA-256 algorithm and passed to the ECDSA signature computation engine.

11.21 `UpdateExtra` Command

The `UpdateExtra` command is used to update the values of the two extra bytes within the Configuration zone (location 84 and 85) after the Configuration zone has been locked. For I^{2}C chips, Byte 85 bay be used as the I2C address, depending on the state of config.ChipMode.Bit0.

These bytes are essentially one-time write bytes: once they are written to a non-cero value they can no longer be written.

- If the mode parameter indicates UserExtra at address 84:
  - If the current value in UserExtra (byte 84 of Configuration zone) is zero, then `UpdateExtra` writes this byte with the LS byte of NewValue and returns success.
  - If the current value in UserExtra is non-zero, then the command returns an execution error.
- If the mode parameter indicates UserExtraAdd at address 85:
  - If the current value in UserExtraAdd (byte 85 of Configuration zone) is zero, then `UpdateExtra` writes this byte with the LS byte of NewValue and returns success.
  - If the current value in UserExtraAdd is non-zero, then the command returns an execution error.

Table 11-51. Input Parameters

	Name	Size	Notes
Opcode	UpdateExtra	1	`0x20`
Param1	Mode	1	Bit 0: `0` = Update Config byte 84. `1` = Update Config byte 85. Bits 1 – 7: Must be zero.
Param2	NewValue	2	LSB: Value to optionally be written to location 84 or 85 in Configuration zone. MSB: Must be `0x00`.
Data	—	0	

Table 11-52. Output Parameter

Name	Size	Notes
Success	1	If the memory byte was updated, this command returns a value of `0x00`; otherwise, it returns an execution error.

11.22 `Verify` Command

The `Verify` command takes an ECDSA [R,S] signature and verifies that it is correctly generated given an input message digest and public key. In all cases, the signature is an input to the command.

An optional MAC can be returned from the Verify command to defeat any man-in-the-middle attack. If the verify calculation shows that the signature is correctly generated from the input digest then a MAC will be computed based on an input nonce stored in TempKey and the value of the IO protection secret which is stored in both the ATECC608A and the host MCU. MAC outputs can only be generated in External and Stored mode. The IO Protection function must be enabled for MAC computation. See below for specific details on the MAC message.

The `Verify` command can operate in four different modes:

1. **External Mode**
   The public key to be used is an input to the command. Prior to this command being run, the message should be written to TempKey using the `Nonce` command. In this mode the device merely accelerates the public key computation and returns a Boolean result.
2. **Stored Mode**
   The public key to be used is found in the KeyID EEPROM slot. The message should have been previously stored in TempKey or the Message Digest Buffer. If the following configuration checks for the public key at KeyID succeed, the public key verification computation is performed and a Boolean result is returned to the system; otherwise, the command returns an execution error.
   - If KeyConfig[KeyID].PubInfo is one, then the key must have been previously validated using the `Verify` command.
   - If KeyConfig[KeyID].ReqAuth is set, then a previous key authorization must have been performed with either the `Verify` or `CheckKey` commands based on the key in KeyConfig.AuthKey.
   - If KeyConfig[KeyId].KeyType indicates an ECC curve not supported by the device or indicates “Not an ECC Key,” then this command will fail.
   - This mode is used to set the key authorization information when the KeyConfig.AuthKey field within some other slot points to KeyID. If the verification succeeds, then an internal authComplete flag will be set and KeyID retained. See Section 4.4.8, Authorized Keys.
   - Data1 and Data2 must be 32 bytes, and Data3 & Data4 should be zero length.
3. **Validate and Invalidate Mode**s
   The Validate and Invalidate modes are used to validate or invalidate the public key stored in the EEPROM at KeyID. The signature is input to the device and a partial message should be in TempKey. The verifying public key is found at SlotConfig[KeyID].ReadKey, and the ECDSA Verify message is composed of KeyID and TempKey, formatted as noted below. Only KeyIDs 8 – 15 can be validated; therefore, this command will return an error if KeyID is 0 – 7.
   - If the ECC verification passes, then the most significant four bits of the first byte of Block 0 of the public key at KeyID will be set to `0x5` for Validate and `0xA` for Invalidate. See Section 5.1.1, ECC Key Formatting.
   - Key (in)validation takes place regardless of the state of the LockValue byte and/or the SlotLocked bit corresponding to this slot.
   - If the X509format byte corresponding to the specified key is non-zero, then the `Verify` command will return an error.

   OtherData[17]:Bit0 represents the validity of the key being acted on. It must be a ’`0`‘(invalid) to permit the Validation operation, or a ‘`1`’ (valid) to permit the Invalidation operation. If OtherData[17]:Bit0 does not match Mode:2 in Validate/Invalidate modes, then this command will return an error.

   Data1 and Data2 sizes are the same as stored mode.

4. **ValidateExternal Mode**
   The ValidateExternal mode is used to validate the public key stored in the EEPROM at KeyID when X.509 format certificates are to be used. The digest of the message must be TempKey. TempKey must have been generated using the `SHA(Public)` command, and the key for that computation must be the same as KeyID. The verifying public key is found at SlotConfig[KeyID].ReadKey, and the ECDSA Verify message is composed of KeyID and TempKey, formatted as below. Only KeyIDs 8 to 15 can be validated, so this command will return an error if KeyID is 0 – 7.
   - If the ECC verification passes, then the most significant four bits of the first byte of Block 0 of the public key at KeyID will be set to `0x5`. See Section 4.4.3, Created ECC Keys.
   - Key validation takes place regardless of the state of the LockValue byte and/or the SlotLocked bit corresponding to this slot.
   - If the X509format byte corresponding to the specified key is zero, then the `Verify` command will return an error.
   - Data1 and Data2 sizes must be 32 bytes each. Data3 and Data4 should both be zero length.

   Regardless of the mode, if the public key is stored within an EEPROM slot and If KeyConfig.ReqRandom is set for that slot, then the value in TempKey must have been generated using the random number generator within ATECC608A. The Message Digest Buffer may not be used for the message in this case.

Table 11-53. Input Parameters

	Name	Size	Notes
Opcode	Verify	1	`0x45`
Param1	Mode	1	Mode[0:2] `000`: Stored Mode. `001`: ValidateExternal Mode. `010`: External Mode. `011`: Validate Mode. `111`: Invalidate Mode. `100-110`: Do not use Mode[3:4] Must be zero. Mode[5] : If 1, the message to be verified should come from the Message Digest Buffer, otherwise the message should be in TempKey. Ignored unless Mode[0:2] is 000 or 010. Mode[6] : Must be 0 Mode[7] : If 1, a validating MAC will be appended to the output. Ignored unless Mode[0:2] is 000 or 010.
Param2	KeyID	2	If Mode:0-2 is Stored Mode, KeyID contains the number of the slot containing the public key to be used for the verification. KeyConfig[KeyID].KeyType determines the curve to be used. If Mode:0-2 is ValidateExternal Mode, KeyID contains the number of the slot containing the public key to be validated which must have been specified by a previous SHA(Public) command. The parent key to be used to perform the validation is stored in SlotConfig[KeyID].ReadKey and KeyConfig[ParentKey]. KeyType determines the curve to be used. If Mode:0-2 is External Mode, KeyID contains the curve type to be used to Verify the signature. The value in this field is encoded identically to the KeyType field in the keyConfig words within the Configuration zone. If Mode:0-2 is Validate or Invalidate mode, KeyID contains the number of the slot containing the public key to be (in)validated. The parent key to be used to perform the (in)validation is stored in SlotConfig[KeyID] KeyProp.ValidateKey. SlotConfig[ParentKey].AlgoInfo determines the curve to be used.

	Name	Size	Notes
Data1	R	32	The R component of the ECDSA signature to be verified.
Data2	S	32	The S component of the ECDSA signature to be verified.
Data3	X	0 or 32	The X component of the public key to be used for verification if Mode:0-1 is External.
Data4	Y	0 or 32	The Y component of the public key to be used for verification if Mode:0-1 is External.
Data5	OtherData	0 or 19	If Validate mode, the bytes used to generate the message for the validation. X and Y should be zero length in this mode and this parameter comes immediately after S in the input parameter stream. Should be zero length for all other modes.

Table 11-54. Output Parameter

Name	Size	Notes
Response	1	Returns a value of zero if the signature of the message can be verified using the public key. Returns a value of one if the signature does not match, or another error code if there is some form of parsing or execution error.

If a validation MAC is to be computed by the device (i.e. Mode:7 is set), this will be computed as the SHA-256 digest of the following message:

32 bytes	Contents of the IO protection key
32 bytes	Message. From TempKey if Mode:5 is 0, else first 32 bytes of Message Digest Buffer
32 bytes	System nonce. First 32 bytes of Message Digest buffer if Mode:5 is 0, else second 32 bytes of Message Digest Buffer
64 byte	Signature passed to device as Data1/2
4 bytes	Input parameters (Opcode, Mode, Param2)

The message to be used for the ECDSA Verify operation depends on the mode as follows:

- **Stored, External Modes**
  Either TempKey or the Message Digest Buffer should contain the SHA-256 digest of the message, depending on Mode:5.
- **ValidateExternal Modes**
  The contents of TempKey should contain the SHA-256 digest of the message.
- **Validate or Invalidate Mode**
  The contents of TempKey should contain a digest of the PublicKey at KeyID. It must have been generated using the `GenKey` command over the KeyID slot. The device then generates a message based on the same format as the `Sign(Internal)` command, except that the parameter and state bytes are copied from the input parameter OtherData. The message is formatted as follows:

32 bytes	TempKey (must have been generated by `GenKey`)
1 byte	Sign Opcode
10 bytes	OtherData[0:9]
1 byte	SN[8]
4 bytes	OtherData[10:13]
2 bytes	SN[0:1]
5 bytes	OtherData[14:18]

This message is hashed using SHA-256 and used as the message input to the ECC `Verify` operation.

11.23 `Write` Command

The `Write` command writes either one four byte word or an 8-word block of 32 bytes to one of the EEPROM zones on the device. Depending upon the value of the WriteConfig byte for this slot, the data may be required to be encrypted by the system prior to being sent to the device. This command cannot be used to write slots configured as ECC private keys (see PrivWrite).

The following restrictions apply to writes within zones using this command:

- **Configuration Zone:** If the Configuration zone is locked or Zone:6 is set, then this command returns an error; otherwise the bytes are written as requested. Any attempt to write any byte for which writes are permanently prohibited (see Section 2.2, EEPROM Configuration Zone) results in a command error with no modifications to the EEPROM.
- **OTP Zone:** If the OTP zone is unlocked, then all bytes can be written with this command. If the OTP zone is locked writes to the OTP zone are never permitted.
- **Data Zone:** If the Data zone is unlocked, then all bytes in all zones can be written with either plain text or encrypted data. After the Data zone is locked, the values within the WriteConfig bytes control access to the data slots. If the WriteConfig bits for this slot are set to *Always*, then the input data should be passed to the device in the clear. If Bit:14 of SlotConfig is set to one, then the input data should be encrypted and an input MAC calculated. If the slot is individually locked using SlotLocked, then this command always returns an error.

Four byte writes are only permitted in the Data zone if all of the following conditions are met:

- SlotConfig.IsSecret must be zero.
- SlotConfig.WriteConfig must be always.
- The input data must not be encrypted, i.e. Zone:6 must be zero.
- The Data/OTP zones must be locked.

Four byte writes are never permitted in the OTP zone.

The two input address bytes are formed as described in Section 10.5. The least significant three bits, Address[0:2], indicate the word within the block, or they are ignored if an entire 32 byte block is being written. Address[3:6] contains the slot number for writes to the Data zone, or the block number for the Configuration and OTP zones. For the Data zones, Address[8:9] is used to indicate the block within the slot. Address values beyond the size of the specified zone result in the command returning an error.

For Slots 8 to 15, if KeyConfig.PubInfo indicates that the slot contains an ECC public key which can be validated, then the key will be invalidated by writing `0xA` to the most significant four bits of byte zero of Block 0 of the slot when any block within the slot is written. If KeyConfig.PubInfo is zero, then the most significant four bits of byte zero of Block 0 of the slot are written with the data from the input parameter.

If KeyConfig.PubInfo is one and the ECC public key has been validated, then writes will fail if WriteConfig is set to `0001` (PubInvalid). Use `Verify(Invalidate)` to invalidate the public key prior to writing.

Any attempt to write the OTP and/or Data zones prior to the Configuration zone being locked results in the device returning an error code. When writing to the Data zone, if the corresponding SlotLocked bit is zero, then this command returns an error regardless of whether or not the OTP/Data zones have been locked.

11.23.1 Input Data Encryption

The input data may be encrypted to prevent snooping on the bus during personalization or system operation. The system should encrypt the data by XORing the plain text with the current value in TempKey. Upon receipt, the device will XOR the input data with TempKey to restore the plain text prior to writing to the EEPROM.

Whenever the data is encrypted, an authorizing input MAC is always required. This MAC is computed as follows:

SHA-256(TempKey, Opcode, Param1, Param2, SN[8], SN[0:1], <25 bytes of zeros>, PlainTextData)

Prior to locking of the OTP/Data zones, Zone:6 is used to indicate to the device whether or not the input data is encrypted. After locking of the OTP/Data zones, Zone:6 is ignored and only bit 14 of the slotConfig corresponding to the slot being written is used to determine whether or not the input data is encrypted.

If data encryption is indicated, TempKey must be valid prior to this command being called, and it must be the result of `GenDig`. Specifically, this means that TempKey.Valid and TempKey.GenDigData must both be set to one. The last slot used by `GenDig` for TempKey creation and stored in TempKey.KeyID must match that in SlotConfig.WriteKey. Prior to data locking, any key can be used to generate TempKey and the GenDigData bit is ignored.

The KeyConfig.ReqRandom, KeyConfig.ReqAuth and KeyConfig.AuthKey are ignored by this command because they will have been checked by the `GenDig` command for the parent encrypting key.

When performing an encrypted Write to a partial block at the end of slots 0 thru 7 and 9 thru 15, all 32 bytes of input must be sent to the device, with the unused bits being used only as part of the MAC calculation. Their value will not affect the final contents of the EEPROM.

Table 11-55. Input Parameters

	Name	Size	Notes
Opcode	Write	1	`0x12`
Param1	Zone	1	Bits 0 – 1: Select among Config, OTP or Data. See Section 10.5.1, Zone Encoding. Bits 2 – 5: Must be zero. Bit 6: `1` = The input data has been encrypted; otherwise the input data is in the clear. Ignored after the Data zone is locked. Bit 7: `1` = 32 bytes will be written; otherwise four bytes are written.
Param2	Address	2	Address of first word to be written within the zone. See Section 10.5
Data_1	Value	4 or 32	Information to be written to the zone may be encrypted.
Data_2	MAC	0 or 32	Message authentication code to validate address and data.

Table 11-56. Output Parameter

Name	Size	Notes
Success	1	Upon successful completion, ATECC608A returns a value of zero.

12 Compatibility

12.1 Microchip ATECC508A

ATECC08A is designed to be fully compatible with the ATECC508A devices with the limited exception of the functions listed below. If the ATECC608A is properly configured, software written for the ATECC508A should work with the ATECC608A without any required changes, again with the exception of the functions listed below.

Most elements of the configuration zone in the ATECC608A are identical in both location and value with the ATECC508A. However the initial values that had been stored in the LastKeyUse field may need to be changed to conform to the new definition of those bytes which can be found in this document. That field contained the initial count for the Slot 15 limited use function which is supported in the ATECC608A via the monotonic counters.

New Features in ATECC608A vs. ATECC508A

- Secure boot function, with IO encryption and authentication
- KDF command, supporting PRF, HKDF, AES
- AES command, including encrypt/decrypt
- GFM calculation function for GCM AEAD mode of AES
- Updated NIST SP800-90 A/B/C Random Number Generator
- Flexible SHA/HMAC command with context save/restore
- SHA command execution time significantly reduced
- Volatile Key Permitting to prevent device transfer
- Transport Key Locking to protect programmed devices during delivery
- Counter Limit Match function
- Ephemeral key generation in SRAM, also supported with ECDH and KDF
- Verify command output can be validated with a MAC
- Encrypted output for ECDH
- Added self test command, optional automatic power-on self test
- Unaligned public key for built-in X.509 cert key validation
- Optional power reduction at increased execution time
- Programmable I2C address after data (secret) zone lock

Features eliminated in ATECC608A vs. ATECC508A

- HMAC command removed, replaced via new more powerful SHA command
- OTP consumption mode eliminated, now read only
- Pause command eliminated along with related Selector function in UpdateExtra
- Slot 15 special limited use eliminated, replaced with standard monotonic counter limited use
- SHA command no longer uses TempKey during calculation, result in TempKey is unchanged

12.2 Microchip ATSHA204A, ATECC108A

The ATECC608A is generally compatible with all ATSHA204/A and ATECC108/A devices. If properly configured, it can be used in most situations where these devices are currently employed. For ATSHA204A and ATECC108A compatibility restrictions, see the ATECC508A datasheet.

13 Mechanical

13.1 Pinouts

The device is offered in multiple packages: 8-lead SOIC, 8-pad UDFN, and 3-lead CONTACT (i.e. non-solder) package. The pinout is as follows:

Figure 13-1. Package Pinouts

Name	Pin
SDA	5
SCL	6
V_{CC}	8
GND	4
NC	1, 2, 3, 7

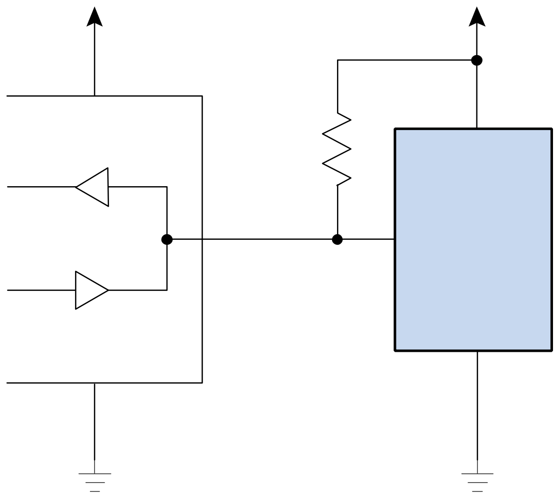
13.2 Wiring Configuration for Single-Wire Interface

Using the Single-Wire Interface allows the connection of ATECC608A to a host using only a single pin (SDA) to transfer data in both directions. This interface does not use the SCL pin, which should be tied to ground.

To prevent forward biasing the internal diode and drawing current across power planes in the system, the resistor pull-up on the SDA pin should either be connected to the same supply that is connected to the V_{CC} pin or to a lower voltage rail.

If the signal levels for SDA are different than the V_{CC} voltage, consult the parametric specifications section of this document to ensure that the signal levels are such that excessive leakage current will be minimized when in sleep modes. This situation might occur if the ATECC608A device is physically distant from the bus master device and the supply voltage for the bus master is different than the supply voltage for ATECC608A.

Figure 13-2. 3-wire Configuration for Single-Wire Interface



14 Ordering Information

Microchip Ordering Code[2]	Package	Delivery Information		Interface Configuration
		Form	Quantity	
ATECC608A-SSHCZ-T	8-lead SOIC	Tape and Reel	4,000 per Reel	Single-Wire
ATECC608A-SSHCZ-B	8-lead SOIC	Bulk in Tubes	100 per Tube	Single-Wire
ATECC608A-SSHDA-T	8-lead SOIC	Tape and Reel	4,000 per Reel	I^2C
ATECC608A-SSHDA-B	8-lead SOIC	Bulk in Tubes	100 per Tube	I^2C
ATECC608A-MAHCZ-T	8-pad UDFN	Tape and Reel	15,000 per Reel	Single-Wire
ATECC608A-MAHDA-T	8-pad UDFN	Tape and Reel	15,000 per Reel	I^2C
ATECC608A-MAHCZ-S	8-pad UDFN	Tape and Reel	3,000 per Reel	Single-Wire
ATECC608A-MAHDA-S	8-pad UDFN	Tape and Reel	3,000 per Reel	I^2C
ATECC608A-RBHCZ-T[1]	3-lead CONTACT	Tape and Reel	5,000 per Reel	Single-Wire

Notes 1. Please contact Microchip for availability.
2. Please contact Microchip for thinner packages.

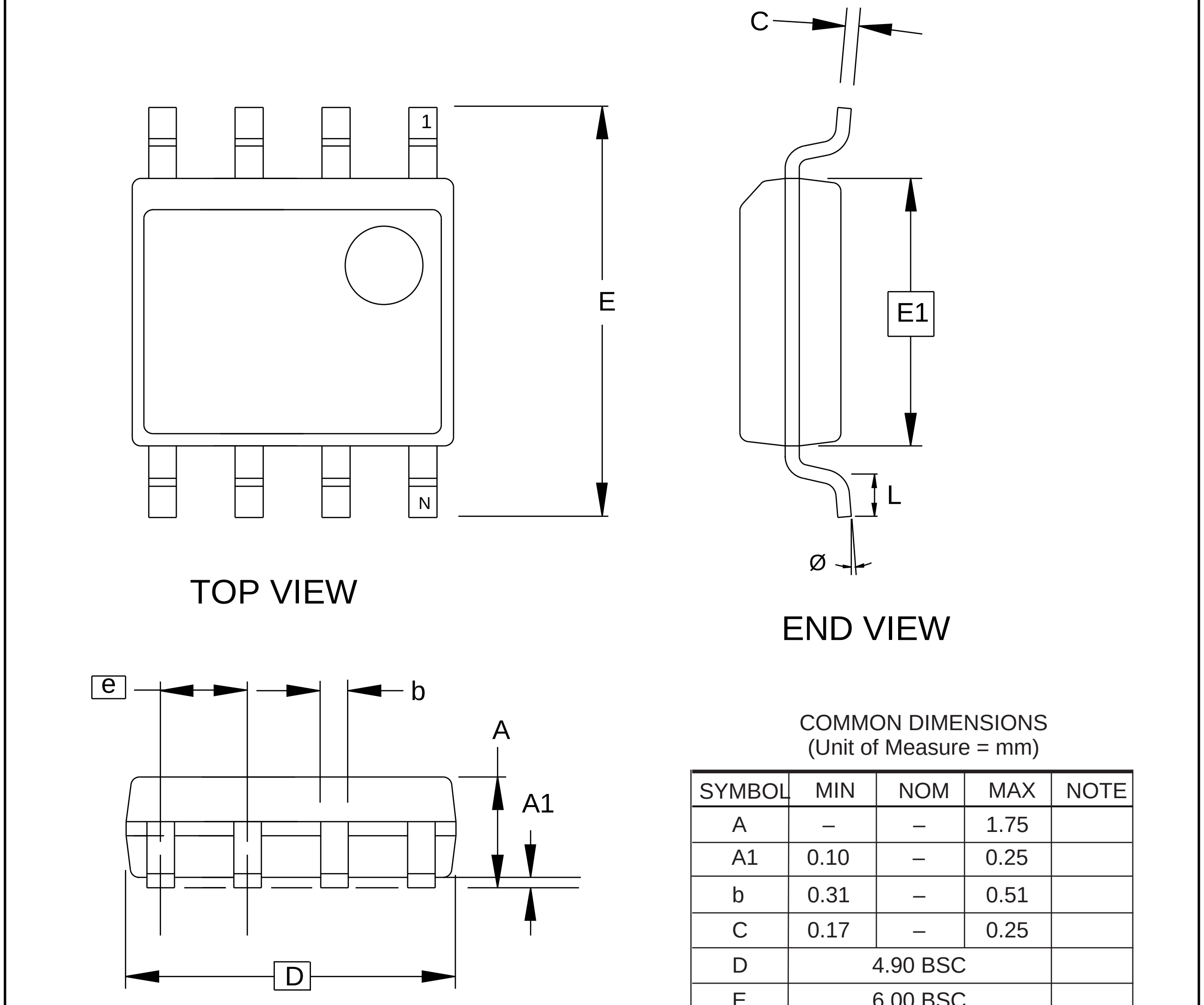
14.1 Part Marking

As part of Microchip's overall security features the part mark for all crypto devices is intentionally vague. The marking on the top of the package does not provide any information as to the actual device type or the manufacturer of the device. The alphanumeric code on the package provides manufacturing information and will vary with assembly lot.

15 Errata

16 Package Drawings

16.1 8-lead SOIC



COMMON DIMENSIONS
(Unit of Measure = mm)

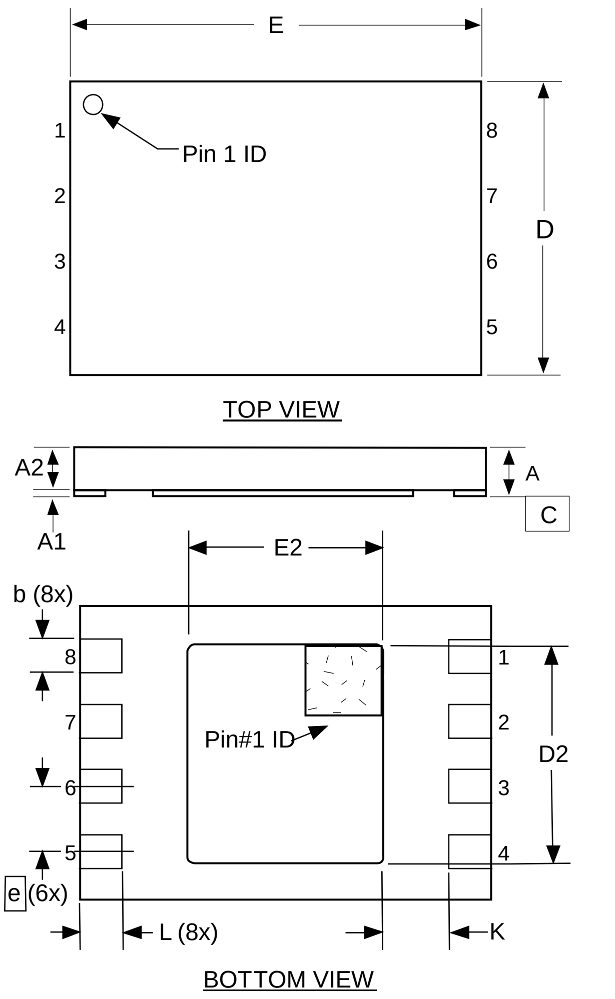
SYMBOL	MIN	NOM	MAX	NOTE
A	–	–	1.75	
A1	0.10	–	0.25	
b	0.31	–	0.51	
C	0.17	–	0.25	
D	4.90 BSC			
E	6.00 BSC			
E1	3.90 BSC			
e	1.27 BSC			
L	0.40	–	1.27	
Ø	0°	–	8°	

Notes: This drawing is for general information only. Refer to JEDEC Drawing MS-012, Variation AA for proper dimensions, tolerances, datums, etc.

3/6/2015

Atmel Package Drawing Contact: packagedrawings@atmel.com	TITLE 8S1, 8-lead (0.150” Wide Body), Plastic Gull Wing Small Outline (JEDEC SOIC)	GPC SWB	DRAWING NO. 8S1	REV. H

16.2 8-pad UDFN



Notes:
1. This drawing is for general information only. Refer to Drawing MO-229, for proper dimensions, tolerances, datums, etc.
2. The Pin #1 ID is a laser-marked feature on Top View.
3. Dimensions b applies to metallized terminal and is measured between 0.15 mm and 0.30 mm from the terminal tip. If the terminal has the optional radius on the other end of the terminal, the dimension should not be measured in that radius area.
4. The Pin #1 ID on the Bottom View is an orientation feature on the thermal pad.

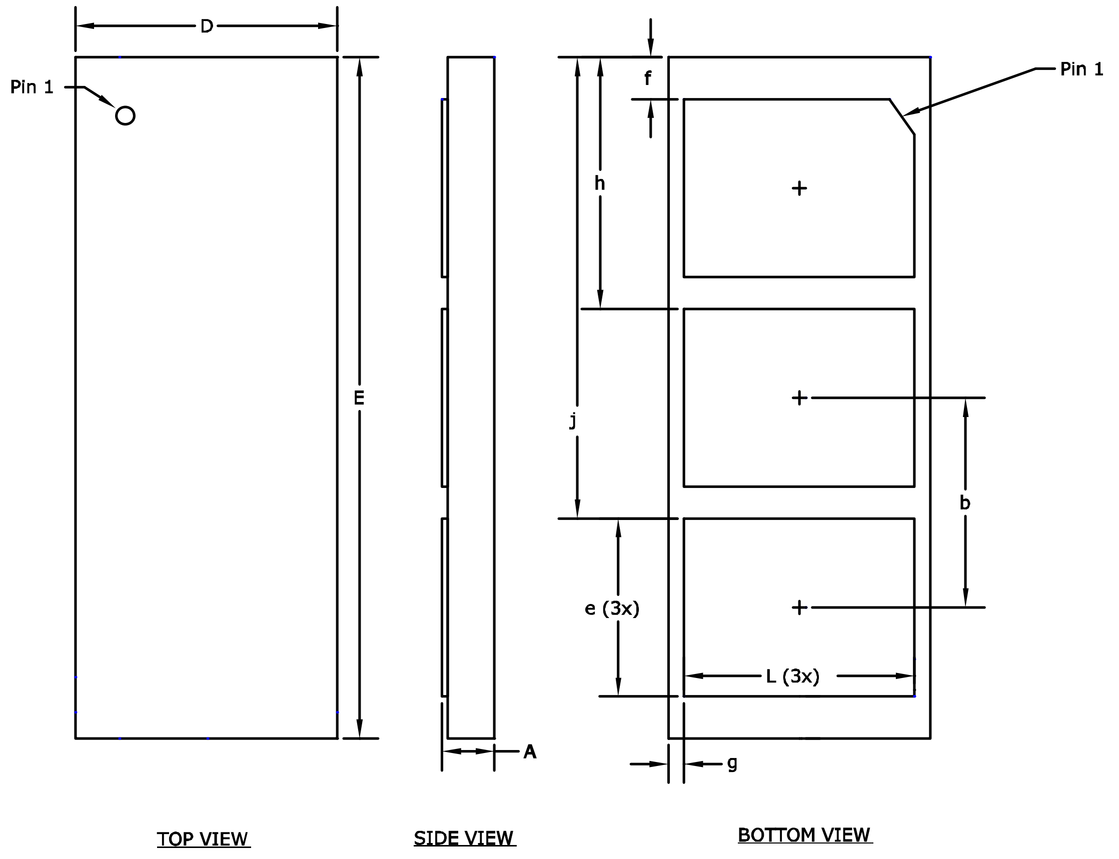
COMMON DIMENSIONS
(Unit of Measure = mm)

SYMBOL	MIN	NOM	MAX	NOTE
A	0.50	0.55	0.60	
A1	0.0	0.02	0.05	
A2	-	-	0.55	
D	1.90	2.00	2.10	
D2	1.40	1.50	1.60	
E	2.90	3.00	3.10	
E2	1.20	1.30	1.40	
b	0.18	0.25	0.30	3
C		0.152 REF		
L	0.35	0.40	0.45	
e		0.50 BSC		
K	0.20	-	-	

11/2/15

	TITLE	GPC	DRAWING NO.	REV.
Atmel Package Drawing Contact: packagedrawings@atmel.com	8MA2, 8-pad 2 x 3 x 0.6mm Body, Thermally Enhanced Plastic Ultra Thin Dual Flat No-Lead Package (UDFN)	YNZ	8MA2	H

16.3 3-lead CONTACT



COMMON DIMENSIONS
(Unit of Measure = mm)

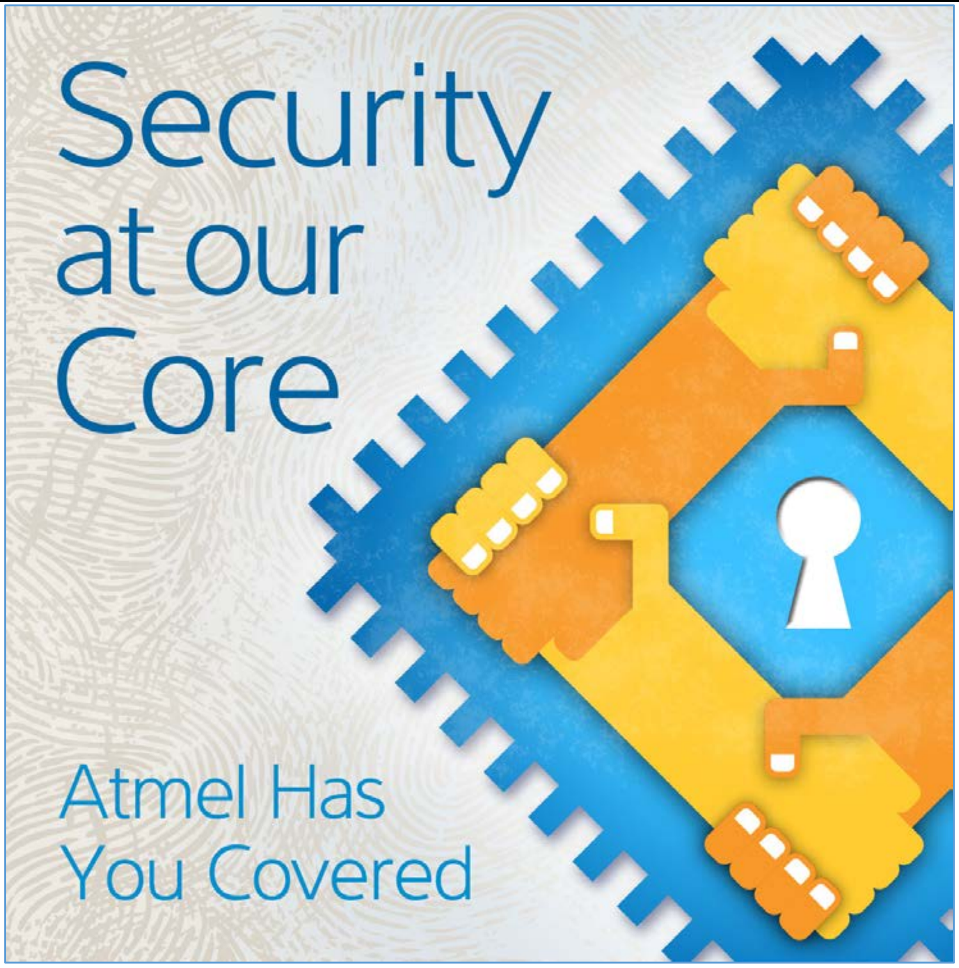
SYMBOL	MIN	NOM	MAX	NOTE
D	2.40	2.50	2.60	
E	6.40	6.50	6.60	
A	0.45	0.50	0.55	
e	1.60	1.70	1.80	
b	1.90	2.00	2.10	
L	2.10	2.20	2.30	
f	0.30	0.40	0.50	
g	0.05	0.15	0.25	
h	2.30	2.40	2.50	
j		4.30	4.40	4.50

1/31/11

	TITLE	GPC	DRAWING NO.	REV.
Atmel Package Drawing Contact: packagedrawings@atmel.com	3RB, 3-lead 2.5x6.5mm Body, 2.0 mm pitch, CONTACT PACKAGE. (Sawn)	RHB	3RB	01

17 Revision History

Doc. Rev.	Date	Comments
1111	03/10/2017	Re-write from 3/1/17 version of ATECC508A datasheet
1111	03/17/2017	Refinement of multiple rough draft issues in first version, formatting cleanup
1111	03/24/2017	Minor updates including new Persistent Latch section
1111	04/07/2017	Fixed broken cross-reference links. Added long tWATCHDOG spec. Fixed character spacing issues. Other minor clarifications
1111	08/06/2017	Converted to Microchip Word Template. Updated most Atmel references to say Microchip



Enabling Unlimited Possibilities®

Atmel Corporation 1600 Technology Drive, San Jose, CA 95110 USA **T:** (+1)(408) 441.0311 **F:** (+1)(408) 436.4200 | www.atmel.com