ATECC108A

Atmel CryptoAuthentication Device

DATASHEET

ATMEL CONDFIDENTIAL
FOR RELEASE ONLY UNDER NON-DISCLOSURE AGREEMENT

CryptoAuthentication

Ensures Things and Code are Real, Untampered, and Confidential

Secure Download and Boot
Authentication and Protect Code In-transit

Ecosystem Control
Ensure Only OEM/Licensed Nodes and Accessories Work

Anti-cloning
Prevent Building with Identical BOM or Stolen Code

Message Security
Authentication, Message Integrity, and Confidentiality of Network Nodes (IoT)

Features

- Crypto Element Devices with Secure Hardware-based Key Storage
- Performs High-Speed Public Key (PKI) Algorithms
  - ECDSA: FIPS186-3 Elliptic Curve Digital Signature Algorithm
- NIST Standard P256, B283, and K283 Elliptic Curve Support
- SHA-256 Hash Algorithm with HMAC Option
- Host and Client Operations
- 256-bit and 283-bit Key Length
- Storage for up to 16 Keys
- Guaranteed Unique 72-bit Serial Number
- Internal High-quality FIPS Random Number Generator (RNG)
- 10Kb EEPROM Memory for Keys, Certificates, and Data
- Storage for up to 16 Keys
- Multiple Options for Consumption Logging and One Time Write Information
- Intrusion Latch for External Tamper Switch or Power-on Chip Enablement. Multiple I/O Options:
  - High-speed Single Pin Interface, with One GPIO Pin
  - 1MHz Standard I^2C Interface
- 2.0V to 5.5V Supply Voltage Range
- 1.8V to 5.5V IO levels
- <150nA Sleep Current
- 8-pad UDFN, 8-lead SOIC, and 3-lead CONTACT Packages

Applications

- Secure Download and Boot
- Ecosystem Control
- Message Security
- 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

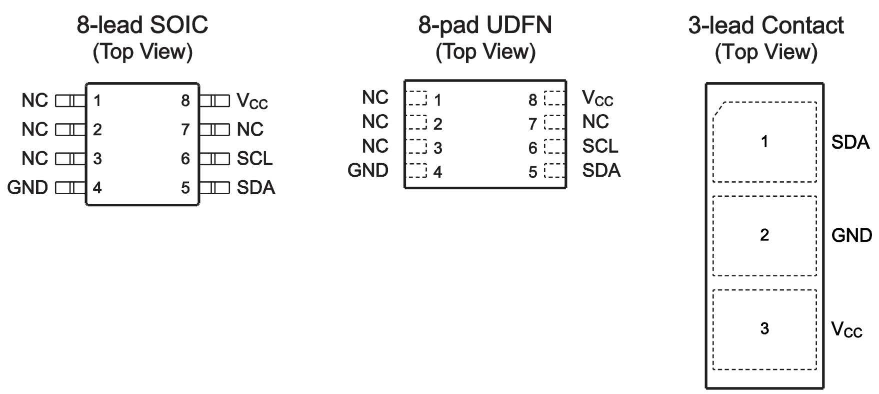


Table of Contents

1 Introduction

1.1 Applications

The Atmel® ATECC108A is a member of the Atmel CryptoAuthentication™ family of crypto engine authentication devices with highly secure hardware-based key storage.

The ATECC108A has a flexible command set that allows use in many applications, including the following, among many others:

- **Network/IoT Node Protection**
  Authenticates node IDs and ensures the integrity of messages.
- **Anti-Counterfeiting**
  Validates that a removable, replaceable, or consumable client is authentic. Examples of clients could be system accessories, electronic daughter cards, or other spare parts. It can also be used to validate a software/firmware module or memory storage element.
- **Protecting Firmware or Media**
  Validates code stored in flash memory at boot to prevent unauthorized modifications, encrypt downloaded program files as a common broadcast, or uniquely encrypt code images to be usable on a single system only.
- **Storing Secure Data**
  Store secret keys for use by crypto accelerators in standard microprocessors. The ATECC108A can be used to store small quantities of data necessary for configuration, calibration, ePurse value, consumption data, or other secrets. Programmable protection is available using encrypted/authenticated reads and writes.
- **Checking User Password**
  Validates user-entered passwords without letting the expected value become known, maps memorable passwords to a random number, and securely exchanges password values with remote systems.

1.2 Device Features

The ATECC108A 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.

The ATECC108A 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 3.3, Security Features). Hardware restrictions on the ways in which keys are used or generated provide further defense against certain styles of attack (see Section 3.2, Key Uses and Restrictions).

Access to the device is made through a standard I^2C Interface at speeds of up to 1Mb/s (see Section 6, I^2C Interface). The interface is compatible with standard Serial EEPROM I^2C 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 5, 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 (see Section 7, General Purpose I/O Pin).

Using either the I^2C or Single-Wire Interface, multiple ATECC108A devices can share the same bus, which saves processor GPIO usage in systems with multiple clients such as different color ink tanks or multiple spare parts, for example. See Sections 4.2, Sharing the Interface and 9.11, `Pause` Command for more details regarding Single-Wire Interface implementation.

Each ATECC108A 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 ATECC108A can generate high-quality FIPS random numbers and employ them 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 (see Sections 3.3.2, Random Number Generator (RNG) and 9.13, `Random` Command).

System integration is easy due to a wide supply voltage range (of 2.0V to 5.5V) and an ultra-low sleep current (of <150nA). Complete DC parametrics are found in Section 8, Electrical Characteristics. Multiple package options are available (See Sections 12, Ordering Information and 14, Package Drawings).

See Section 10 for information regarding compatibility with the Atmel ATSHA204 and ATECC108.

1.3 Cryptographic Operation

The ATECC108A 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, 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 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 ATECC108A also supports a standard hash-based challenge-response protocol in order to simplify programming. In its most basic instantiation, the system sends a challenge to the device, which combines that challenge with a secret key via the `MAC`, `HMAC` or `SHA` commands 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.

Due to the flexible command set of the ATECC108A, these basic operation sets (i.e. ECDSA signatures and SHA-256 challenge-response) can be expanded in many ways. Using the `GenDig` command (see Section 9.4, `GenDig` Command), the values in other slots can be included in the response digest or signature, which provides an effective way of proving that a data read really did come from the device, as opposed to being inserted by a man-in-the-middle attacker. This same command can be used to combine two keys with the challenge, which is useful when there are multiple layers of authentication to be performed.

In a host-client configuration where the host (for instance a mobile phone) needs to verify a client (for instance an OEM battery), there is a need to store the secret in the host in order to validate the response from the client. The `CheckMac` command (see Section 9.2, `CheckMac` Command) allows the device to securely store the secret in the host system and hides the correct response value from the pins, returning only a *yes* or *no* answer to the system.

Finally, the hash combination of a challenge and secret key can be kept on the device and XORed with the contents of a slot to implement an encrypted `Read` command (see Section 9.14, `Read` Command), or it can be XORed with encrypted input data to implement an encrypted `Write` command (see Section 9.19, `Write` Command).

All hashing functions are implemented using the industry-standard SHA-256 secure hash algorithm, which is part of the latest set of high-security cryptographic algorithms recommended by various government agencies and cryptographic experts (see Section 3.1.1, SHA-256 and Section 3.1.2, HMAC/SHA-256). The ATECC108A employs full-sized 256 bit secret keys to prevent any kind of exhaustive attack.

1.4 Commands

The ATECC108A is a command-based device which receives commands from the system, executes those commands, and then returns a result or error code. Within this document, the following nomenclature is used to describe the various commands:

- **Security Commands**
  Described in Section 9, Security Commands; this group of commands generally access the EEPROM space and/or perform cryptographic computation. These commands are indicated with a special font in this document (e.g. `GenDig`) and are available from all interfaces.
- **Cryptographic Commands**
  This subset of the security commands includes all the ECC commands which access the hardware ECC accelerator (`GenKey`, `Sign`, and `Verify`) and the SHA commands which access the hardware SHA accelerator (`CheckMac`, `DeriveKey`, `GenDig`, `HMAC`, `MAC`, `SHA`, and `Nonce`).

2 Device Organization

The ATECC108A contains an integrated EEPROM storage memory and SRAM buffer.

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

Table 2-1. ATECC108A 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 ATECC108A 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 or one-way fuse type consumption logging information.	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 2.5, Static RAM (SRAM) Memory.

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	Only these slots implement the single use feature (Section 3.2.5, Single Use Keys). While all slots support key derivation; only these slots include UpdateCounters (Section 3.2.2, Rolled Keys).
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 – 14	3	72	576	Public Key, Signature or Certificate	For curves 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.
15	3	72	576	Private, Data, Secret Key, Signature, or Certificate	This is the only slot that supports the 128 count limited use feature (Section 3.2.6, Limited Use Key (Slot 15 only)), If this feature is not required, then it can otherwise be used for the same purposes as Slots 9 thru 14.

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 4.1.1, 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.

2.1.1 Certificate Storage

Certificates based upon 256 bit ECC keys generally take substantially more storage than the 72 bytes (576 bits) which are available in Slots 9 thru 15; however the only cryptographic component of these certificates is the ECDSA signature of the item being signed (usually including a public key) which takes only 64 bytes (512 bits). The remaining ten bits in those slots can be used in anyway desired.

The remaining bytes in those signature slots can be used in any way desired, including storage of the necessary compressed information needed to create a full X.509 certificate. In this way, multiple certificates can be efficiently stored in the device.

Generally, the public certificate data will include information such as the public key, creation/expiry dates, static names, certificate serial number, and the like. Contact Atmel for more information on how this information can be stored and for tools to facilitate usage of X.509 certificates.

The public certificate template table can include variable information such as generation and expiry dates, as well as static data such as the algorithm, owner's name, use restrictions, and the like. Such public data can be stored in any slot within the ATECC108A, within the system in flash, or on the network in a database associated with the index or serial number.

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. 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 2.2.6, Special Memory Values in the Config Zone (Bytes 0 thru 12).	Never	Always
4 → 7	RevNum	Device revision number. See Section 2.2.6.	Never	Always
8 → 12	SN[4:8]	Part of the serial number value. See Section 2.2.6.	Never	Always
13	Reserved	Set by Atmel.	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 Atmel 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 Atmel.	Never	Always
16	I2C_Address	When I2C_Enable:0 is one, this field is the I2C_Address with a default value of `0xC0`. Bit 0: RFU must be zero. Bits 1-7: For I^{2}C interface parts the most significant seven bits of this byte form the Device Address value to which this device will respond. When I2C_Enable:0 is zero, this field is the GPIO mode control. Bits 0-1: GPIO Mode (see Section 7, General Purpose I/O Pin). `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: Default state of SCL pin on power-up when configured as an output. Bit 3: 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. Intrusion latch is set via authorization and cleared if SCL falls. Bits 4-7: SignalKey/KeyID. If IO_Mode is `01`, the slot number for the GPIO authorizing key. For all other modes, must be `0000`.	If Config Unlocked	Always
17	Reserved	Reserved. Must be zero.		

Byte	Name	Description	Write	Read
18	OTPmode	`0xAA` (Read-only Mode): Writes to the OTP zone are forbidden when the OTP zone is locked. Reads of all words are permitted. `0x55` (Consumption Mode): Writes to the OTP zone when the OTP zone is locked; causes bits to transition only from a one to a zero. Reads of all words are permitted. `0x00` (Legacy Mode): When the OTP zone is locked, the device operates in a manner compatible with the ATSA102S. All other values of OTP mode are reserved and should not be used.	If Config Unlocked	Always
19	ChipMode	Bit 0: SelectorMode. `0` = Selector can always be written with the `UpdateExtra` command. `1`= Selector can only be written if it currently has a value of zero. Bit 1: TTLenable. `1` = Input levels are V_{CC} referenced. `0` = Input levels use a fixed reference. Bit 2: Watchdog Duration. `0` = $t_{WATCHDOG}$ is 1.3s, nominal. `1` = $t_{WATCHDOG}$ is 10.0s, nominal. Atmel recommends this be set to zero for the best security Bits 3-7: Should be set to `000000`.	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 2.2.1, SlotConfig (Bytes 20 to 51).	If Config Unlocked	Always
52,54,56,58 60,62,64.66	UseFlag	For limited use keys 0 to 7, this byte indicates how many times a key may be used before such use is disabled. Applies to Keys 0 to 7 *only* (byte 52 corresponds to Key0, 54 to Key1, and so on). Initialized to `0xFF`. See 3.2.5, Single Use Keys.	If Config Unlocked	Always
53,55,57,59 61,63,65,67	UpdateCount	For keys that can be updated with the `DeriveKey`, these bytes indicate how many times this operation has been performed. Applies to Keys 0 to 7 *only*, (byte 53 corresponds to Key0, 55 to Key1, and so on). Initialized to `0x00`. See Sections 3.2.2, Rolled Keys, 3.2.4, Created Secret Keys, and 9.3, `DeriveKey` Command.	If Config Unlocked	Always
68 → 83	LastKeyUse	128 bits to control limited use for KeyID 15. Initialized to `0xFF`. See Section 3.2.6, Limited Use Key (Slot 15 only).	If Config Unlocked	Always
84	UserExtra	One byte value that can be modified via the `UpdateExtra` command after the Data zone has been locked.	Via Update Extra Cmd Only	Always
85	Selector	Selects which device will remain in active mode after execution of the `Pause` command. See Sections 9.11, `Pause` Command and 9.17, `UpdateExtra` Command).	Via Update Extra Cmd Only	Always
86	LockValue	Controls the ability to write the OTP and Data zones. `0x55` = unlocked; `0x00` = locked. On shipment from Atmel, this byte has a value of `0x55` corresponding to the unlocked state. After the `Lock` command has been run, this byte will have a value of `0x00`. See Sections 2.4, EEPROM Locking and 9.7, `Info` Command. When locked, the OTP zone can be modified only with the `Write` command, and slots in the Data zone can be modified *only* if the corresponding WriteConfig field so indicates. When unlocked, the `Read` command is prohibited within these two zones.	Via Lock Command Only	Always

Byte	Name	Description	Write	Read
87	LockConfig	Controls the ability to modify the Configuration zone. `0x55` = unlocked; `0x00` = locked. On shipment from Atmel, this byte has a value of `0x55` corresponding to the unlocked state. After the `Lock` command has been run, this byte will have a value of `0x00`. See Section 2.4, EEPROM Locking and 9.7, `Info` Command.	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 9.8, `Lock` Command.	If Config Unlocked, Via Lock Command	Always
90 – 91	RFU	Must be zero.	If Config Unlocked	Always
92 – 95	X509format	Four individual format bytes are associated with the X.509 certificate formatting 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 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.	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 2.2.5, KeyConfig (Bytes 96 thru 127).	If Config Unlocked	Always

2.2.1 SlotConfig (Bytes 20 to 51)

The 16 SlotConfig elements are used to configure the access protections for each of the 16 slots within the ATECC108A 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 this table, the Section 9.14, `Read` Command, and Table 2-6 for more details. `0` = Then this slot can be the source for the `CheckMac` copy operation. See Section 3.2.7, 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 `CheckMac` or `GenKey` are enabled. Bit 2 and 3: Reserved for the future use. These bits must be set to zero. For slots containing public keys that can be validated (PubInfo is one, see Section 2.2.5, 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 or HMAC commands. When this key is used to generate or modify TempKey, then that value may not be used by the MAC and HMAC commands. `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". See Section 3.2.6, Limited Use Key (Slot 15 only). ▶ LimitedUse is only supported for slots 0 to 7. This bit must be zero for slots 8 – 14. `0` = There are no usage limitations.
6	EncryptRead	`1` = Reads from this slot will be encrypted using the procedure specified in the `Read` command (Section 9.1.4, Address Encoding) using ReadKey (bits 0 to 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. See Section 9.19, `Write` Command.
12 → 15	WriteConfig	Controls the ability to modify the data in this slot. See Table 2-7, Table 2-8, Table 2-10, and Section 9.19.

2.2.2 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 Section 9.14, `Read` Command. The encryption key is in the slot specified by ReadKey. 4-byte reads and writes are prohibited.

2.2.3 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 2.2.4, 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	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 (Section 9.19, `Write` Command). 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). See Section 3.2, Key Uses and Restrictions.

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*. 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. It may only be used for crypto operations.

Some security policies require that secrets be updated from time to time. The ATECC108A 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.

2.2.4 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 (Section 9.12, `PrivWrite` Command).

2.2.5 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 `0x5` to validate the slot.
2 → 4	KeyType	If the slot contains an ECC public or private key, then the key type field below must be set to indicate a curve type supported by the device. If the slot contains any other kind of data, key, or secret, then this field *must* be set to seven for proper operation. `0` = B283 NIST ECC key `1` = K283 NIST ECC key `2` = RFU (reserved for future use) `3` = RFU (reserved for future use) `4` = P256 NIST ECC key `5` = RFU (reserved for future use) `6` = RFU (reserved for future use) `7` = Not an ECC key
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`, `HMAC`, `CheckMac`, `Verify`, `DeriveKey`, and `GenDig`. `1` = A random nonce is required. `0` = A random nonce is *not* required.

Bit	Name	Description
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 3.2.9, 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	IntrusionDisable	`1` = Then use of this key is prohibited for the following commands if the IntrusionLatch is zero: `CheckMac`, `DeriveKey`, `GenDig`, `HMAC`, `MAC`, and `Sign`. All other commands are permitted regardless of the state of the latch. `0` = Then use of this key is independent of the state of the IntrusionLatch.
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.

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 ATECC108A 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 ATECC108A 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.

- **KeyType**
  The four ECC commands that use private keys (i.e. `GenKey`, `Sign, 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 SHA-256 computation commands (i.e. `CheckMac`, `DeriveKey`, `MAC`, or `HMAC`) 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 SHA-256 computation commands (i.e. `CheckMac`, `DeriveKey`, `MAC`, or `HMAC`) should be set to a KeyType of seven. 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`, `MAC`, and `HMAC` 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`, `HMAC`, `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.

2.2.6 Special Memory Values in the Config Zone (Bytes 0 thru 12)

Various fixed information is included in the ATECC108A that can never be written under any circumstances and can always be read, regardless of the state of the lock bits.

- **SerialNum**
  Nine bytes (SN[0:8]) that together form a unique value that is never repeated for any device in the CryptoAuthentication family. 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 ATECC108A. Their default value is `0x01 23 EE`. These 24 bits are always included in the cryptographic computations that the ATECC108A makes.
  - **SN[2:3] and SN[4:7]**
    The values of these bits are programmed by Atmel 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 ATECC108A.
- **RevNum**
  Four bytes of information that are used by Atmel to provide manufacturing revision information. These bytes can be freely read as RevNum[0:3], but they should never be used by system software because they may be revised by Atmel from time to time.

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 or consumption logging purposes. It is organized as two blocks of 32 bytes each.

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. If desired, the data to be written can be encrypted. Prior to locking the Data/OTP zones using LockValue, this zone cannot be read at all.

Once the OTP zone is locked, the OTPmode byte in the Configuration zone controls the access permissions of this zone as follows:

- **Read-only Mode**
  The data cannot be modified and would be used to store fixed model numbers, calibration information, manufacturing history, or other data that should never change. 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.
- **Consumption Mode**
  The bits function as one-way fuses and can be used to track consumption or usage of the item to which the ATECC108A device is attached. In a battery, for example, they might be used to track charging cycles or use time. In a printer ink cartridge, they might track the quantity of material consumed. In a medical device, they might track the number of permitted uses for a limited use item.

  The `Write` command can only cause bits to transition from a one to a zero. Logically, this means that the data value in the input parameter list will be ANDed with the current value in the word(s), and the result written back to memory. As an example, writing a value of `0xFF` results in no change to the byte and writing a value of `0x00` causes the byte in memory to go to zero, regardless of the previous value. Once a bit has transitioned to a zero, it can never transition back to a one.

  All 64 bytes within the OTP zone are always available for reading using either 4 or 32 byte reads.
- **Legacy Mode**
  Only supported in the ATECC108A. Contact Atmel for more details.

All OTP bits have a value of one upon shipment from the Atmel 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 lock bits 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.

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.

It is vital that the Data and OTP zones be locked prior to release into the field of the system containing the device. 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 Atmel for optional secure personalization services.

2.4.1 Configuration Zone Locking

Certain bytes within the Configuration zone cannot be modified regardless of the state of LockConfig. Access to the remainder of the 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)	Read	<Never>

2.4.2 Data and OTP Zone Locking

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 and OTP Zone Access Restrictions

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

Note: 1. After the Data/OTP zones are locked using LockValue, reads and writes of the OTP zone additionally depend on the state of the OTP mode bytes in the Configuration zone. See Section 2.3, EEPROM One Time Programmable (OTP) Zone for more information.

2.4.3 Individual Slot Locking

ATECC108A 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 2-14. Individual Slot Locking After Configuration Zone is Locked

SlotLocked Bit	Lockable Bit	Lock Command	`PrivWrite`, `Write`, `DeriveKey`, 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/OTP zones are 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.

2.5 Static RAM (SRAM) Memory

The device includes an SRAM array that is used to store the input command or output result, intermediate computation values, and/or an ephemeral key. The entire contents of this memory are always invalidated whenever the device goes into sleep mode or the power is removed. The ephemeral key is named TempKey and can be used as an input to the `MAC`, `HMAC`, `CheckMac`, `GenDig`, `Sign`, `Verify`, and `DeriveKey` commands. It is also used as the data protection (encryption or decryption) key by the `Read` and `Write` commands.

2.5.1 TempKey

TempKey is a storage register in the SRAM array that can be used to store an ephemeral result value from the `Nonce`, `GenDig`, `SHA`, or `GenKey` commands. The contents of the 32 byte data value in this register can never be read from the device (although the device itself can read and use the contents internally). The `Info` command can be used to return the value of the nine status/flag bits within this register.

Execution of `GenDig` or `GenKey` replaces the old contents of TempKey with the new calculated output, which is a combination of the old TempKey value and other information. Execution of the `Nonce` command or the copy mode of the `CheckMac` command completely replaces any previous output of the `GenDig` or `GenKey` commands. This register contains the elements shown in the table below:

Table 2-15. TempKey Storage Register

Name	Length	Description
TempKey	256 bit (32 byte)	Nonce (from `Nonce` command) or digest (from `GenDig` command).
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 seed only, no internal random generation (Input).
GenDigData	1 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` and `HMAC` commands. 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, TempKey refers to the contents of the 256 bit data register. The remaining bit fields are referred to as TempKey.SourceFlag, TempKey.GenDigData, and so forth.

The TempKey.Valid bit is cleared to zero during power-up, sleep, brown-out, watchdog expiration, or tamper detection. The contents of TempKey are retained when the device enters idle mode. Depending upon the command and the circumstances, the TempKey.Valid bit is also cleared as follows:

- **`Nonce`, `GenKey`, or `GenDig` Commands**:
  TempKey.valid will be cleared on any error other than CRC (communications) or ECC (retry).
- **`CheckMac` Command**:
  TempKey.valid will be cleared unless a successful copy takes place (Section 3.2.7, Password Checking).
- **`Info` Command**:
  TempKey.valid is not modified regardless of success or failure.
- **All Others**:
  TempKey.valid will be cleared for all return codes (including success) other than CRC (communications) or ECC (retry).

3 Security Information

3.1 Cryptographic Standards

The ATECC108A follows various industry standards for the computation of cryptographic results. These reference documents are described in the sections below.

3.1.1 SHA-256

The ATECC108A `MAC` command calculates the digest of a secret key concatenated with the challenge or nonce. It optionally includes various other pieces of information stored on the device within the digested message. The ATECC108A computes the SHA-256 digest based upon the algorithm documented in the following site:

http://csrc.nist.gov/publications/fips/fips180-2/fips180-2.pdf

The complete SHA-256 message processed by the ATECC108A is listed in Section 9, Security Commands 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.

3.1.2 HMAC/SHA-256

The response to the challenge can also be computed using the HMAC algorithm based upon the SHA-256 documented at the following site:

http://csrc.nist.gov/publications/fips/fips198/fips-198a.pdf

Because of the increased computation complexity, the `HMAC` command is not as flexible as the `MAC` command, and the computation time is extended for HMAC. While the HMAC sequence is not necessary to ensure the security of the digest, it is included for compatibility with various software packages.

3.1.3 Elliptic Curve Digital Signature Algorithm (ECDSA)

The ATECC108A 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

3.2 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 3.2.1, Diversified Keys to 3.2.9, 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 3.2.8, Transport Keys, the most significant 12 bits of all KeyID parameters should be zero.

3.2.1 Diversified Keys

If the host or validating entity has a place to securely store secrets, or contains an ATECC108A 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 ATECC108A key slot.

The ATECC108A `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

3.2.2 Rolled Keys

In order to prevent repeated uses of the same secret key value, the ATECC108A 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. Each time the roll operation is performed on Slots 0 to 7, the `UpdateCount` field for that slot is incremented in order to permit the host device to re-create the key value.

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 either the key or the UpdateCount 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.

3.2.3 Created ECC Keys

For the highest security, private ECC keys may be created within the ATECC108A using the internal high quality RNG. These 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 ATECC108A.

The public key corresponding to the generated private key is returned to the system, and the device can also use another internally stored key to create a MAC or signature (using the `Sign-Internal` command) covering the new public key.

3.2.4 Created Secret Keys

There may be a need to have unique ephemeral symmetric keys on each client; a function also supported by the ATECC108A. 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 Sections 3.2.5 Single Use Keys and 3.2.6, Limited Use Key (Slot 15 only)). 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. Each time the create operation is performed on Slots 0 to 7, the UpdateCount field for that slot is incremented.

3.2.5 Single Use Keys

For the KeyID values corresponding to slots zero through seven in the Data zone of the EEPROM, repeated usage of the key stored in the slot can be strictly limited. This feature is enabled if the LimitedUse bit is set in the SlotConfig field. The LimitedUse bit is ignored for Slots 8 thru 14. The number of remaining uses is stored as a bit map in the UseFlag byte corresponding to the slot in question.

Prior to execution of any of the commands that use this slot as a key, the following takes place:

- If SlotConfig[keyId].LimitedUse is set and UseFlag[KeyID] is `0x00`, then the device returns an error.
- Starting at bit 7 of UseFlag[keyID], clear to zero the first bit that is currently a one.

In practice, this procedure permits LimitedUse keys to be used eight times. If power is lost during the execution of any command referencing a key that has this feature enabled, one of the use bits in UseFlag may still be cleared even though the command did not complete. For this reason, Atmel recommends that the key be used a single-time only, with the other bits providing a safety margin for errors.

The single use capability is checked for the key specified by the KeyID parameter within the `CheckMac`, `GenDig`, `HMAC`, `MAC`, and `Sign` commands. The `DeriveKey` command checks single use restrictions only for the parent key. The `GenKey` command checks single use restrictions only when asked to generate the public key for a previously loaded or generated private key.

The single use capability is explicitly ignored for the slot being copied to TempKey during a `CheckMac(Copy)` command. (See Sections 3.2.7, Password Checking and 9.2, `CheckMac` Command). This facility provides a robust method to prevent any alternate access method for the target slot. To limit the use of the target slot being copied, use the single use capability for the parent slot.

Under normal circumstances, all eight UseFlag bytes should be initialized to `0xFF`. If it is the intention to permit fewer than eight uses of a particular key, these bytes should be initialized to `0x7F` (seven uses), `0x3F` (six uses), `0x1F` (five uses), `0x0F` (four uses), `0x07` (three uses), `0x03` (two uses), or `0x01` (one use). Initialization to any other value besides these values or `0xFF` is prohibited.

During normal use, the `GenKey` and `DeriveKey` commands may also reset the UseFlag to `0xFF` when a new key is created in a slot for which Single Use is enabled. See below.

The `Read`, `Write`, `GenKey`, and `DeriveKey` commands operate slightly differently:

- `Read` and `Write` Commands
  The `Read` and `Write` commands ignore the state of the LimitedUse bit and the useFlag byte does not change as a result of their execution. LimitedUse slots in which the useFlag is exhausted (value of `0x00`) can still be read or written (subject to the appropriate SlotConfig limitations) although the value in the slot cannot ever be used as a key for cryptographic commands.

If SlotConfig.WriteKey for slot X points back to X, but useFlag[X] is exhausted, encrypted writes to the slot will never succeed because the prior `GenDig` command will have returned an error due to the usage limitation. A similar situation occurs with reads and ReadKey.

- `DeriveKey` Command

  If the parent key is used for either MAC calculation or as the source, then if LimitedUse (for the parent) is set and useFlag (also for the parent) is `0x00`, the `DeriveKey` command will return an error. The LimitedUse and UseFlag bits are ignored for the target key. When successfully executed, the `DeriveKey` command always resets the useFlag to `0xFF` for the target key.

  Use of the `DeriveKey` command is optional. It is legal to be run only if WriteConfig:13 is set for this slot. In some situations it may be advantageous to simply have a key that can be used eight times, in which case the other security commands will clear the bits in UseFlag one at a time until they all are cleared, at which time the key is disabled.
- `GenKey` Command

  When successfully executed in key creation mode, the `GenKey(Create)` command always resets the useFlag to `0xFF` for the target key.

3.2.6 Limited Use Key (Slot 15 only)

If slot[15].LimitedUse is set, usage of key number 15 is limited through a different mechanism than the single use limitation described in the previous section (which applies only to Slots 0 thru 14). Contact Atmel for more information on how to use Slot 15 in combination with other slots to limit the use of a key to any count between 128 and 800,000.

Prior to any use of Key 15 by a command, the following takes place:

- If all bytes in LastKeyUse are `0x00`, then return error.
- Starting at bit 7 of the first byte of LastKeyUse (byte 68 in Configuration zone), clear to zero the first bit, which is currently a one. If byte 68 is `0x00`, then check bit 7 of byte 69, and so forth up through byte 83. Only a single bit is cleared each time prior to using Key 15.

There is no reset mechanism for this limitation. After 128 uses (or the number of one bits set in LastKeyUse on personalization), key 15 is permanently disabled. This capability is not susceptible to power interruptions. Even if the power is interrupted during execution of the command, only a single bit in LastKeyUse will be unknown, all other bits in LastKeyUse will be unchanged, and the key will remain unchanged.

If fewer than 128 uses are desired for Key 15, then some of the bytes within this array should not be initialized to `0xFF`. The only legal values for bytes within this field (besides `0xFF`) are `0x7F`, `0x3F`, `0x1F`, `0x0F`, `0x07`, `0x03`, `0x01`, or `0x00`. The total number of bits set to one indicates the number of uses.

`Example:` How to set 16 uses is as follows:
`0xFF`, `0xFF`, `0x00`, `0x00`, `0x00`, `0x00`, `0x00`, `0x00`,
`0x00`, `0x00`, `0x00`, `0x00`, `0x00`, `0x00`, `0x00`, `0x00`

The Limited Use capability applies to the same commands, and in the same situations, in which they are checked for the LimitedUse feature (see the previous section for more information). In addition, the `Verify` command will check for single use restrictions on both the public key (when that key is stored internally), and the key to be validated when the command is run in validation mode and either is stored in Slot 15.

3.2.7 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 ATECC108A 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`, `HMAC`, and `Read` commands).
- Valid is set to one.

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

3.2.8 Transport Keys

The ATECC108A 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 Atmel. 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.

3.2.9 Authorized Keys

The ATECC108A 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` Command
  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` Command
  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.

3.3 Security Features

3.3.1 Physical Security

The ATECC108A incorporates a number of physical security features designed to protect the EEPROM contents from unauthorized exposure. The security measures include:

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

Pre-programmed transport keys stored on the ATECC108A 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.

3.3.2 Random Number Generator (RNG)

The ATECC108A includes a high-quality RNG which returns 32 random bytes to the system. See http://csrc.nist.gov/groups/STM/cavp/documents/drbg/drbgval.html for further documentation on NIST CAVP certification of this RNG. The device generally combines this generated number with a separate input number to form a nonce that is stored within the device in TempKey and may be used by subsequent commands.

The system may use this RNG for any purpose. The device provides a special `Random` command for such purposes that do not affect the internally stored nonce.

Random numbers are generated from a combination of the output of a hardware RNG and an internal seed value, which is not externally accessible. The internal seed is stored in the EEPROM and is normally updated once after every power-up or sleep/wake cycle. After the update, this seed value is retained in registers within the device that are invalidated if the device enters sleep mode, or the power is removed.

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.

4 General I/O Information

Communications to the ATECC108A 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.
- **I^2C Interface**
  This mode is compatible with the I^2C standard and also with the Atmel AT24C16 Serial EEPROM interface. Two pins, Serial Data (SDA) and Serial Clock (SCL), are required. The I^2C interface supports a bit rate of up to 1Mb/s.

  The ATECC108A and AT24C16B have different default I^2C addresses. The ATECC108A I^2C address can be modified from default by writing a new value into the Configuration zone.

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 security commands and error codes, which are documented in Section 9, Security Commands.

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 9.1.6, Watchdog Failsafe.

4.1 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 4.1.1, 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 ATECC108A Least Significant bit (LSb) first on the bus.
- On the I^2C interface, data is transferred to and from the ATECC108A Most Significant bit (MSb) first on the bus.

4.1.1 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 regardless of the curve in use:
  - B283: The most significant six bits of the first byte are pad bits.
  - K283: The most significant seven bits of the first byte are pad bits.
  - P256: 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:

- When the public key is an output of `GenKey` command or an input to `Verify` command:
  - B283/K283: The most significant five bits of the first and 37th byte are pad bits (72 bytes).
  - P256: 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.
  - B283/K283: The most significant five bits of the first and 37th byte are pad bits.
  - P256: 4 pad bytes, 32 bytes of X, four pad bytes, and 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 the following:
  - B283/K283: 17 bytes of pad, followed by 36 bytes of X, then 36 bytes of Y.
  - P256: 25 bytes of pad, followed by 32 bytes of X, and 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 noted below:
  - For B283 and K283: The most significant five bits of the first and 37th byte are pad bits.
  - For P256: 4 pad bytes, 32 bytes of X, four pad bytes, and 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.

4.2 Sharing the Interface

Multiple CryptoAuthentication devices may share the same interface, as follows:

1. The system issues a Wake token to wake-up all devices.
2. The system issues the `Pause` command (Section 9.11, `Pause` Command) to put all but one of the devices into the idle mode. Only the remaining device will then see any of the commands that the system sends. When the system has completed talking to the one active device, it then sends an idle flag that puts the remaining active device into the idle mode and the idle devices will ignore.

Steps 1 and 2 are repeated for each device on the wire. If the system has completed communications with the final device, it should wake up all the devices, and then put all the devices to sleep to reduce total power consumption.

The device uses the Selector byte within the Configuration zone to determine which device stays awake. Only that device with a selector value that matches the input parameter of the `Pause` command will stay awake. In order to facilitate late configuration of systems which use the multi-device sharing mode, the following three update capabilities for the Selector byte are supported:

- **Unlimited Updates**:
  At any time, the `UpdateExtra` command can be executed to Write the value in the selector field of the Configuration zone. To enable this mode, clear the SelectorMode bit in ChipMode.
- **One-time Field Update**:
  If the SelectorMode bit is set to one, and the Selector byte has a zero value prior to locking the Configuration zone, then at any time after the Configuration zone is locked the `UpdateExtra` command can be used one-time to set Selector to a non-zero value. The `UpdateExtra` command is not affected by the LockValue byte.
- **Fixed Selector Value**:
  The Selector byte can never be modified after the Configuration zone is locked if SelectorMode is set to one and the Selector byte is set to a non-zero value. The `UpdateExtra` command will always return an error code.

5 Single-Wire Interface

In this mode, communications to and from the ATECC108A 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 ATECC108A.

See the Atmel 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 11.2, Wiring Configuration for Single-Wire Interface for more information about how to connect the device in the Single-Wire Interface mode.

5.1 I/O Tokens

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

- **Input** (to the ATECC108A):
  - 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 ATECC108A):
  - 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 ATECC108A 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 ATECC108A, 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.

5.2 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 5-1. 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 ATECC108A 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 9-4, 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 ATECC108A to the idle mode, which causes the input/output buffer to be flushed. It does ***not*** invalidate the contents of the TempKey and RNG Seed 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 ATECC108A 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.

5.3 Synchronization

Because the communications protocol is half-duplex, there is the possibility that the system and the ATECC108A 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.

5.3.1 I/O Timeout

After a leading transition for any data token has been received, the ATECC108A 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.

5.3.2 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.

6 I²C Interface

The I²C Interface uses the SDA and SCL pins to indicate various I/O states to the ATECC108A. This interface is designed to be compatible at the protocol level with the Atmel AT24C16 Serial EEPROM operating at 1MHz.

Note Well: There are many differences between the two devices (for example, the ATECC108A and AT24C16 have different default I²C addresses); therefore, designers should read the respective datasheets carefully.

The SDA pin is normally pulled high with an external pull-up resistor because the ATECC108A 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 ATECC108A 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.

6.1 I/O Conditions

The device responds to the following I/O conditions:

6.1.1 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 I²C 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 6.1.2, 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 I²C 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 ATECC108A devices on the bus, and the I²C interface is run at 133KHz or slower, the transmission of certain data patterns (such as `0x00`) will cause all the ATECC108A 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.

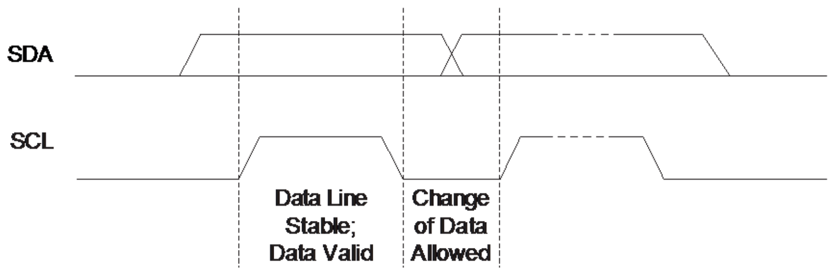
In the I²C mode, the device will ignore a wake sequence that is sent when the device is already awake.

6.1.2 Device is Awake

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

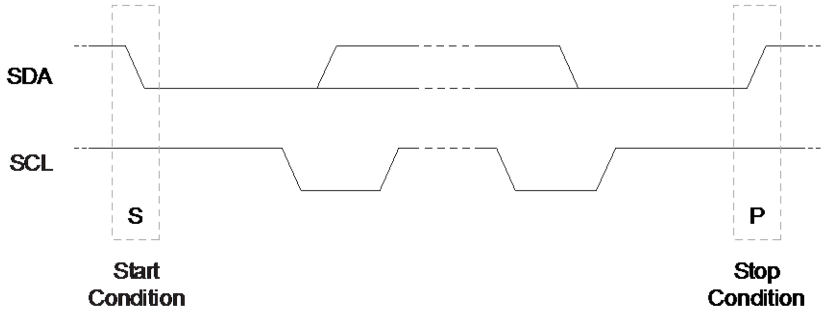
- **DATA Zero:** If SDA is low and stable while SCL goes from low to high to low, then a zero bit is being transferred on the bus. SDA can change while SCL is low.
- **DATA One:** If SDA is high and stable while SCL goes from low to high to low, then a one bit is being transferred on the bus. SDA can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

Figure 6-1. Data Bit Transfer on I²C Interface



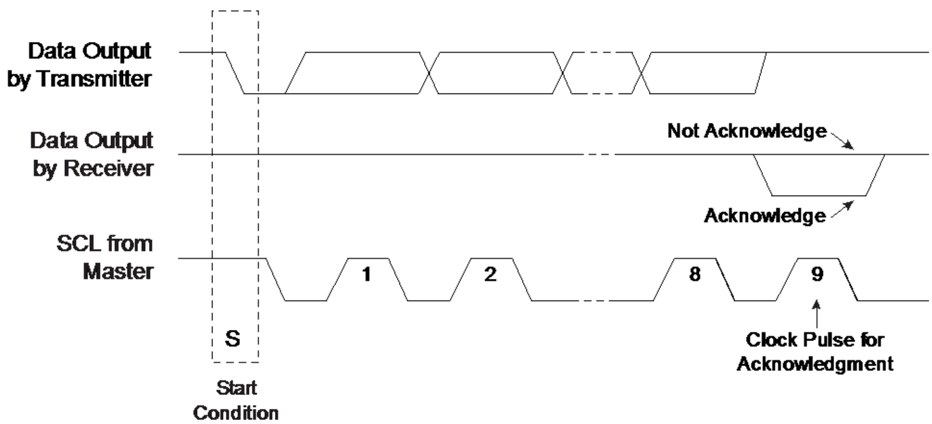
- **Start Condition:** A high-to-low transition of SDA with SCL high is a Start condition which must precede all commands.
- **Stop Condition:** A low-to-high transition of SDA with SCL high is a Stop condition. After this condition is received by the device, the current I/O transaction ends. On input, 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 6-2. 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 6-3. NOT ACK and ACK Conditions on I²C Interface



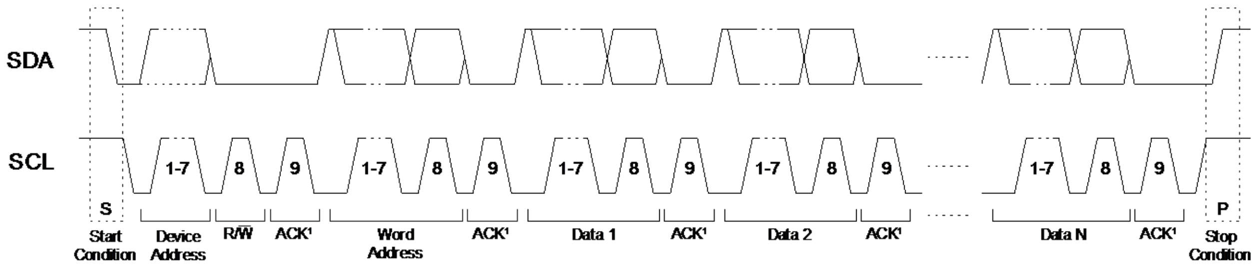
Multiple ATECC108A devices can easily share the same I²C interface signals if the I2C_Address byte in the Configuration zone is programmed differently for each device on the bus. Because all seven of the bits of the device address are programmable, ATECC108A can also share the I²C interface with any I²C device, including any Serial EEPROM.

6.2 I²C Transmission to ATECC108A

The transmission of data from the system to the ATECC108A 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 6-4. Normal I²C Transmission to ATECC108A



Note: SDA is driven low by ATECC108A ACK periods.

The tables below label the bytes of the I/O transaction. The column labeled "I²C Name" provides the name of the byte as described in the AT24C16 datasheet.

Table 6-1. I²C Transmission to ATECC108A

Name	I²C Name	Description
Device Address	Device Address	This byte selects a particular device on the I²C interface. ATECC108A 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 I²C 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 6.2.1, Word Address Values and 6.6, 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 9.1, 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 I²C 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 ATECC108A will start the computation of the command. Failure to send a Stop condition may eventually result in a loss of synchronization; see Section 6.8, I²C Synchronization for recovery procedures.

6.2.1 Word Address Values

During an I^2C write packet, the ATECC108A interprets the second byte sent as the word address, which indicates the packet function as it is described in the table below:

Table 6-2. Word Address Values

Name	Value	Description
Reset	`0x00`	Reset the address counter. The next I^2C read or write transaction will start with the beginning of the I/O buffer.
Sleep (Low-power)	`0x01`	The ATECC108A 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 ATECC108A goes into the idle mode and ignores all subsequent I/O transitions until the next wake flag. The contents of TempKey and RNG Seed 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.

6.2.2 Command Completion Polling

After a complete command has been sent to the ATECC108A, 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 6.5, I^2C Transmission from the ATECC108A). 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.

6.3 Sleep Sequence

Upon completion of the use of the ATECC108A 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.

6.4 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.

6.5 I^2C Transmission from the ATECC108A

When the ATECC108A is awake and not busy, the bus master can retrieve the current buffer contents from the device using an I^2C Read. If valid command results are available, the size of the group returned is determined by the particular command which has been run (see Section 9, Security Commands); 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 8-4, I^2C Synchronous Data Timing.

Table 6-3. I^2C Transmission from the ATECC108A

Name	I^2C Name	Direction	Description
Device Address	Device Address	To slave	This byte selects a particular device on the I^2C interface and ATECC108A 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 I^2C 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 9.1, 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 ATECC108A along the I^2C 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 ATECC108A 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 ATECC108A will *not* ACK the device address to indicate that no data is available to be read.

6.6 Address Counter

Writes to and/or reads from the ATECC108A I/O Buffer over the I^2C interface are treated as if the device were a FIFO. Either the I^2C 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 ATECC108A 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 9.1.2, 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.

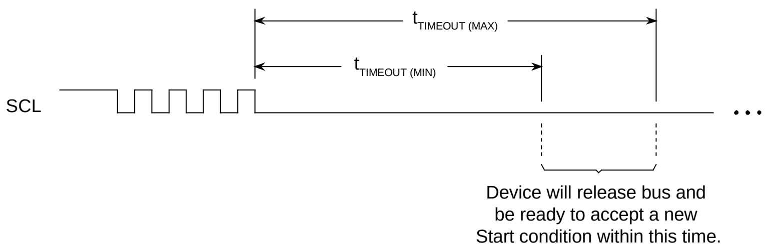
There may be situations where the system may wish to re-read the output buffer, for example when the CRC check reveals an error. In this case, the host should send a two-byte sequence to the ATECC108A consisting of the correct device address and a word address of `0x00` (Reset per Table 6-2, Word Address 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.

6.7 SMBus Timeout

The ATECC108A supports the SMBus Timeout feature in which the ATECC108A 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 ATECC108A will be ready to accept a new Start condition before $t_{TIMEOUT}$ maximum has elapsed.

Figure 6-5. SMBus Timeout



6.8 I²C Synchronization

It is possible for the system to lose synchronization with the I/O port on the ATECC108A, perhaps due a system reset, I/O noise, or other condition. Under this circumstance, the ATECC108A 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 I²C 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 ATECC108A 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 ATECC108A 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.

7 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 7-1. 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^2C 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	Input	The SCL pin will be permanently configured as an input. On power-up, an internal intrusion latch is set to zero. The intrusion latch is set via authorization and is cleared if SCL falls. The state of latch can be determined via the `Info` command. It will remain in that state so long as a voltage greater than 1.8V is applied to the SCL pin and V_{CC} remains above 2.0V regardless of the internal state (asleep, idle, or wake) of the ATECC108A. Any falling edge on the SCL pin resets the intrusion latch to zero regardless of whether or not the ATECC108A is in wake or sleep mode. Reading the state of the GPIO pin via the `Info` command returns the value of the intrusion 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.

8 Electrical Characteristics

8.1 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.

8.2 Reliability

The ATECC108A is fabricated with the Atmel high reliability of the CMOS EEPROM manufacturing technology.

Table 8-1. EEPROM Reliability

Parameter	Min	Typical	Max	Units
Write Endurance (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

8.3 AC Parameters: All I/O Interfaces

Figure 8-1. AC Timing Diagram: All Interfaces

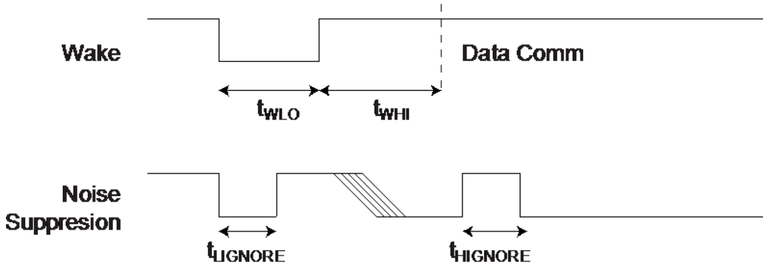


Figure 8-2. AC Parameters: All I/O Interfaces

Parameter [1]	Symbol	Direction	Min	Typ	Max	Unit	Notes
Power-Up Delay	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	500			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	Maximum time from wake until device is forced into sleep mode. See Section 9.1.6, Watchdog Failsafe.

Note: 1. These parameters are guaranteed through characterization, but not tested.

8.3.1 AC Parameters: Single-Wire Interface

Figure 8-3. AC Timing Diagram: Single-Wire Interface

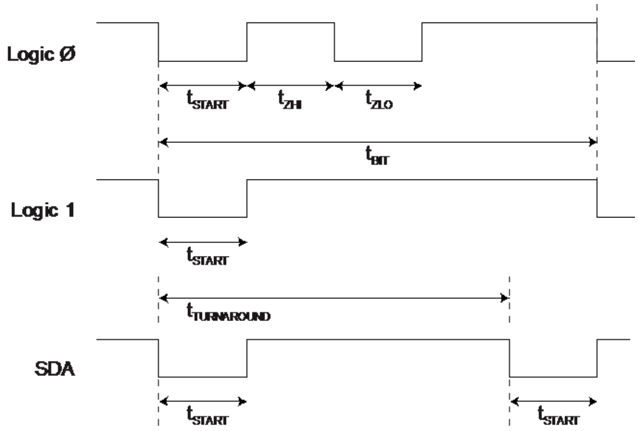


Table 8-2. AC Parameters: Single-Wire Interface

Applicable from TA = -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 ATECC108A may enter the sleep mode. See Section 5.3.1, I/O Timeout.
		From Crypto Authentication	41	54	78	µs	
Turn Around Delay	$t_{TURNAROUND}$	From Crypto Authentication	64	96	131	µs	ATECC108A 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 ATECC108A 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 ATECC108A.
IO Timeout	$t_{TIMEOUT}$	To Crypto Authentication	45	65	85	ms	ATECC108A may transition to the sleep mode if the bus is inactive longer than this duration. See Section 5.3.1.

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.

8.3.2 AC Parameters: I²C Interface

Figure 8-4. I²C Synchronous Data Timing

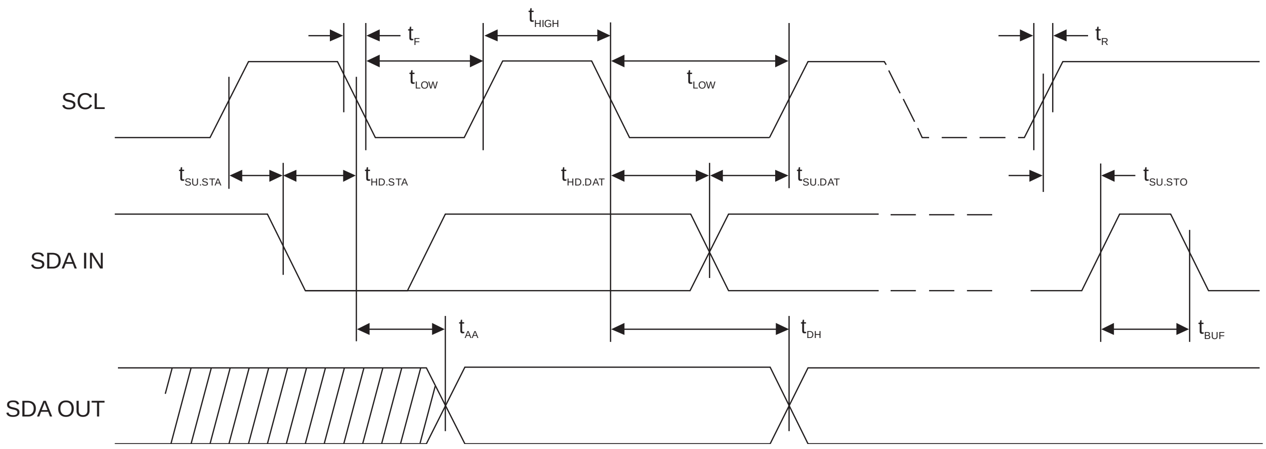


Table 8-3. AC Characteristics of I²C Interface

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		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.

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}

8.4 DC Parameters: All I/O Interfaces

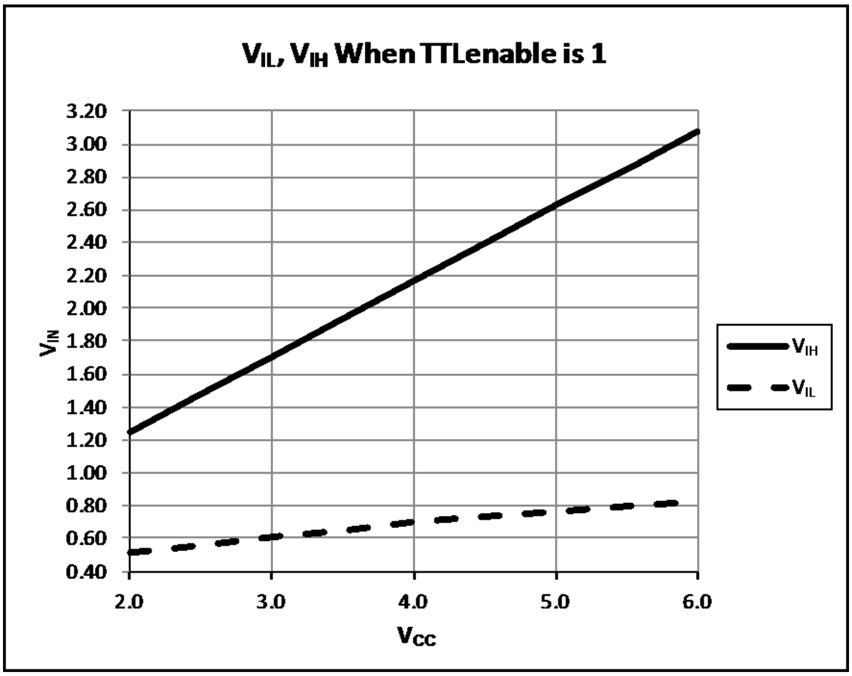
Table 8-4. DC Parameters on All I/O Interfaces

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}		3	6	mA	Waiting for I/O during I/O transfers or execution of non-ECC commands when ChipMode:3 is zero.
			—	16	mA	During ECC command execution. See Table 2-4, Configuration Zone for options when ChipMode indicates a slower internal clock speed.
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

8.4.1 V_{IH} and V_{IL} Specifications

The input voltage thresholds when in sleep or idle mode are dependent on the V_{CC} level as shown in the graph below. 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. When a common voltage is used for the ATECC108A V_{CC} pin and the input pull-up resistor, then this bit should be set to a one, which permits the input thresholds to track the supply as shown in the graph below.

Figure 8-5. V_{IH} and V_{IL} When TTLenable = 1 on All I/O Interfaces



If the voltage supplied to the V_{CC} pin of the ATECC108A 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 according to the following table. The following applies only when the device is active:

Table 8-5. V_{IL}, V_{IH} on All I/O Interfaces

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.

9 Security Commands

9.1 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 9-1. 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 ATECC108A 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 ATECC108A 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.

9.1.1 Security Command Packets

The security command packet is broken down as shown in Table 9-2, below:

Table 9-2. Security Command Packets

Byte	Name	Meaning
0	Opcode	The command code. See Section 9.1.3, Command Opcodes, Short Descriptions, 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 ATECC108A 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 9.1.3.

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.

9.1.2 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 9-3, 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 9-3. 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 ATECC108A. 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.
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 ATECC108A 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.

9.1.3 Command Opcodes, Short Descriptions, and Execution Times

During parsing of the parameters and subsequent 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 and its parameter values, the state of the device, the environmental conditions, and other factors according to the following table:

Table 9-4. Command Opcodes, Short Descriptions, and Execution Time

Command	Opcode	Description	Typ. Exec. Time(1)	Max. Exec. Time(2)	Unit
`CheckMac`	0x28	Verify a MAC calculated on another CryptoAuthentication device.	5	12	ms
`DeriveKey`	0x1C	Derive a target key value from the target or parent key.	3	50	ms
`GenDig`	0x15	Generate a data digest from a random or input seed and a key.	6	10	ms
`GenKey`	0x40	Generate an ECC public key. Optionally generate an ECC private key.	9	101	ms
		P256 Keys.	9	90	
`HMAC`	0x11	Calculate response from key and other internal data using HMAC/SHA-256.	14	23	ms
`Info`	0x30	Return device state information.	0.1	1	ms
`Lock`	0x17	Prevent further modifications to a zone of the device.	9	23	ms
`MAC`	0x08	Calculate response from key and other internal data using SHA-256.	5	11	ms
`Nonce`	0x16	Generate a 32-byte random number and an internally stored Nonce.	0.1	6	ms
`Pause`	0x01	Selectively put just one device on a shared bus into the idle mode.	0.1	3	ms
`PrivWrite`	0x46	Write an ECC private key into a slot in the Data zone.	0.8	47	ms
`Random`	0x1B	Generate a random number.	2	23	ms
`Read`	0x02	Read four bytes from the device, with or without authentication and encryption.	0.1	1	ms
`Sign`	0x41	Perform ECDSA Sign operation. B283 and K283 Keys	22	25	ms
		P256 Keys.	34	37	
`SHA`	0x47	Computes a SHA-256 or HMAC digest for general purpose use by the system.	7	9	ms
`UpdateExtra`	0x20	Update bytes 84 or 85 within the Configuration zone after the Configuration zone is locked.	9	10	ms
`Verify`	0x45	Perform ECDSA verify operation. B283 and K283 Keys P256 Keys.	30	60	ms
			38	72	

Command	Opcode	Description	Typ. Exec. Time[1]	Max. Exec. Time[2]	Unit
`Write`	`0x12`	Write 4 or 32 bytes to the device, with or without authentication and encryption.	7	26	ms

Notes: 1. Typical execution times are representative of the duration to execute the command assuming no error conditions, fastest mode setting, and favorable environmental conditions. For best performance, delay for this interval and then start polling to determine actual command completion.

2. Maximum execution times are representative of the longest duration of a successful command execution under the worst case statistical and environmental conditions. Some internal modes such as limited use and others will cause the delays to be as much as 50ms longer.
In most but not all cases, failing commands will return relatively quickly, often well before the typical execution time.

9.1.4 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).

Table 9-5. 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 9-6. 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 9-7. 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 mode is read-only, all offsets in both blocks are available to use with 4 or 32 byte reads. If OTP mode is consumption, then writes are also permitted to all offsets. See Table 2-7, Write Configuration Bits: **`Write`** Command if OTP mode is legacy.
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.

Slot 8 contains an additional nine blocks, each containing 32 bytes that are not included in Table 9-8.

To use a four byte `Read` or `Write` command to access the first word in a block, use the addresses shown in Table 9-8; 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 9-8. Data Zone Address Values

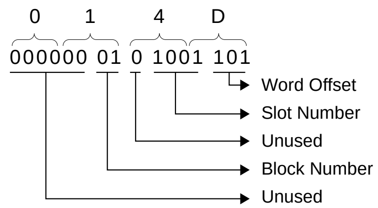
	Block 0		Block 1		Block 2		Block 3	
Slot	Address	Size	Address	Size	Address	Size	Address	Size
0	0x0000	32	0x0100	4				
1	0x0008	32	0x0108	4				
2	0x0010	32	0x0110	4				
3	0x0018	32	0x0118	4				
4	0x0020	32	0x0120	4				
5	0x0028	32	0x0128	4				
6	0x0030	32	0x0130	4				
7	0x0038	32	0x0138	4				
8	0x0040	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 9-6, the address parameter to the `Read` command is `000000 01 0 1001 101` or `0x014`.



9.1.5 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 9-9. Zone Encoding (Param1)

Zone	Param1 Value	Size	Read	Write
Config	0	1024 bits 128 bytes 2 Slots	Always available.	Partially, when unlocked. Never when locked. Never encrypted.
OTP	1	512 bits 64 bytes 2 Slots	Never when unlocked. Always when locked, except in legacy mode, see Section 2.5, Static RAM (SRAM) Memory.	All writeable when unlocked using `Write`. When locked, write permissions depend on OTPmode, see Section 2.5.
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.

9.1.6 Watchdog Failsafe

After the ATECC108A 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 two SRAM registers (TempKey and RNG Seed).

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

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.

9.2 CheckMac Command

The CheckMAC command calculates a MAC response that would have been generated on an ATECC108A, 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 3.2.9, 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, and/or LastKeyUse for the target slot.

Table 9-10. 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 – 4: Must be zero. Bit 5: = Use 64 bits of OTP zone in calculation.0 = Use 64 zeros. Bits 6 – 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 9-11. 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	OTP[0:7]	(or 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]	

9.3 `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`.

- ATECC108A:

  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 UseFlag[ParentKey] is `0x00`. `DeriveKey` always ignores LimitedUse and UseFlag for the target key.

  For Slots 0 thru 7 only, if input parsing and the optional MAC check succeed, then useFlag[TargetKey] gets set to `0xFF`, and UpdateCount[TargetKey] is incremented. If UpdateCount currently has a value of 255, then it wraps to zero. If the command fails for any reason, then these bytes are not updated. The value of updateCount may be corrupted if power is interrupted during the execution of `DeriveKey`.

Table 9-12. 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 9-13. Output Parameter

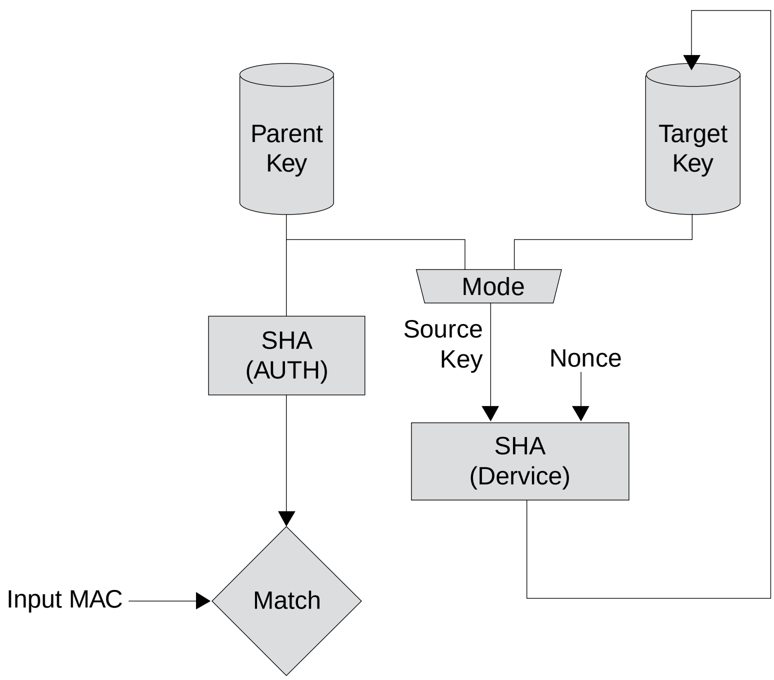
Name	Size	Notes
Success	1	Upon successful completion, the ATECC108A returns a value of zero.

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 9-1. Data Flow for DeriveKey Command



9.4 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, 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, CheckMac, or HMAC 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 2.5, Static RAM (SRAM) Memory 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 9-14. 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), is not supported by ATECC108A. 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 9-15. Output Parameter

Name	Size	Notes
Success	1	Upon successful execution, ATECC108A 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 “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

9.5 GenKey Command

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

1. **Private Key Creation:**
   Creates a new random private key and writes that key into the slot specified by the KeyID parameter. The EEPROM RNG seed will automatically be updated prior to the execution of this command if it has not been already updated this power cycle.
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, it 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.
- For Slots 0 – 7 *only*, if key creation succeeds, then useFlag[KeyID] gets set to `0xFF` and updateCount[KeyID] is incremented. If updateCount currently has a value of 255, then it wraps to zero. If the command fails for any reason, then these bytes are not updated. The value of updateCount may be corrupted if power is interrupted during the execution of `GenKey`. useFlag[TargetKey] is ignored on input to `GenKey`.
- There is a small statistical probability that the generated key will be unacceptable for the curve specified, in which case this command will return a single byte containing the ECC fault code (see Table 9-3, 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 9-16. 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 9-17. Output Parameter

Name	Size	Notes
Response	1 or 64 or 72	Public Key X and Y coordinates. ECC fault code if generated private key was unacceptable.

Table 9-18. 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 of a 566 bit public key (i.e. the public part of a 283 bit private key) 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]
17 bytes	Zeros
72 bytes	X and Y coordinates of the public key. Upper five bits of both are masked to zeros.

When the public key length is 512 bits, the message is as follows:

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

9.6 HMAC Command

The `HMAC` command computes an HMAC/SHA-256 digest using a key stored in the device over a challenge stored in the TempKey register, and/or other information stored within the device. The output of this command is the output of the HMAC algorithm computed over this message using the specified key.

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.
3. Run this `HMAC` 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).

See the `SHA` command, which can generate an HMAC digest over an arbitrary length message without any special formatting.

Table 9-19. Input Parameters

	Name	Size	Notes
Opcode	HMAC	1	`0x11`
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 HMAC message.
Data	—	0	—

Table 9-20. Output Parameter

Name	Size	Notes
Response	32	HMAC digest.

The HMAC digest is computed using the key at KeyID as the HMAC key over a message consisting of the following information:

32 bytes	Zeros	
32 bytes	TempKey	
1 byte	Opcode	(always `0x11`)
1 byte	Mode	
2 bytes	KeyID	
8 bytes	OTP[0:7]	(or zeros.)
3 bytes	OTP[8:10]	(or zeros.)
1 byte	SN[8] bits	(*never* zeroed out)
4 bytes	SN[4:7] bits	(or zeros, see Table 9-21)
2 bytes	SN[0:1]	(*never* zeroed out)
2 bytes	SN[2:3]	(or zeros, see Table 9-21)

Table 9-21. Mode Encoding

Bits	Meaning
0 – 1	Must be zero.
2	The value of this bit must match the value in TempKey.SourceFlag or the command will return an error.
3	Must be zero.
4	1 = Include the first 88 OTP bits (OTP[0] through OTP[10]) in the message; otherwise, the corresponding message bits are set to zero.
5	1 = Include the first 64 OTP bits (OTP[0] through OTP[7]) in the message; otherwise, the corresponding message bits are set to zero. If Mode[4] is set, the value of this mode bit is ignored.
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.

9.7 `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 9-22. 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 EEPROM RNG: Seed has been updated this power cycle. Bit 1 SRAM RNG: Seed has been updated this power cycle. 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`.

Table 9-23. 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 9-24. 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^2C 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 intrusion latch. If Param2:1 indicates output, and if AuthValid is one and AuthKey matches SignalKey, then set the intrusion latch to Param2:Bit 0. The Intrusion latch can be set to one only.
- 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.

9.8 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 ATECC108A 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 one 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 9-25. Input Parameters

	Name	Size	Notes
Opcode	LOCK	1	`0x17`
Param1	Mode	1	See Table 9-27.
Param2	Summary	2	Summary (CRC) of the designated zones, or should be `0x0000` if mode:7 is set.
Data	Ignored	0	—

Table 9-26. Output Parameter

Name	Size	Notes
Success	1	Upon successful execution, ATECC108A returns a value of zero.

Table 9-27. 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. `1`: Check of the zone summary is ignored and the zone is locked regardless of the contents of the zone. Atmel does not recommend using this mode.

9.9 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 9-28. 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 9-29. 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 Table 9-30, Mode Encoding)
32 bytes	Challenge or TempKey	(see Mode Encoding)
1 byte	Opcode	(always 0x08)
1 byte	Mode	
2 bytes	Param2	
8 bytes	OTP[0:7]	(or zeros)
3 bytes	OTP[8:10]	(or zeros)
1 byte	SN[8] bits	(*never* zeroed out)
4 bytes	SN[4:7] bits	(or zeros, see Mode Encoding)
2 bytes	SN[0:1]	(*never* zeroed out)
2 bytes	SN[2:3]	(or zeros, see Mode Encoding)

Table 9-30. 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	Must be zero.
4	`1`: Include the first 88 OTP bits (OTP[0] through OTP[10]) in the message; otherwise, the corresponding message bits are set to zero.
5	`1`: Include the first 64 OTP bits (OTP[0] through OTP[7]) in the message; otherwise, the corresponding message bits are set to zero. If Mode[4] is set, the value of this mode bit is ignored.
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.

9.10 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 ATECC108A 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 must be 32 bytes long and it is passed directly to TempKey without modification. No SHA-256 calculation is performed and TempKey.SourceFlag is set to Input. 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 9-31. Input Parameters

	Name	Size	Notes
Opcode	NONCE	1	`0x16`
Param1	Mode	1	Controls the mechanism of the internal RNG and seed update.
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	Input value from system.

Table 9-32. 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. If Mode[0:1] is one, the automatic random number seed update is suppressed. See Section 3.3.2, Random Number Generator (RNG).

32 bytes	RandOut	
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, the input parameter (NumIn) must be 32 bytes long and TempKey is loaded with NumIn. No SHA-256 calculation is performed, no data is returned to the system, and TempKey.SourceFlag is set to Input.

Table 9-33. Mode Encoding

Bits	Meaning
0 – 1	`0`: Combine new random number with NumIn, store in TempKey. Automatically update EEPROM seed only if necessary prior to random number generation. Recommended for highest security. `1`: Combine new random number with NumIn, store in TempKey. Generate random number using existing EEPROM seed, do *not* update EEPROM seed. Not recommended for general use. `2`: Invalid. `3`: Operate in pass-through mode and Write TempKey with NumIn.
2 – 7	Must be zero.

9.11 Pause Command

All devices on the bus for which the configuration Selector byte does ***not*** match the input Selector parameter will go to the idle mode. This command is used to prevent bus conflicts in a system that includes multiple ATECC108A devices sharing the same bus.

This command differs from the Idle Flag/Sequence in that individual devices on the single pin bus may be selected to go into the idle mode, as opposed to the Idle Flag which causes all the CryptoAuthentication devices on the bus into the idle mode.

If the EEPROM Selector byte does ***not*** match the input selector parameter, then the device will immediately go to the idle mode and no result information will be available. If the input selector parameter does match the configuration Selector byte, then the device returns a success code of `0x00`.

The `Pause` command cannot be used to put the devices into the sleep mode.

Table 9-34. Input Parameters

	Name	Size	Notes
Opcode	PAUSE	1	`0x01`
Param1	Selector	1	All devices that do not match this value go to idle mode.
Param2	Zero	2	Must be `0x0000`.
Data	Ignored	0	

Table 9-35. Output Parameter

Name	Size	Notes
Success	1	If the command indicates that some other device should idle, ATECC108A returns a value of `0x00`. If this device goes to idle, no value is returned.

9.12 **`PrivWrite` Command**

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

For best security, Atmel 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. For B283 keys, the first six bits on the bus should have a plain text value of zero, for K283 keys the first seven bits should be zero, and for P256 keys 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 9-36. 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 9-37. Output Parameter

Name	Size	Notes
Success	1	Upon successful completion, ATECC108A returns a value of zero.

9.13 Random Command

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

Random numbers are generated through a combination of the output of a hardware RNG and an internal seed value stored in the EEPROM or SRAM. The external system may choose to update the internally stored EEPROM seed value prior to the generation of the random number as part of the execution of the `Nonce` or `Random` command.

The `Random` command does not provide a mechanism to integrate an input number with the internal stored seed. If this functionality is desired, then the system should use the `Nonce` command and ignore the generated Nonce.

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

The same internal stored seeds are used for both the `Nonce` and `Random` commands.

Table 9-38. Input Parameters

	Name	Size	Notes
Opcode	RANDOM	1	`0x1B`
Param1	Mode	1	Controls the mechanism of the internal RNG and seed update.
Param2	Zero	2	Must be `0x0000`.
Data	Ignored	0	—

Table 9-39. Output Parameter

Name	Size	Notes
RandOut	32	The output of the RNG.

Table 9-40. Mode Encoding

Bits	Meaning
0	0 = Automatically update EEPROM seed only if necessary prior to random number generation. Recommended for highest security. 1 = Generate random number using existing EEPROM seed, do ***not*** update EEPROM seed.
1 – 7	Must be zero.

9.14 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 9.1.4, Address Encoding 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, if OTPmode is set to a non-zero value and the address points to either word zero or one, then the command also returns an error; otherwise, the corresponding word within the OTP zone is returned in the clear. If OTPmode is Legacy, then only four byte reads are permitted.
- **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 9-41. Input Parameters

	Name	Size	Notes
Opcode	READ	1	`0x02`
Param1	Zone	1	Bits 0 and 1: Select among Configuration, OTP, or Data. See Section 9.1.4. Bits 2-6: Must be zero. Bit 7: `1` = 32 bytes are read; otherwise four bytes are read. Must be zero if reading from OTP zone.
Param2	Address	2	Address of first word to be read within the zone. See Section 9.1.4.
Data	—	0	—

Table 9-42. 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*.

9.15 SHA Command

Computes a SHA-256 or HMAC/SHA digest for general purpose use by the system. It may also be used by the `Verify(ValidateExternal)` command to verify an X.509 certificate and store that validation status with an internally stored public key.

Calculation of a SHA-256 digest occurs in the following three steps:

1. **Start**:
   Initialization of the SHA-256 calculation engine and initialization of the SHA context in memory. This mode does not accept any message bytes.
2. **Update**:
   The command can be called a variable number of times with this mode to add bytes to the message. Each iteration of this mode must include a message of 64 bytes. A variation on `SHA(Update)` is `SHA(Public)` which inserts the contents of a public key slot into the message.
3. **End**:
   The SHA-256 calculation is completed, and the resulting digest is placed into the output buffer. It is also stored in the TempKey register for subsequent (optional) use by the `Verify(ValidateExternal)` command. From 0 bytes to 63 bytes may be passed to the device for this mode.

On any error return code, or if any command other than `SHA` is sent to the device, the internal SHA context is invalidated and TempKey is also invalidated.

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.

`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.

Calculation of an HMAC digest occurs in the following three steps, which are similar to the SHA process above with the exception that a key slot is specified in the first step, and its value is used in the calculations at the beginning and end of the message per the HMAC specification.

1. **HMACstart**:
   Initialization of the HMAC calculation engine and initialization of the SHA context in memory. A stored key legal for use with SHA operations must be specified. This mode does not accept any message bytes.
2. **Update**:
   Along with SHA(Public), identical to SHA-256.
3. **HMACend**:
   The HMAC calculation is completed re-using the key value from HMACstart, and the resulting digest is placed into the output buffer. It is also stored in the TempKey register. From 0 bytes to 63 bytes may be passed to the device for this mode.

Table 9-43. Input Parameters

	Name	Size	Notes
Opcode	SHA	1	`0x47`
Param1	Mode	1	Bits 0-2 = `000` (Start): Load TempKey with the initialization value for SHA2-56. No message bytes are accepted (Length must be zero). Bits 0-2 = `001` (Update): Add 64 bytes in the message parameter to the SHA context. Bits 0-2 = `010` (End): Complete the SHA-256 computation and load the digest into TempKey and the output buffer. Up to 63 message bytes are accepted (Length must be 0 through 63 inclusive.) Bits 0-2 = `011` (Public): Add 64 bytes of a public key stored in one of the Data zone slots to the SHA context. Param2 should contain the slot ID of the public key, and the command will return an error if the slot contains anything other than a public key. No further bytes should appear in the input stream (Message size is zero). Bits 0-2 = `100` (HMACstart): Load TempKey with the initialization value for SHA2-56. Length field specifies the key to be used for the HMAC calculation. No message bytes are accepted. Bits 0-2 = `101` (HMACend): Complete the HMAC/SHA-256 computation and load the digest into TempKey and the output buffer. Up to 63 message bytes are accepted (Length must be 0 through 63 inclusive.) Bits 3-7: Must be zero.
Param2	Length	2	Number of bytes in the Message parameter. KeySlot for the HMAC key if Mode is “HMACstart”.
Data	Message	0 – 64	Up to 64 bytes of data to be included into the hash operation.

Table 9-44. Output Parameter

Name	Size	Notes
Response	1 or 32	The SHA256 digest if Mode is `10`, otherwise zero for success or an error code.

9.16 `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 using the `Nonce` command. The ATECC108A cannot compute the SHA-256 digest of a random external message.
  - External signatures must be enabled using SlotConfig.ReadKey:0 or else this command will return an error.
  - For 283 bit ECC keys, the message in TempKey is extended to 36 bytes as follows:
    - If SN[8] has a value of `0xEE`, then the message in TempKey is zero extended prior to being signed; otherwise, the message is extended with SN[0], SN[1], SN[8], and the MSB of the message is zero.
- **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 ephemeral key is always generated using the RNG. The EEPROM RNG seed must have been updated prior to the execution of this command or else Sign will return an error. This can be accomplished with the `Random` command, ignoring the output data.

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.

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 9-3, Status/Error Codes in Four byte Groups). The command will generally succeed if resubmitted.

Table 9-45. 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 9-46. Output Parameter

Name	Size	Notes
Response	64 or 72	The signature composed of R and S, or an error code.

Table 9-47. 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 `0`.
1 – 5	Must be `0`.
6	`1`: Include the 48 bits SN[2:3] and SN[4:7] in the message for internal signatures `0`: The corresponding message bits are set to `0`. This bit is ignored if mode:7 is `1`.
7	`0`: The message to be signed is internally generated as below. `1`: The message to be signed is in TempKey and sign extended as above.

Internal signatures are always generated over digest information placed in TempKey by `GenKey` or `GenDig`, and include further configuration information regarding the key used for the TempKey calculation.

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. This 55 byte message is created as follows:

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 zero.

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	UseFlag[TempKeyFlags.keyId]	(*see note below)
1 byte	UpdateCount[TempKeyFlags.keyId]	(*see note below)
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. This secondary digest is always zero extended prior to being signed.

Note: * On the ATECC108A, the UseFlag and UpdateCount bytes in this message are always set to `0x00`. Regardless, these bytes are set to zero if TempKeyFlags.keyId is eight or higher.

9.17 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.

- If Mode:1 is set, The command implements a fast decrement of the limited use counters which may be associated with a particular key.
  - If the slot indicated by the "NewValue" param does not contain a key for which limited use is implemented or enabled, then the command returns silently without taking any action.
  - If the indicated slot contains a limited use key, which does not have any uses remaining, then the command returns an error. The command does not modify Config.UpdateCount for the slot in question.
- 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 Selector at address 85:
  - If the SelectorMode (bit 1 of the device mode within the Configuration zone) is set to one and Selector (byte 85 of the Configuration zone) is non-zero, then this command will Write Selector with the LS byte of NewValue and return success.
  - If SelectorMode is cleared, indicating that no check of the current Selector should be made, then this command always updates Selector and always succeeds.

Table 9-48. Input Parameters

	Name	Size	Notes
Opcode	UpdateExtra	1	0x20
Param1	Mode	1	Bit 0: 0 = Update Config byte 84. 1 = Update Config byte 85. Bit 1: 1 = Ignore bit 0, and decrement the limited use counter associated with the key in slot NewValue. 0 = Update Config byte 84 or 85. Bits 2 – 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 9-49. 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.

9.18 `Verify` Command

The `Verify` command takes an ECDSA [R,S] signature and verifies that it is correctly generated from a given message and public key. In all cases, the signature is an input to the command. The most significant five bits of the X and Y values will be ignored for all B283 or K283 keys.

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.

   Note: The Data1 through Data4 must either all be 32 or all be 36 bytes long.

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. 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 3.2.9, Authorized Keys.
   - Data1 and Data2 may be 32 or 36 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 4.1.1, 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 3.2.3, 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. The ValidateExternal mode is not supported for 283 bit key sizes.

Table 9-50. 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[3:7] Must be zero.
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.
Data1	R	32 or 36	The R component of the ECDSA signature to be verified.
Data2	S	32 or 36	The S component of the ECDSA signature to be verified.
Data3	X	0, 32, or 36	The X component of the public key to be used for verification if Mode:0-1 is External.
Data4	Y	0, 32, or 36	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 9-51. 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.

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

- **Stored, External, and ValidateExternal Modes**
  The contents of TempKey should contain the SHA-256 digest of the message.
  - 283 bit private (566 bit public) Keys (ECC108A only)
    TempKey is extended as follows:

    If SN[8] has a value of `0xEE`, then the contents of the TempKey register are zero extended to form the message, which is input to the ECDSA Verify algorithm in addition to the signature and public key; otherwise, TempKey is extended with SN[0], SN[1], SN[8] and a fourth byte of zero.
  - 256 bit private (512 bit public) Keys
    TempKey does not need to be extended with any pad bits prior to execution of the ECDSA Verify algorithm.
- **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.

9.19 `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 Section 9.12, `PrivWrite` Command).

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 and the OTPmode byte in the Configuration zone is read-only or legacy, then this command returns an error; otherwise, OTP mode should be consumption and this command sets to zero those bits in the OTP zone that correspond to the zero bits in the input parameter Value. When the data and OTP zones are locked, encrypted writes to the OTP zone are never permitted regardless of OTPmode.
- **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 only permitted in the OTP zone if all of the following conditions are met:

- Config:OTPmode must be consumption.
- The input data must not be encrypted, i.e. Zone:6 must be zero.
- The Data/OTP zones must be locked.

The two input address bytes are formed in a manner to achieve compatibility with the ATSHA204 (see Section 9.1.4, Address Encoding). 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).

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.

9.19.1 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 9-52. Input Parameters

	Name	Size	Notes
Opcode	Write	1	`0x12`
Param1	Zone	1	Bits 0 – 1: Select among Config, OTP or Data. See Section 9.1.5, 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 9.1.4, Address Encoding.
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 9-53. Output Parameter

Name	Size	Notes
Success	1	Upon successful completion, ATECC108A returns a value of zero.

10 Compatibility

10.1 Atmel ATSHA204

ATECC108A is fully compatible with the ATSHA204 devices. If properly configured, it can be used in all situations where the ATSHA204 or ATSHA204A is currently employed. Because the Configuration zone is larger, the personalization procedures for the device must be updated when personalizing the ATSHA204 or ATSHA204A. For proper compatibility, care should be taken with the KeyType, ReqRandom, and ReqAuth slots containing keys that are used with ATSHA204 or ATSHA204A sequences.

10.2 Atmel ATECC108

ATECC108A is designed to be fully compatible with the ATECC108 and ATECC508A devices. If properly configured, can be used in all situations where ATECC108 is currently employed. In many situations, the ATECC508A can also be used in an ATECC108 application without change. The new revisions provide significant advantages as outlined below:

New Features in ATECC108A vs. ATECC108

- Intrusion Detection Capability, Including Gating Key Use
- New `SHA` Command, Also Computes HMAC
- X.509 Certificate Verification Capability
- Programmable Watchdog Timer Length
- Shared Random Nonce and Key Configuration Validation (`Gendig` Command)
  Larger Slot 8 which is Extended to 416 bytes

Minor Changes in ATECC108A vs ATECC108

- The `GenDig` Command Verifies that a Random Nonce is Used When Generating Transport Keys
- The `Info` Command DevRev Mode Now Returns `0x1006` for ATECC108A. This value should not be used in the software as it will vary with each minor revision.

Fixes vs. ATECC108

- Minimum V_{CC} is 2.0V
- TTLenable Bit in the Configuration Zone Switches the Input Thresholds Per the Datasheet
- SIO Sleep Mode Errata Eliminated
- I/O Timeout Meets Datasheet Specification
- I^2C Start/Stop and Repeated Start No Longer Returns the CRC Error
- Individually Locked Slots Can Properly Be Read

11 Mechanical

11.1 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 11-1. Package Pinouts

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

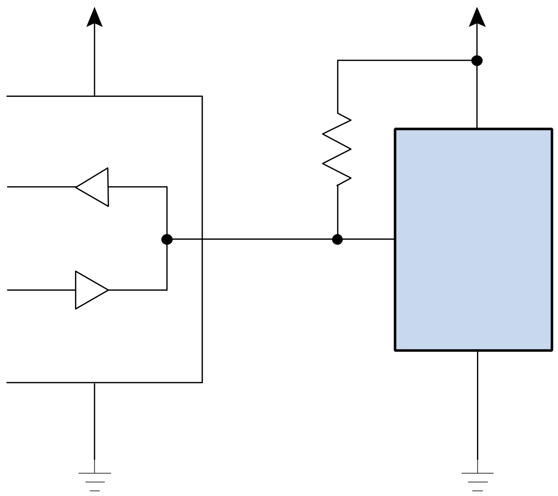
11.2 Wiring Configuration for Single-Wire Interface

Using the Single-Wire Interface allows the connection of ATECC108A 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 ATECC108A device is physically distant from the bus master device and the supply voltage for the bus master is different than the supply voltage for ATECC108A.

Figure 11-2. 3-wire Configuration for Single-Wire Interface



12 Ordering Information

Atmel Ordering Code[4]	Package	Interface Configuration
ATECC108A-SSHCZ-T	8-lead SOIC, Tape and Reel[2]	Single-Wire
ATECC108A-SSHCZ-B	8-lead SOIC, Bulk in Tubes[1]	Single-Wire
ATECC108A-SSHDA-T	8-lead SOIC, Tape and Reel[2]	I^2C
ATECC108A-SSHDA-B	8-lead SOIC, Bulk in Tubes[1]	I^2C
ATECC108A-MAHCZ-T	8-pad UDFN, Tape and Reel[2]	Single-Wire
ATECC108A-MAHDA-T	8-pad UDFN, Tape and Reel[2]	I^2C
ATECC108A-RBHCZ-T[3]	3-lead CONTACT, Tape and Reel[2]	Single-Wire

Notes:
1. B = Bulk
2. T = Tape and Reel
    - SOIC = 4,000 units per reel.
    - UDFN = 15,000 units per reel.
    - RBH = 5,000 units per reel.
3. Please contact Atmel for availability.
4. Please contact Atmel for thinner packages.

13 Errata

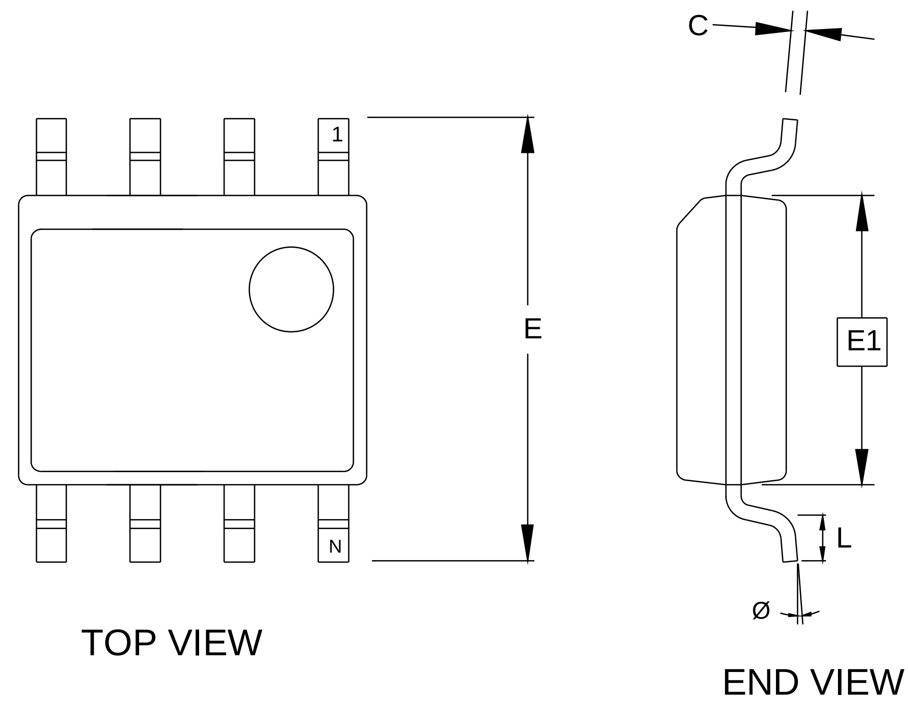
13.1 ATECC108A

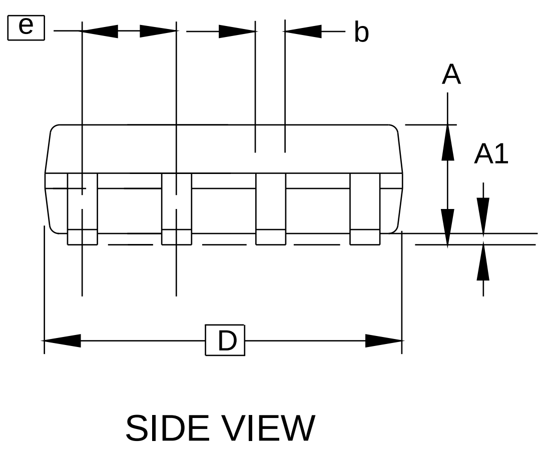
Issue: **For device revisions `0x1005 and` lower (use the `Info` command to get device revision).**

- Slot 8 is limited to 224 bytes. Attempts to write data beyond this limit will result in errors.
- GenDig modes of SharedRandomNonce and KeyConfig not supported.

14 Package Drawings

14.1 8-lead SOIC





COMMON DIMENSIONS
(Unit of Measure = mm)

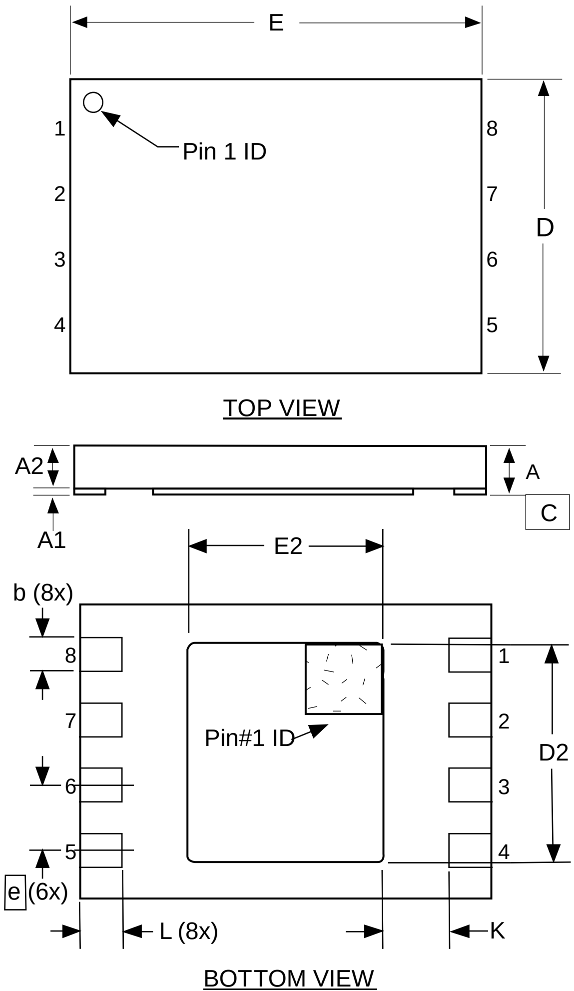
SYMBOL	MIN	NOM	MAX	NOTE
A	–	–	1.75	
A1	0.10	–	0.25	
b	0.31	–	0.51	
C	0.17	–	0.25	
D	4.90 BSC			
E	6.00 BSC			
E1	3.90 BSC			
e	1.27 BSC			
L	0.40	–	1.27	
Ø	0°	–	8°	

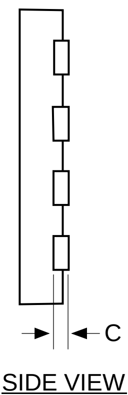
Notes: This drawing is for general information only.
Refer to JEDEC Drawing MS-012, Variation AA
for proper dimensions, tolerances, datums, etc.

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

14.2 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.

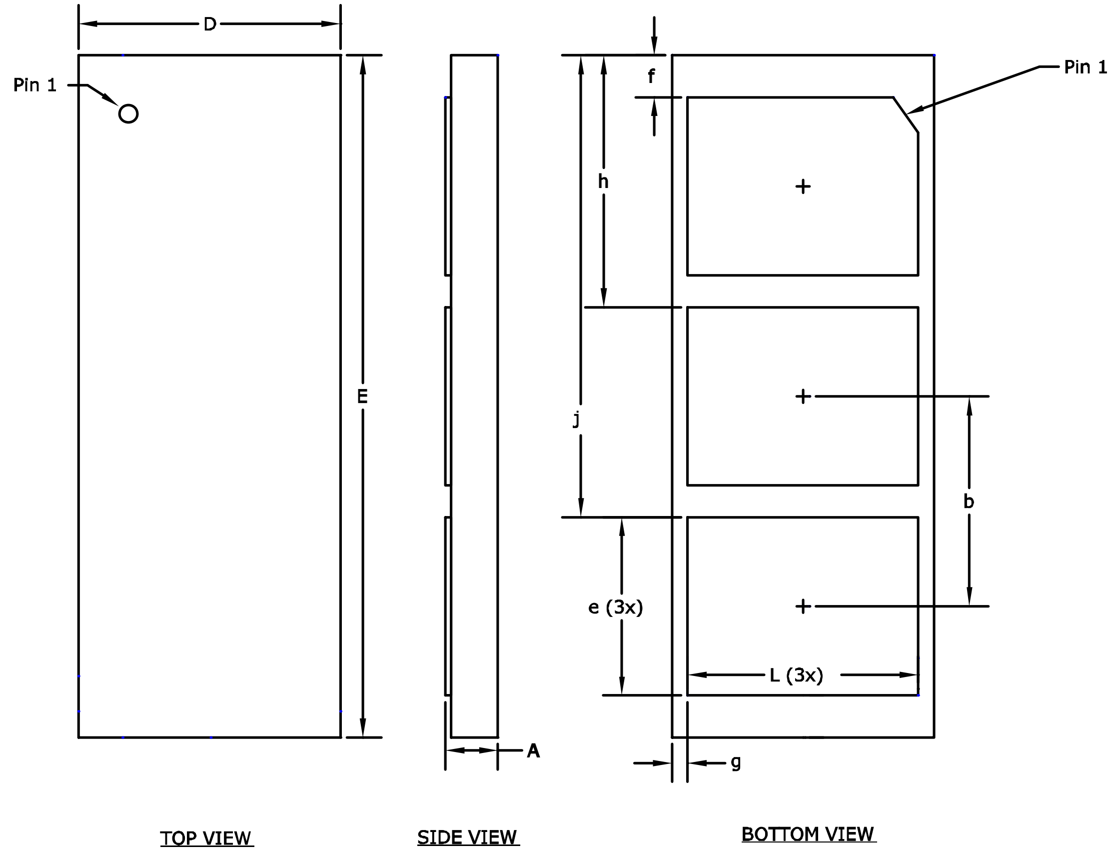
COMMON DIMENSIONS
(Unit of Measure = mm)

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

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

14.3 3-lead CONTACT



COMMON DIMENSIONS
(Unit of Measure = mm)

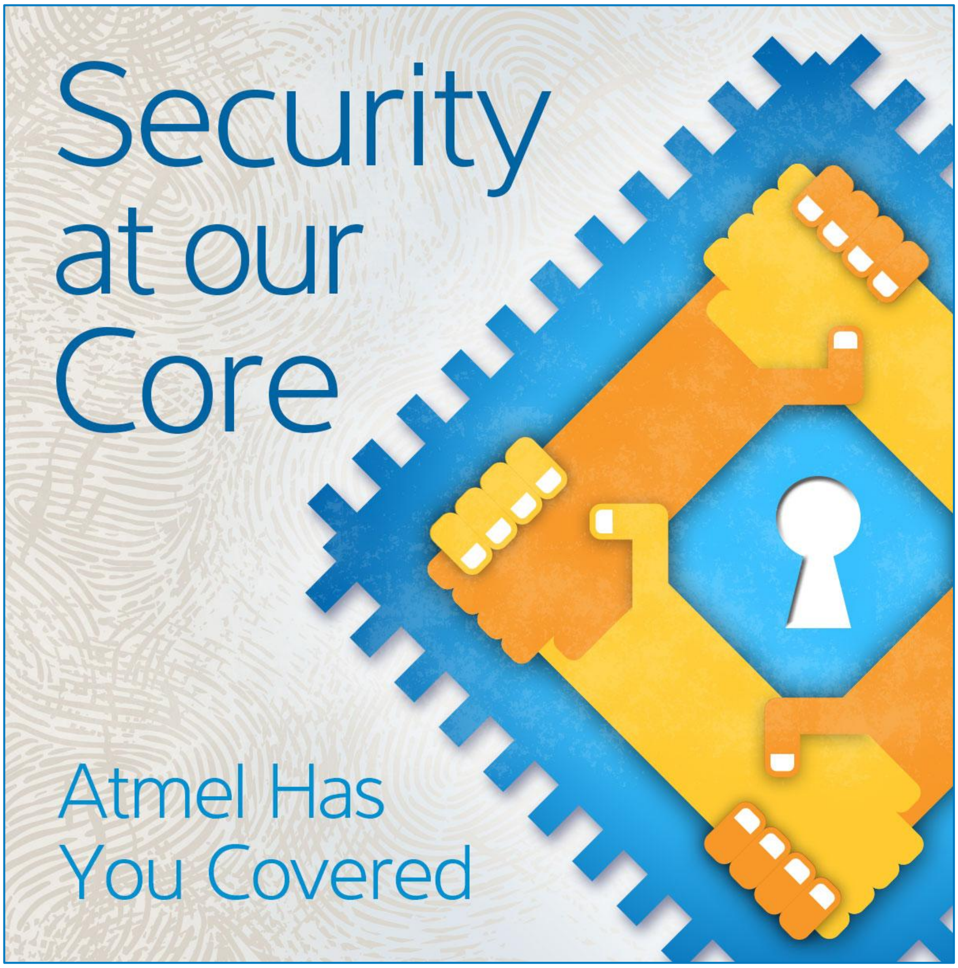
SYMBOL	MIN	NOM	MAX	NOTE
D	2.40	2.50	2.60	
E	6.40	6.50	6.60	
A	0.45	0.50	0.55	
e	1.60	1.70	1.80	
b	1.90	2.00	2.10	
L	2.10	2.20	2.30	
f	0.30	0.40	0.50	
g	0.05	0.15	0.25	
h	2.30	2.40	2.50	
j		4.30	4.40	4.50

1/31/11

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

15 Revision History

Doc. Rev.	Date	Comments
8895CX	01/13/2016	Updated write endurance from write cycles of 100,000 to 400,000 minimum and the 8S1 and 8MA2 package drawings.
8895BX	03/26/2015	Removed preliminary document status.
8895AX	02/15/2015	Document release only under non-disclosure agreement.


Security
at our
Core
Atmel Has
You Covered