ATECC608A-TNGTLS CryptoAuthentication™ Data Sheet

Introduction

The ATECC608A-TNGTLS is a pre-provisioned variant of the ATECC608A. The Trust&GO secure element is part of Microchip's family of generically provisioned security-focused devices. The device configuration was designed to make the secure element applicable to some of the most common use cases in the IoT market, while minimizing the learning and start-up curves associated with security devices.

This data sheet provides the slot and key configuration information that is unique to the ATECC608A-TNGTLS. This information clearly defines the access policies of each of the data zone slots. Only relevant command and I/O operating information has been included. An application section discussing Microchip's hardware and software tools that can aid in developing an application is also provided with additional links to the location of the tools.

Features

- Fully Specified Configuration Zone
- I^{2}C Interface with One-Time Changeable I^{2}C Address
- One Permanent Primary P-256 Elliptic Curve Cryptography (ECC) Private Key Fixed at Manufacturing Time
- One Internal Sign Private Key for Key Attestation
- Three Secondary P-256 ECC Private Keys that Can Be Regenerated by the User
- Signer Public Key from Signer Certificate
- Public Key Validation Support
- One Customizable Symmetric Secret Key Slot
- IO Protection Key Slot to Protect I^{2}C Communication
- Secure Boot Enabled with Customizable Secure Boot Public Key at Time of Manufacture
- ECDH/KDF Key Slot Capable of Being Used with AES Keys and Commands
- X.509 Compressed Certificate Storage
- Available in 8-Pad UDFN and 8-Pin SOIC Packages in 2k or 100 unit production quantities.

Applications

- Secure IoT TLS 1.2 and 1.3 Connections

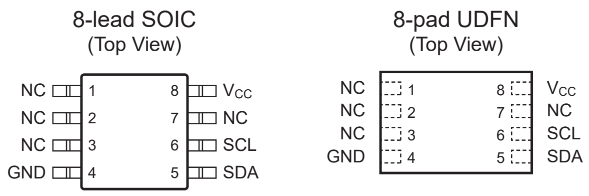
Table of Contents

1. Pin Configuration and Pinouts

Table 1-1. Pin Configuration

Pin	Function
NC	No Connect
GND	Ground
SDA	I^2C Serial Data
SCL	I^2C Serial Clock Input
V_{CC}	Power Supply

Figure 1-1. UDFN and SOIC Pinout



Note: Backside paddle of the UDFN should be connected to GND.

2. EEPROM Memory and Data Zone Access Policies

The EEPROM memory contains a total of 1,400 bytes and is divided into the following zones:

Table 2-1. ATECC608A-TNGTLS EEPROM Zones

Zone	Description	Nomenclature
Configuration	Zone of 128 bytes (1,024 bits) EEPROM that contains: • Device Configuration • Slot Access Policy Information • Counter Values • Device Serial Number • Lock Information The LockConfig byte has already been set. Nothing can be directly written to this zone. The zone can always be read.	Config[a:b] = A range of bytes within a field of the Configuration zone.
Data	Zone of 1,208 bytes (9.7 Kb) split into 16 general purpose read-only or read/write memory slots. The slots are divided in the following way: • Slots 0-7 Contain 36 Bytes • Slot 8 Contains 416 Bytes • Slots 9-15 Contains 72 Bytes The Access Policy information defined by the Configuration zone bytes determines how each slot can be accessed. The Access Policy for each data slot in the ATECC608A-TNGTLS device has already been set and the slot Access Policies defined by the Configuration zone are in full effect. Some slots can be read from or written to while others cannot, depending upon that slot's Access Policy.	Slot[YY] = The entire contents stored in Slot YY of the Data zone.
One Time Programmable (OTP)	Zone of 64 bytes (512 bits) arranged into two blocks of 32 bytes each. For the ATECC608A-TNGTLS, the zone has been preloaded with a predefined value. This zone cannot be modified but can be read at any time. See Section 2.3 for more information	OTP[bb] = A byte within the OTP zone, while OTP[aa:bb] indicates a range of bytes.

Table 2-2. Document Terms

Terms discussed within this document will have the following meanings:

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. This memory is not directly accessible by the user. See Section 3. Static RAM (SRAM) Memory.
Word	A single 4-byte word of data read from or written to a block. The word is the smallest unit of data access.

2.1 ATECC608A-TNGTLS Configuration Zone

The 128 bytes in the Configuration zone contain the manufacturing identification data, general device and system configuration information and access policy 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 majority of these values have been preconfigured and fixed for the ATECC608A-TNGTLS. A discussion of the bytes that are modifiable can be found in Section 2.1.1 Modifiable Configuration Zone Bytes.

The bytes of this zone have been configured as shown in the table below:

Table 2-3. ATECC608A-TNGTLS Configuration Zone Settings

Byte	Name	Configured Value [LSB MSB]	Description
[0:3]	SN[0:3]	`01 23 xx xx`	Part of the serial number value.
[4:7]	RevNum	`00 00 60 02`	Device revision number.
[8:12]	SN[4:8]	`xx xx xx xx 01`	Part of the serial number value.
[13]	AES_Enable	`01`	AES Operations are Enabled.
[14]	I2C_Enable	`xx`	**b[7:1]** Programmed by Microchip and will vary with device. **b[0]** `1` - For I^2C Mode devices
[15]	Reserved	`00`	Set by Microchip will always be 0x00.
[16]	I2C_Address	`6A`	Default 7 bit I^2C address is 0x35.
[17]	Reserved	`00`	Reserved. Must be zero.
[18]	CountMatch	`00`	Counter match function is disabled
[19]	ChipMode	`01`	**b[7:3]** `0x00` Clock Divider mode is High Speed **b[2]** `0` Watchdog Time is set to 1.3s **b[1]** `0` I/O's use Fixed Reference mode **b[0]** `1` Alternate I^2C address mode is enabled
[20:51]	SlotConfig	See Section 2.2.4	Two bytes of access and usage permissions and controls for each slot of the Data zone.

[52:59]	Counter[0]	`FF FF FF FF`
		`00 00 00 00`
			Monotonic Counter 0 is not attached to any keys but can be used as a system counter if so desired.

[60:67]	Counter[1]	`FF FF FF FF`
		`00 00 00 00`
			Monotonic Counter 1 is not attached to any keys but can be used as a system counter if so desired.

[68]	UseLock	`00`	Use Lock Key is disabled.
[69]	VolatileKey Permission	`00`	Volatile Key Permission is disabled.

...........continued

Byte	Name	Configured Value [LSB MSB]	Description
[70:71]	SecureBoot	`03 F7`	**b[15:12]** `0xF` Secure Boot Public Key is stored in Slot 15 **b[11:8]** `0x7` Secure Boot Digest is stored in Slot 7 **b[7:4]** `0x0` must be set to zero **b[3]** `0` Random Nonce is not required but recommended **b[2]** `0` Secure Boot Persistent Latch is disabled **b[1:0]** `0b11` Secure Boot FullDig mode enabled
[72]	KdfIvLoc	`00`	No effect since ChipOptions.KDFPROT does not force encryption in this configuration.
[73:74]	KdfIvStr	`69 76`	No effect since ChipOptions.KDFPROT does not force encryption in this configuration.
[75:83]	Reserved	Zeros	Must be zero.
[84]	UserExtra	`00`	One byte value that can be modified via the `UpdateExtra` command after the Data zone has been locked. Can be written via UpdateExtra only if it has a value of zero.
[85]	UserExtraAdd	`00`	This byte will be the I^{2}C address of the device, if the value of this byte is != `0x00`. If the value is `0x00`, then this value can be written via the `UpdateExtra` command.
[86]	LockValue	`00`	Data zone has been locked therefore this value will be 0x00.
[87]	LockConfig	`00`	Config zone has been locked therefore this value will be 0x00.
[88:89]	SlotLocked	`FF FF`	For the ATECC608A-TNGTLS, the following individual slots may be locked: Slots 2-6, 8, 13 and 15. All other slots are non-writable.
[90:91]	ChipOptions	`0E 60`	**b[15:12]** `0x6` IO Protection Key set to Slot 6 **b[11:10]** `0b00` Output of the KDF function in the clear is allowed but encryption is possible **b[9:8]** `0b00` Output of ECDH master secret in the clear is allowed but encryption is possible **b[7:4]** Must be zero **b[3]** `1` = The Health Test Failure bit is cleared after any time that a command fails as a result of a health test failure. If the failure symptom was transient, then when run a second time the command may pass. **b[2]** `1` The KDF AES mode is enabled **b[1]** `1` IO Protection Key is enabled **b[0]** `0` Power On Self Tests are disabled on wake
[92:95]	X509format	`00 00 00 00`	Certificate Formatting is disabled/ignored.
[96:127]	KeyConfig	See Section 2.2.4	Two bytes of additional access and usage permissions and controls for each slot of the Data zone.

2.1.1 Modifiable Configuration Zone Bytes

No bytes within the Configuration zone can be directly written since the Config zone has already been locked. Several bytes can still be modified through use of other commands.

SlotLocked Bits
For the ATECC608A-TNGTLS, the following individual slots may be locked: Slots 2-6, 8, 13 and 15.These may be locked through the SlotLock mode of the `Lock` command. Each slot where this feature is enabled can be individually locked just once. Once a slot has been locked it can never be modified or unlocked but can still be used based on the Access Policies defined for that slot.

I^{2}C Address Redefinition
This device configuration has been created such that the I^{2}C address can be redefined one time. The `UpdateExtra` command may be used to rewrite byte 85 of the Configuration zone to a new I^{2}C Address. When this byte is set to a non-zero value, the device configuration uses byte 85 as its I^{2}C address instead of the default address. Once this byte has been rewritten, the device must be powered-down or put into Sleep mode before this change takes effect.

Important: If there is no need to change the I^{2}C address then this location should be written with the default I^{2}C address.

User Extra Byte
The UserExtra byte can be used for any desired purpose. This byte can be updated just once with the `UpdateExtra` command.

Counter[0,1]
While the counters are not used by this device, they are not disabled. If so desired, the monotonic counters may be used by the system. Note that the counters are initialized to zero and can count to the maximum value of 2,097,151. The counter value can be incremented or read through use of the `Counter` command. How this counter is used is strictly up to the system and independent of anything else on the device.

2.2 Data Zone and Access Policies

The following sections describe the detailed access policy information associated with each slot. The actual access policy information is stored within the Slot and Key configuration sections in the EEPROM Configuration zone. Each Data zone slot has 2 Slot Configuration Bytes and 2 Key Configuration Bytes associated with it. Together, these four bytes create the “Access Policies” for each slot. The actual type of data stored within the slot is determined by the Access Policies for that slot.

2.2.1 Data Zone Data Types

The following section provides more details on the various types of data capable of being stored in the ATECC608A-TNGTLS data slots.

2.2.1.1 Private Keys

ECC private keys are the fundamental building blocks of ECC Security. These keys are private and unique to each device and can never be read. ECC private keys are randomly generated by the secure element's TRNG and are securely held in slots configured as ECC private keys.

Primary Private Key
This is the primary authentication key. It is permanent and cannot be changed. Each device has its own unique private key.

This key is enabled for two primary elliptic curve functions:

- ECDSA Sign for authentication
- ECDH for key agreement. If encryption of the ECDH output is required, then the IO protection key needs to be first setup. See Section 2.2.1.7 IO Protection Key for setup details.

This private key is the foundation for the generation of the corresponding public key and the X.509 Certificates.

Secondary Private Key
There are additional private keys that can be used for future use cases such as additional service authentication.

These keys are enabled for the following primary elliptic curve functions:

- ECDSA Sign for authentication.
- ECDH for key agreement. If encryption of the ECDH output is required, then the IO protection key needs to be first setup. See Section 2.2.1.7 IO Protection Key for setup details.
- GenKey for overwriting the slot with a new internally generated random private key.

While the primary key and certificates are permanent, these other keys can be overwritten with a new internally generated private key (`GenKey` command mode = 0x04) to enable key deletion, key rotation, and remote provisioning. The keys are also slot lockable (KeyConfig.Lockable bit is set to zero), meaning the `Lock` command can be used in Slot Lock mode to render the current key permanent and prevent it from being changed by the `GenKey` command. When performing key changes, Key Attestation is required to ensure the new key is properly secured in the ATECC608A-TNGTLS device before it can be trusted.

Key Attestation
The private key in slot 1 is configured as an internal sign only key, which means it can only sign messages generated internally by the `GenKey` or `GenDig` commands and cannot be used to sign arbitrary external messages. This feature allows the internal sign key to be used to attest to what keys are in the device and their configuration/status to any system that knows (and trusts) the internal sign public key.

2.2.1.2 Public Keys

Public keys are associated with the ECC private keys. Every ECC private key will have its own unique public key. A couple of slots have been set aside to store public keys for validation purposes. These are often used as secure storage of root-of-trust public keys. The slots for these keys can be operated in two different modes:

- Permanent Public Key - In this mode the required public key should be written to the slot labeled Parent Public Key and the slot locked to make it permanent. The Validated Public Key slot is not used in this mode.
- Securely Updatable Public Key - Here, a parent public key should be written and locked in the Parent Public Key slot. The public key to be validated must then be written to the Validated Public Key slot. Finally, the private key counterpart to the parent public key (off chip) needs to be used to validate the public key to enable its use and prevent unauthorized changes. See Section 2.2.1.2 Validated Public Key for more details on this process.

Parent Public Key
The parent public key is a primary system key generated from an ECC private key that is stored off chip.

Validated Public Key
A validated public key requires that a key be validated before use or invalidated before being updated. Validation and invalidation are done using the `Verify` command in Validate/Invalidate mode. See Section 5.2.5.3 Verify - Validate and Invalidate.

2.2.1.3 Certificates Dynamic Storage

The ATECC608A-TNGTLS storage is centered around keys. X.509 certificates tend to be larger than what will fit into the ATECC608A-TNGTLS slots, so a compressed format is used. This technique may be better called a partial certificate as it stores dynamic certificate information on the device and imposes some limitations. Dynamic information is certificate content that can be expected to change from device to device (e.g., public key, validity dates, etc.). Firmware is expected to have a certificate definition (atcacert_def_t from CryptoAuthLib) with a template of the full X.509 certificate containing static information (data that are the same for all certificates) and instructions on how to rebuild the full certificate from the dynamic information in the compressed certificate.

The following application note documents the compressed certificate format: ATECC Compressed Certificate Definition.

The CryptoAuthLib library also contains the atcacert module for working with compressed certificates.

Device Certificate
The device certificate consists of information associated with the actual end unit. For the ATECC608A-TNGTLS, the device certificate is stored in Slot #10.

Signer Certificate
The signer certificate consists of the information associated with the signer used to sign the device certificate. For the ATECC608A-TNGTLS the signer certificate is stored in Slot #12 The signer public key is also required to complete the full signer certificate.

Signer Public Key
The signer public key is the public key needed to verify the signer and the information that is associated with the signer compressed certificate. For the ATECC608A-TNGTLS, it is stored in Slot #11.

The following table shows all the slots associated with certificates in the ATECC608A-TNGTLS:

Slot	Description
0	Primary private key. The public key can be generated at any time using the `GenKey` command in Mode = 0x00.
10	Device certificate. This is stored here in a compressed format. See Section 4.5.3 Certificate Storage.
11	Signer public key. See Section 4.5.1.1 Public Key Formats.
12	Signer certificate. This is stored in a compressed format. See Section 4.5.3 Certificate Storage.

These certificates are also permanent and the slots they are stored in cannot be changed.

2.2.1.4 Secure Boot

The `SecureBoot` command has been enabled for the ATECC608A-TNGTLS. This allows the system to cryptographically validate its firmware via a boot loader before performing a full boot. This functionality can also be used to validate new firmware images before they're loaded.

The secure boot feature requires establishing a P-256 firmware signing key before it can be used. The private key will be held by the firmware developers for signing the firmware image. The public key needs to be written to the secure boot public key slot and the slot locked to make it permanent.

See Section 5.2.3 SecureBoot Command for full details.

To implement the SecureBoot, several data slots are required.

Secure Boot Digest
The Secure Boot Digest is a 32 byte SHA256 digest calculated over the firmware application code. This digest needs to be updated every time the firmware is updated. For the ATECC608A-TNGTLS, the digest is stored in Slot #7.

Secure Boot Public Key
The Secure Boot public key is used to do a verify function to validate the Secure Boot Digest and signature. The Secure Boot public key is stored in Slot #15.

2.2.1.5 Secret Key

This slot can be used to store a secret 32-byte value or key. This key can be used with the ATECC608A-TNGTLS's symmetric key commands (`GenDig`, `MAC`, `CheckMac`, `KDF`, `SHA/HMAC`, `AES`).

Writing this key requires an encrypted write with the IO protection key as the write key. Therefore, the 2.2.1.7 IO Protection Key must be set before the secret key can be written.

2.2.1.6 AES Key Storage

Commands such as `ECDH` and `KDF` output symmetric keys. These commands can optionally save those keys to a slot for secure storage and use. The AES key storage slot has been set as a destination slot for those keys. Multiple keys are capable of being stored in a slot.

2.2.1.7 IO Protection Key

The `Verify`, `ECDH`, `SecureBoot`, and `KDF` commands can optionally use the IO protection feature to encrypt some parameters and validate (via MAC) some responses. This is to help protect against man-in-the-middle attacks on the physical I^{2}C bus. However, before this feature can be used, the MCU and ATECC608A-TNGTLS need to generate and save a unique IO protection key, essentially pairing the MCU and ATECC608A-TNGTLS devices to each other. The pairing process must happen on first boot.

IO Protection Key Generation:

1. MCU uses random command to generate a random 32-byte IO protection key.
2. MCU saves the IO protection key in its internal Flash.
3. MCU writes IO protection key to the IO protection key slot.
4. MCU slot locks that slot to make the IO protection key permanent.

As a pairing check, the MCU could use the `MAC` command to issue a challenge to the IO protection key and verify the IO protection key stored in Flash matches the one in the ATECC608A-TNGTLS.

2.2.1.8 General Data Storage

A number of slots have been set up to support general public data storage. These slots may be used to store any data that are allowed to be publicly accessible. These slots can always be read and written in the clear.

2.2.2 ATECC608A-TNGTLS Slot Configuration Summary

The ATECC608A-TNGTLS has 16 slots that can be configured for different use cases. Below is a summary of those slots with their configuration and proposed uses for the ATECC608A-TNGTLS:

Slot	Use Case	Description	Primary Configuration
0	Primary private key	Primary authentication key.	Permanent, Ext Sign, ECDH
1	Internal sign private key	Private key that can only be used to attest the internal keys and state of the device. It cannot be used to sign arbitrary messages.	Permanent, Int Sign
2	Secondary private key 1	Secondary private key for other uses.	Updatable, Ext Sign, ECDH, Lockable
3	Secondary private key 2	Secondary private key for other uses.	Updatable, Ext Sign, ECDH, Lockable
4	Secondary private key 3	Secondary private key for other uses.	Updatable, Ext Sign, ECDH, Lockable
5	Secret key	Storage for a secret key.	No Read, Encrypted write (6), Lockable, AES key
6	IO protection key	Key used to protect the I^{2}C bus communication (IO) of certain commands. Requires setup before use.	No read, Clear write, Lockable
7	Secure boot digest	Storage location for secure boot digest. This is an internal function, so no reads or writes are enabled.	No read, No write
8	General data	General public data storage (416 bytes).	Clear read, Always write, Lockable
9	AES key	Intermediate key storage for ECDH and KDF output.	No read, Always write, AES key
10	Device compressed certificate	Certificate primary public key in the CryptoAuthentication™ compressed format.	Clear read, No write
11	Signer public key	Public key for the CA (signer) that signed the device cert.	Clear read, No write

...........continued

Slot	Use Case	Description	Primary Configuration
12	Signer compressed certificate	Certificate for the CA (signer) certificate for the device certificate in the CryptoAuthentication™ compressed format.	Clear read, No write
13	Parent public key or general data	Parent public key for validating/invalidating the validated public key. It can also be used just as a public key or general data storage (72 bytes).	Clear read, Always write, Lockable
14	Validated public key	Validated public key cannot be used (`Verify` command) or changed without authorization via the parent public key.	Clear read, Always write, Validated (13)
15	Secure boot public key	Secure boot public key.	Clear read, Always write, Lockable

2.2.3 Slot Configuration Terminology

The following section provides a set of terms used to discuss configuration options. The terms are arranged alphabetically.

Term	Description
AES Key	Slot can be used as a key source for `AES` commands. The AES key is 128 bits in width for the ATECC608A-TNGTLS.
Always Write	Slot can be written in the clear with the write command.
Clear Read	Slot is considered public (non-secret) and its contents can be read in the clear with the read command.
ECDH	Elliptic Curve Diffie Hellman. Private key can be used with the `ECDH` command.
Encrypted Write	Slot can only be written using an encrypted write based on the write key specified.
Ext Sign	Private key can be used to sign external (arbitrary) messages.
Int Sign	Private key can be used to sign internal messages generated by the `GenKey` or `GenDig` commands. Used to attest the device's internal keys and configuration.
Lockable	Slot can be locked at some point in the future. Once locked, the slot contents cannot be changed (read/use only).
No Read	Slot is considered secret and its contents cannot be read with the read command. Private keys and symmetric secrets should always be configured as No Read.
No Write	Slot cannot be changed with the write command.
Permanent	Private key is permanent/unchangeable. It is internally generated during factory provisioning.
Updatable	Private key can be overwritten later with a new random internally generated private key. Its initial value is internally generated during factory provisioning.
Validated	Public key can only be used with the `Verify` command once it has been validated by the parent public key.

2.2.4 ATECC608A-TNGTLS Detailed Slot Access Policies

The following tables provide a more detailed description of each slot key and slot configuration information along with what commands and command modes can be run using this slot.

Table 2-4. Slot 0 Configuration Information

Slot	Configuration Value		Description of Enabled Features
0	Key:	0x0053	**Primary Private Key** • Contains P256 NIST ECC private key (KeyType = 0x4)[1] • The corresponding public key can always be generated
	Slot:	0x0085	• Slot is secret • Can sign external messages • Can use with `ECDH` command • Random Nonce is required
	Valid commands		• GenKey - Public Key Generation • Sign - External Messages • ECDH - Create a Shared Secret

Table 2-5. Slot 1 Configuration Information

Slot	Configuration Value		Description of Enabled Features
1	Key:	0x0053	**Internal Sign Private Key** • Contains P256 NIST ECC private key (KeyType = 0x4)[1] • The corresponding public key can always be generated
	Slot:	0x0082	• Slot is secret • Can sign internal messages generated by GenDig or GenKey • ECDH disabled • Random Nonce is required
	Valid commands		• GenKey - Public Key Generation • Sign - Internal Messages generated by GenDig or GenKey

Table 2-6. Slot and Key Configuration Slots 2-4

Slot	Configuration Value		Description of Enabled Features
2,3 or 4	Key:	0x0073	**Secondary Private Keys 1-3** • Contains P256 NIST ECC private key (KeyType = 0x4)[1] • The corresponding public key can always be generated • When using this key, a random nonce is always required • This slot can be individually locked
	Slot:	0x2085	• GenKey can be used to generate a new ECC private key in this slot prior to locking • Slot is secret • Can sign external messages • Can use with `ECDH` command
	Valid commands		• GenKey - Private Key Regeneration • GenKey - Public Key Generation • Sign - External Messages • ECDH - Create a Shared Secret • Lock - SlotLock Mode

Table 2-7. Slot 5 Configuration Information

Slot	Configuration Value		Description of Enabled Features
5	Key:	0x0038	**Secret Key** • Slot can store up to 2 AES 128-bit (16 byte) symmetric keys (KeyType = 0x6)(1) • This slot can be individually locked
	Slot:	0x468F	• New symmetric key can be written with an encrypted write only • Key in slot 6 is the key used to encrypt the write • The contents of the slot are secret • Slot cannot be used for the `CheckMac Copy` command
	Valid commands		• Write - Data Zone - Encrypted Write • AES - Encrypt / Decrypt Modes • `MAC` Command • `CheckMac` Command • Lock - SlotLock mode

Table 2-8. Slot 6 Configuration Information

Slot	Configuration Value		Description of Enabled Features
6	Key:	0x007C	**IO Protection Key** • Can contain a SHA256 symmetric key or other data. (KeyType = 0x7)(1). If the IO protection key is not used, this slot can be used for other data. • A random nonce is required when this key is used. • This slot can be individually locked
	Slot:	0x0F8F	• Data can be written in the Clear. • The contents of this slot are secret and cannot be read • Slot cannot be used for the `CheckMac Copy` command
	Valid commands		• Clear Text Write to slot 6. • Write - Encrypted Where this key is the Encryption Key • `MAC` Command • Lock - SlotLock mode

Table 2-9. Slot 7 Configuration Information

Slot	Configuration Value		Description of Enabled Features
7	Key:	0x001C	**Secure Boot Digest** • This slot is designated to be used for other data (KeyType = 0x7)(1)
	Slot:	0x8F9F	• This slot cannot be directly written or read • This slot is secret and cannot be used by the `MAC` command • This slot cannot be used for `CheckMac Copy` command
	Valid commands		• SecureBoot - FullCopy mode • SecureBoot - FullStore(Digest)

Table 2-10. Slot 8 Configuration Information

Slot	Configuration Value	Description of Enabled Features
8	Key: 0x003C	**General Data** • This slot is designated for use with general data (KeyType = 0x7)[1] • Slot is lockable
	Slot: 0x0F0F	• Clear text writes and reads are permitted to this slot • Slot cannot be used for the `CheckMac Copy` command
	Valid commands	• Write - Clear Text • Read - Clear Text • GenDig - Data Source • `MAC` Command • Lock - SlotLock mode

Table 2-11. Slot 9 Configuration Information

Slot	Configuration Value	Description of Enabled Features
9	Key: 0x001A	**AES Key** • Slot can store up to four AES 128-bit symmetric keys (KeyType = 0x6)[1]
	Slot: 0x0F8F	• Clear text writes are allowed to this slot • This slot is secret • Slot cannot be used for the `CheckMac Copy` command
	Valid commands	• Write - Clear Text • AES - Encrypt / Decrypt (Source Keys) • `MAC` Command

Table 2-12. Slot 10 Configuration Information

Slot	Configuration Value	Description of Enabled Features
10	Key: 0x001C	**Device Compressed Certificate** • Slot defined to store other data. (KeyType = 0x7)[1]
	Slot: 0x8F0F	• Data cannot be overwritten • Data can be read in the clear
	Valid commands	• Read - Clear Text • GenDig - Data Source • `MAC` Command

Table 2-13. Slot 11 Configuration Information

Slot	Configuration Value		Description of Enabled Features
11	Key:	0x0010	**Signer Public Key** • Slot is defined for ECC key (KeyType - 0x4)(1) • ECC key is a public key
	Slot:	0x8F0F	• Data cannot be overwritten • Data can be read in the clear
	Valid commands		• Read - Clear Text • GenDig - Data Source • Verify Command • MAC Command • GenKey - Public Digest Mode

Table 2-14. Slot 12 Configuration Information

Slot	Configuration Value		Description of Enabled Features
12	Key:	0x001C	**Signer Compressed Certificate** • Slot defined to store other data. (KeyType = 0x7)(1)
	Slot:	0x8F0F	• Data cannot be overwritten • Data can be read in the clear
	Valid commands		• Read - Clear Text • GenDig - Data Source • MAC Command

Table 2-15. Slot 13 Configuration Information

Slot	Configuration Value		Description of Enabled Features
13	Key:	0x0030	**Parent Public Key or General Data** • Slot is defined for ECC key (KeyType - 0x4)(1) • Slot is lockable
	Slot:	0x0F0F	• Slot can be written in the clear (unless locked) • Slot can always be read
	Valid commands		• Write - Clear Text • Read - Clear Text • Lock - SlotLock mode • Verify Command • MAC Command • GenDig - Data Source

Table 2-16. Slot 14 Configuration Information

Slot	Configuration Value		Description of Enabled Features
14	Key:	0x0012	**Validated Public Key** • Slot is defined for ECC key (KeyType - 0x4)(1) • Public key can be used by the `Verify` command if the key has been validated
	Slot	0x1F0D	• Write mode set to PubInvalid • Can write to slot if key is invalidated first • Slot can always be read in the clear
	Valid commands		• Write - Clear Text (slot must first be Invalidated) • Read - Clear Text • Verify - Validate/Invalidate • Verify - Stored Mode

Table 2-17. Slot 15 Configuration Information

Slot	Configuration Value		Description of Enabled Features
15	Key:	0x0030	**Secure Boot Public Key** • Slot is defined for ECC key (KeyType - 0x4)(1) • Slot is lockable
	Slot:	0x0F0F	• Always writable unless locked • Slot can always be read
	Valid commands		• Write - Clear Text • Read - Clear Text • Lock - SlotLock mode • `MAC` Command • GenDig - Data Source

Note:
1. KeyType is specified by Key Configuration bits [4:2] for each slot.

2.3 ATECC608A-TNGTLS EEPROM One Time Programmable (OTP) Zone

The OTP zone of 64 bytes (512 bits) is part of the EEPROM array and can be used for read-only storage. It is organized as two blocks of 32 bytes each. For the ATECC608A-TNGTLS, the OTP zone is shipped pre-locked and contains the following information:

```
52 73 75 79 35 59 4A 68 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

The data byte values written into the OTP zone are always available for reading using either 4 or 32 byte reads but can never be modified.

3. Static RAM (SRAM) Memory

The device also includes an SRAM array that is used to store the input command or output result, nonces, intermediate computation values, ephemeral keys, the SHA context, etc. The contents of the SRAM can never be read directly; only used internally by the secure element. The entire contents of this memory are invalidated whenever the device goes into sleep mode or the power is removed.

3.1 TempKey

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

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

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

Table 3-1. TempKey Flags

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

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

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

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

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

3.2 Message Digest Buffer

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

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

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

The Message Digest Buffer is cleared to zero during power-up, sleep, brown-out, watchdog expiration or tamper detection. The Message Digest buffer is generally cleared after the execution of most commands with the exception of the `Nonce` and `SHA` commands. It can only be used (read) in a single command without reloading as it is always cleared upon use.

3.3 Alternate Key Buffer

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

The Alternate Key Buffer is cleared to zero during power-up, sleep, brown-out, watchdog expiration or tamper detection.

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

3.4 SHA Context Buffer

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

This SHA context buffer is neither read nor written by any other ATECC608A-TNGTLS command and is therefore not disrupted regardless of the success or failure of the execution of any other commands. Like all SRAM memory in the device, it is cleared to zero during power-up, sleep, brown-out, watchdog expiration or tamper detection.

4. General Command Information

The following sections provide some general information on the basic I/O transactions, command structure, error codes, memory addressing and formatting of keys and signatures that are used in the ATECC608A-TNGTLS.

4.1 I/O Transactions

The ATECC608A-TNGTLS utilizes the I^2C protocol to communicate with a host microcontroller. Security commands are sent to the device and responses received from the device within a transaction that is constructed in the following way:

Table 4-1. I/O Transaction Format

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 4.2 Command Packets for general command packet information or 5. Device Commands for specific parameters for each command.
N-1, N	Checksum	CRC-16 verification of the count and packet bytes. The CRC polynomial is 0x8005. Prior to the start of the CRC calculation the CRC register is initialized to zero. After the last bit of the count and packet have been transmitted, the internal CRC register must have a value that matches the checksum bytes in the block. The first CRC byte transmitted (N-1) is the Least Significant Byte of the CRC value, so the last byte of the group is the Most Significant Byte of the CRC.

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

4.2 Command Packets

The command packet is broken down as shown in Table 4-2:

Table 4-2. Command Packets

Byte	Name	Meaning
0	Opcode	The command code. See Section 5. Device Commands
1	Param1	The first parameter; always present.
2 – 3	Param2	The second parameter; always present.
0-155	Data	Optional remaining input data.

After the ATECC608A-TNGTLS 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 transitions on the I^2C SDA input signal.

4.3 Status/Error Codes

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

- Count
- Packet
- Two byte CRC

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

After the wake and 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. For a four bytes length of that group, the codes returned are detailed in Table 4-3. Some commands return more than four bytes when they execute successfully. The resulting packet description is listed in the Section 5. Device Commands.

CRC errors are always returned before any other type of error. They indicate that an 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 4-3. 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 response did not match the expected value.
Parse error	0x03	Command was properly received but the length, command opcode, or parameters are illegal regardless of the state (volatile and/or EEPROM configuration) of the ATECC608A-TNGTLS. Changes in the value of the command bits must be made before it is re-attempted.
ECC Fault	0x05	A computation error occurred during ECC processing that caused the result to be invalid. Retrying the command may result in a successful execution.
Self Test error	0x07	There was a Self Test error and the chip is in Failure mode waiting for the failure to be cleared.
Health Test error	0x08	There was a random number generator Health Test error and the chip fails subsequent commands requiring a random number until it is cleared.
Execution error	0x0F	Command was properly received but could not be executed by the device in its current state. Changes in the device state or the value of the command bits must be made before it is re-attempted.
After Wake, Prior to First command	0x11	Indication that ATECC608A-TNGTLS has received a proper Wake token.
Watchdog About to Expire	0xEE	There is insufficient time to execute the given command before the Watchdog Timer expires. 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 ATECC608A-TNGTLS and should be re-transmitted by the I/O driver in the system. No attempt was made to parse or execute the command.

4.4 Address Encoding

The following subsections provide detailed information on how to address the various memory zones of the ATECC608A-TNGTLS device.

4.4.1 Configuration Zone Addressing

The Configuration zone can be accessed either 4 or 32 bytes at a time. Individual bytes cannot be accessed. The Configuration zone address is a 2-byte (16-bit value). Only the lowest five bits of the address word are used in addressing of the Configuration zone. For the ATECC608A-TNGTLS device, these addresses can only be used with the read command.

Table 4-4. Address Format

Byte 1: Addr[15:8]	Byte 0: Addr[7:0]		
Unused	Unused	Block	Offset
Addr[15:8]	Addr[7:5]	Addr[4:3]	Addr[2:0]

Table 4-5. Configuration Zone Addresses

Block # (Addr[4:3])	Offset Value (Addr[2:0])							
	000	001	010	011	100	101	110	111
00	[0:3]	[4:7]	[8:11]	[12:15]	[16:19]	[20:23]	[24:27]	[28:31]
01	[32:35]	[36:39]	[40:43]	[44:47]	[48:51]	[52:55]	[56:59]	[60:63]
10	[64:67]	[68:71]	[72:75]	[76:79]	[80:83]	[84:87]	[88:91]	[92:95]
11	[96:99]	[100:103]	[104:107]	[108:111]	[112:115]	[116:119]	[120:123]	[124:127]

4.4.2 OTP Zone Addressing

The One Time Programmable (OTP) zone can be accessed either 4 or 32 bytes at a time. The zone has a total of 64 bytes. Individual bytes cannot be accessed. The OTP zone address is a 2-byte (16-bit value). Only the lowest four bits are used in addressing.

For the ATECC608A-TNGTLS device, these addresses can only be used with the read command.

Table 4-6. Address Format

Byte 1: Addr[15:8]	Byte 0: Addr[7:0]		
Unused	Unused	Block	Offset
Addr[15:8]	Addr[7:4]	Addr[3]	Addr[2:0]

Table 4-7. OTP Zone Byte Addresses

Block # (Addr[3])	Block Offset Value (Addr[2:0])							
	000	001	010	011	100	101	110	111
0	[0:3]	[4:7]	[8:11]	[12:15]	[16:19]	[20:23]	[24:27]	[28:31]
1	[32:35]	[36:39]	[40:43]	[44:47]	[48:51]	[52:55]	[56:59]	[60:63]

4.4.3 DataZone Addressing

Read/Write access to the Data zone is much more complex than the Configuration and OTP zones. There are a total of 16 slots and the size of the slots vary. Each slot's access policies individually control whether or not a slot has the ability to be read or written.

For the ATECC608A-TNGTLS:

- Data Slots 8-9, 13 and 15 can be written as clear text.
- Data Slots 5-6 can be written with encrypted text.
- Data Slots 8 and 10-15 can be read as clear text.
- Any slots not specified cannot be read or written.

Table 4-8. Address Format by Data Slot Size

Data Zone	Byte 1 Addr[15:8]		Byte 0: Addr[7:0]		
	Unused	Block	Unused	Slot	Offset
Data Slots[7:0]	Addr[15:9]	Addr[8]	Addr[7]	Addr[6:3]	Addr[2:0]
Data Slot[8]	Addr[15:12]	Addr[11:8]	Addr[7]	Addr[6:3]	Addr[2:0]
Data Slot[15:9]	Addr[15:10]	Addr[9:8]	Addr[7]	Addr[6:3]	Addr[2:0]

Data Slots[7:0]
To fully access one of these slots require two 32-byte accesses or nine 4-byte accesses

Table 4-9. Data Zone Addresses Slots 0-7

Slot# (Addr[6:3])	Block # (Addr[8])	Block Offset Value (Addr[2:0])							
		000	001	010	011	100	101	110	111
0x0 to 0x7	00	[0:3]	[4:7]	[8:11]	[12:15]	[16:19]	[20:23]	[24:27]	[28:31]
	01	[32:35]	Not Valid	Not Valid	Not Valid	Not Valid	Not Valid	Not Valid	Not Valid

Data Slot[8]
To fully access this slot require thirteen 32-byte accesses or 104 4-byte accesses or a combination of the two methods.

Table 4-10. Data Zone Addressing Slot 8

Slot# (Addr[6:3])	Block # (Addr[8])	Block Offset Value (Addr[2:0])							
		000	001	010	011	100	101	110	111
0x8	0x0	[0:3]	[4:7]	[8:11]	[12:15]	[16:19]	[20:23]	[24:27]	[28:31]
	0x1	[32:35]	[36:39]	[40:43]	[44:47]	[48:51]	[52:55]	[56:59]	[60:63]
	...	...	...	...	...	...	...	...	...
	0xC	[384:387]	[388:391]	[392:395]	[396:399]	[400:403]	[404:407]	[408:411]	[412:415]

Data Slots[15:9]
To fully access these slots requires three 32-byte accesses or eighteen 4-byte accesses or a combination of the two methods.

Table 4-11. Data Zone Addressing Slots 9-15

Slot# (Addr[6:3])	Block # (Addr[8])	Block Offset Value (Addr[2:0])							
		000	001	010	011	100	101	110	111
0x9 to 0xF	00	[0:3]	[4:7]	[8:11]	[12:15]	[16:19]	[20:23]	[24:27]	[28:31]
	01	[32:35]	[36:39]	[40:43]	[44:47]	[48:51]	[52:55]	[56:59]	[60:63]
	10	[64:67]	[68:71]	Not Valid	Not Valid	Not Valid	Not Valid	Not Valid	Not Valid

4.5 Formatting of Keys, Signatures and Certificates

The following sections provide detailed formatting information for ECC keys, Signatures and Compressed certificates.

4.5.1 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. Microchip recommends all pad bytes be set to zero for consistency.

- ECC private keys appear to the user only as the input parameter to the `PrivWrite` command. This parameter is always 36 bytes in length and the first four bytes (32 bits) are all pad bits.
  ECC public keys appear as the input or output parameters to several commands, and they can also be stored in EEPROM. They are composed of an X value first on the bus or in memory, followed by a Y value. They are formatted differently depending on the situation as noted below:
- **The public key is an output of the `GenKey` command or an input to the `Verify` command**:
  32 bytes of X, then 32 bytes of Y. (36 bytes) There are no pad bytes.
- **Write command**:
  Public keys can be written directly to the EEPROM using the write command and are always 72 bytes long, formatted as follows: 4-pad bytes, 32 bytes of X, four pad bytes, then 32 bytes of Y.
- **`GenKey` command**:
  SHA Message: Public keys can be hashed and placed in TempKey by the `GenKey` command. The SHA message contains various bytes that are independent of the size of the key. These are followed by 25 bytes of pad, followed by 32 bytes of X, then by 32 bytes of Y.
- **`Verify` command**:
  SHA Message: When used to validate a stored public key, the `Verify` command expects an input signature created over a SHA-256 digest of a key stored in memory. Such an inner SHA calculation is always performed over 72 bytes formatted as they are stored in EEPROM as 4-pad bytes, 32 bytes of X, 4-pad bytes, then 32 bytes of Y.

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

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

4.5.1.1 Public Key Formats

The ATECC608A-TNGTLS works with the P-256 elliptic curve public keys in two formats. The following example illustrates those two formats in detail.

For the following examples, we'll use a sample public key, with the X and Y integers expressed as fixed-width big-endian unsigned integers:

```
X: b2be345ad7899383a9aab4fb968b1c7835cb2cd42c7e97c26f85df8e201f3be8
Y: a82983f0a11d6ff31d66ce9932466f0f2cca21ef96bec9ce235b3d87b0f8fa9e
```

Command Public Key Format
Any command that returns a public key (GenKey) or accepts a public key as a parameter (Verify and ECDH) will format the public key as the X and Y big-endian unsigned integers concatenated together for a total of 64 bytes.

For example:
```
b2be345ad7899383a9aab4fb968b1c7835cb2cd42c7e97c26f85df8e201f3be8
a82983f0a11d6ff31d66ce9932466f0f2cca21ef96bec9ce235b3d87b0f8fa9e
```

Stored Public Key Format
When storing a public key in a slot for use with the Verify or SecureBoot commands, the X and Y integers will be padded out to 36 bytes and concatenated together for a total of 72 bytes.

For example:
```
00000000b2be345ad7899383a9aab4fb968b1c7835cb2cd42c7e97c26f85df8e201f3be8
00000000a82983f0a11d6ff31d66ce9932466f0f2cca21ef96bec9ce235b3d87b0f8fa9e
```

Note: Only slots 8-15 are large enough to hold a public key.

Stored Validated Public Key Format
A validated or invalidated public key format is the same as a stored public key format with the exception of the four Most Significant bits of the LSB. If a key is validated, the Least Significant Nibble will be 0x5 and 0xA if invalidated. These values can be changed by the `Verify` command in Validate or Invalidate mode. When written, the key will be initially invalidated.

Example Validated Public Key:
```
50000000b2be345ad7899383a9aab4fb968b1c7835cb2cd42c7e97c26f85df8e201f3be8
00000000a82983f0a11d6ff31d66ce9932466f0f2cca21ef96bec9ce235b3d87b0f8fa9e
```

Example Invalidated Public Key:
```
A0000000b2be345ad7899383a9aab4fb968b1c7835cb2cd42c7e97c26f85df8e201f3be8
00000000a82983f0a11d6ff31d66ce9932466f0f2cca21ef96bec9ce235b3d87b0f8fa9e
```

Note: Only slots 8-15 are large enough to hold a public key.

4.5.2 Signature Format

The ECDSA signature that is generated and output by the `Sign` command or input to the `Verify` or `SecureBoot` command is always 64 bytes in length. The signature is divided into R and S components. Both components are 32 bytes in length and R always appears before S on the bus. Each portion of the signature appears MSB first on the bus, meaning the MSB of the signature is in the lowest memory location.

Example R/S Signature
Any command that returns a signature (`Sign`) or accepts a signature as a parameter (`Verify` and `SecureBoot`) will format the signature as the R and S big-endian unsigned integers concatenated together for a total of 64 bytes.

For example:
```
R: 7337887F8C39DF79FD8BF88DDFBFB9DB15D7B1AD68196AE3FB0CE5BFA2842DF3
S: 72868A43A42831E950E1DA9F73B29F5C0ED8A96B2889E3CBBE8E61EA6C67F673
```

4.5.3 Certificate Storage

The amount of storage required for a full X.509 Certificate within the device can rapidly use up multiple EEPROM memory slots. Depending on the actual application, it may or may not be desirable to use these slots for certificate storage. Due to these memory limitations, Microchip has defined an encoding that allows for a full X.509 Certificate to be reconstructed from a minimal amount of information.

The host system would actually be responsible for reconstructing the full X.509 Certificate but how to do this will be determined by the data stored in the encoded certificate. Data that are common to all devices for a given system can readily be stored in the host system. Other data can readily be calculated or extracted from data that are already stored in the device. Table 4-12 indicates the type of data that are stored in an X.509 Certificate and how it can be encoded to fit into a single 72-byte slot.

Table 4-12. Certificate Storage

X.509 Certificate		Encoded Certificate		
X.509 Element	**Size (Bytes)**	**Encoded Certificate Element**	**Device Cert (Bits)**	**Signer Cert (Bits)**
Serial Number	8-20	Serial number source	4	4
Issue Date	13	Compressed format	19	19
Expiry Date	13	# of years before expiration	5	5
Signer ID[2]	4	ID of the specific signer used to sign the certificate (device cert) or of the signer itself (signer cert)	16	16
AuthorityKeyIdentifier	20	SHA1 HASH of the authority public key	0	0
SubjectKeyIdentifier	20	SHA1 HASH of the subject Public Key	0	0
Signature R	32	Stored in device	256	256
Signature S	32	Stored in device	256	256
Public Key X[1]	32	Calculated from the private key or stored in the device[1]	0	256
Public Key Y[1]	32	Calculated from the private key or stored in the device[1]	0	256
n/a	0	Cert format	4	4
n/a	0	Template ID	4	4
n/a	0	Chain ID	4	4
n/a	0	Reserved/User Defined	8	8
Total	(206-218 bytes)		576 bits (72 bytes)	1088 bits (136 bytes)

Note:
1. For the device certificate, the device public key can be regenerated from the private key. For the signer certificate, the public key is typically stored in a separate slot.
2. For the device certificate, the ID of the signer used to sign the certificate is stored. For the signer certificate, the actual ID of the signer is stored so that the device can identify it.

Slot 8 contains a total of 416 bytes. Depending on the size of the serial number stored in the cert, it may or may not be possible to store two complete certificates. Often within devices where a chain of trust has been created, the device certificate, the signer certificate, and the signer public key must be stored within the device.

For more information, see the Compressed Certificate Definition application note.

5. Device Commands

The following section details all of the commands broken out by Command mode that are allowed in the ATECC608A-TNGTLS. The commands have been broken into three categories:

1. **General Device Commands**
   These commands fall into two categories:
   - General device access commands that are used to send data to the device or retrieve data but typically do not perform any cryptographic functions.
   - General cryptographic commands that can be used by the device or the system but typically do not operate on specific data slots.
2. **Asymmetric Cryptography Commands**
   These commands perform asymmetric cryptographic operations, such as key generation, message signing and message verification that utilize an ECC public or private key. These commands are limited to use on ECC Data zone slots.
3. **Symmetric Cryptography Commands**
   These commands perform a symmetric cryptographic function, such as generating a digest or MAC, key derivation or AES encryption and decryption.

Input Parameters for all Commands
Multibyte input parameters are shown as big-endian (MSB first) values in the input parameters tables unless otherwise specified. Note that the ATECC608A-TNGTLS device actually expects the data to be sent little-endian (LSB first).

5.1 General Device Commands

The following table provides a summary of the general device commands:

Table 5-1. General Device Commands

Command Name	Opcode	Description
`Counter`	0x24	Increments and reads the monotonic counters.
`Info`	0x30	Used to read revision and status information from the device.
`Lock`	0x17	Used to lock the individual lockable slots in the device.
`Nonce`	0x16	Used to generate or pass a number used once into the device.
`Random`	0x1B	Used to generate a 32-byte random number used by the system.
Read	0x02	Used to read various zones of the device.
`SelfTest`	0x77	Tests the various internal cryptographic computation elements.
`SHA`	0x47	Computes a SHA-256 or HMAC digest for general purpose use by the system.
`UpdateExtra`	0x20	Updates bytes 84 or 85 within the Configuration zone after the Configuration zone is locked.
Write	0x12	Used to write 4 or 32 bytes to the device, with or without authentication and encryption.

5.1.1 `Counter` Command

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

For the ATECC608A-TNGTLS, the counters are not attached to any keys but may still be used by the system. Each count is set to its default value and can count to the maximum value.

Table 5-2. Input Parameters Count

Opcode (1 Byte)	Mode (1 Byte)	KeyID (2 Bytes)	Description
0x24	0x00	0x00 00	Reads the value of Counter[0]
		0x00 01	Reads the value of Counter[1]
	0x01	0x00 00	Increments the value of Counter[0]
		0x00 01	Increments the value of Counter[1]

Table 5-3. Output Response Count

Name	Size	Description
Count	4 bytes	Counter value in binary if the command succeeds.
Response	1 byte	Error code if the command fails.

5.1.2 `Info` Command

The `Info` command is used to read the status and state of the device. This information is useful in determining errors or to operate various commands.

5.1.2.1 Info - Revision

The Revision mode of the `Info` command reads back the silicon revision of the ATECC608A-TNGTLS. This information is hard coded into the device. This information may or may not be the same as what is read back in the Revision bytes shown in the Configuration zone.

Table 5-4. Input Parameters - Info Revision

Opcode (1 Byte)	Mode (1 Byte)	Param (2 Bytes)	Data (0 Bytes)	Description
0x30	0x00	0x00 00	—	Returns silicon revision

Table 5-5. Output Response - Info Revision

Name	Response	Description
Response	00 00 60 vv	Revision Info. 0x60 indicated ATECC608A-TNGTLS. vv is the most recent silicon version.

5.1.2.2 Info - KeyValid

The KeyValid mode is useful to determine if an ECC private or public key that is stored is a valid ECC key. If the PublicKey cannot be output, then the info returned by this command is not useful. If the KeyType is not ECC, the output of this command is also not useful.

For the ATECC608A-TNGTLS device, the keys stored in Slots 1-4, 11 and 13-15 are ECC keys that can be checked with the KeyValid mode of the `Info` command.

Table 5-6. Input Parameters - Info KeyValid

Opcode (1 Byte)	Mode (1 Byte)	Param (2 Bytes)	Data (0 Bytes)	Description
0x30	0x01	0x00 0[Slot]	—	Returns whether the slot contains a valid ECC private or public key.

Table 5-7. Output Response - Info KeyValid

Name	Size (4 Bytes)	Description
Response	0x00 00 00 00	ECC key is invalid
	0x00 00 00 01	ECC key is valid

5.1.2.3 Info - Device State

The current status of the device is returned with the `Info` command in this mode. The status bits are useful in determining the current state of the device and may be useful in determining why a given command fails or if a command can be executed.

Table 5-8. Input Parameters - Info Device State

Opcode (1 Byte)	Mode (1 Byte)	Param (2 Bytes)	Data (0 Bytes)	Description
0x30	0x02	0x00 00	—	Returns device state

Table 5-9. Output Response - Info Device State

Name	Response	Description
Response	0x00 00 Byte[1] Byte[0]	

Table 5-10. Status Flags

Byte #	Bit #	Name	Description
0	7	TempKey.NoMacFlag	`0`: NoMacFlag is invalid `1`: NoMacFlag is valid
	6	TempKey.GenKeyData	`0`: GenKeyData is invalid `1`: GenKeyData is valid
	5	TempKey.GenDigData	`0`: GenDigData is invalid `1`: GenDigData is valid
	4	TempKey.SourceFlag	`0`: TempKey is fixed source `1`: Temp Key is from random source
	3:0	TempKey.KeyID	TempKey Key SlotID
1	7	TempKey.Valid	`0`: TempKey is not valid `1`: TempKey is valid
	6:3	AuthComplete.KeyID	Authorization Key SlotID
	2	AuthComplete.Valid	`0`: Authorization invalid `1`: Authrorization valid
	1:0	Unused	`2'b00`

5.1.3 `Lock` Command

For the ATECC608A-TNGTLS, the Configuration zone has already been locked and the access policies of the Data zone have already been enforced. However, several of the data slots can still be updated through the use of other commands. If so desired, some of these slots can be permanently locked from future updates by using the Slot Locking mode of the `Lock` command.

5.1.3.1 SlotLock

The Slot Lock mode of the `Lock` command is used to individually lock slots. Any slot that has the Lockable bit set and has not previously been slot-locked can be locked to prevent any further updates. This process is not reversible. Once a slot has been locked, it is permanently locked. In this mode of operation, the Summary CRC and Data fields are ignored.

For the ATECC608A-TNGTLS, the following individual slots may be locked: Slots 2-6, 8, 13 and 15.

Table 5-11. SlotLock Input Parameters

Opcode (1 Byte)	Mode(1 Byte)	Summary CRC (2 Bytes)	Data (0 Bytes)	Description
0x17	8’b00[bb_bb]10	0x00 00	—	Individual Slot Lock

Table 5-12. Slot Lock Output

Name	Size	Description
Response	1 byte	0x00 if the command successfully locks the slot. An error code will output if the command fails.

5.1.4 `Nonce` Command

The `Nonce` command generates a nonce (Number used Once) for use by a subsequent command by combining a random number (which can be generated internally or externally) with an input value from the system. The resulting nonce is stored internally in three possible buffers: TempKey, Message Digest Buffer, and Alternate Key Buffer. Instead of generating a nonce, a value may be passed to the device if so desired.

5.1.4.1 Nonce - Random

When the `Nonce` command is run in Random mode, it generates a new nonce based on the input values shown in the tables below. If Param2 is 0x00 00, then a new random number is generated based on the internal RNG. If Param2 is 0x80 00, a value stored in TempKey is used to generate a new nonce instead and the random number generator is not run. TempKey must be valid prior to running the `Nonce` command in this case. Upon completion, the TempKey.SourceFlag is set to Rand.

It is recommended that the 20 bytes of data sent to the device be generated from a random source to prevent replay attacks.

Table 5-13. Random Nonce Input Parameters

Opcode	Mode	Param2	Data	Description
0x16	0x00 or 0x01	0x00 00	20 bytes	• 32 bytes written to TempKey • Output is from the RNG • SHA256 digest includes the random number
	0x00 or 0x01	0x80 00	20 bytes	• 32 bytes written to TempKey • Output is the SHA256 Hash value • SHA256 digest includes the TempKey

Table 5-14. Random Nonce Output Response

Name	Input Param2	Size	Description
Response	0x00 00	32 bytes	Random number
	0x80 00	32 bytes	New TempKey value

Note:
1. TempKey.SourceFlag set to 0 on success.
2. TempKey.Valid set to 1 on success.

Table 5-15. Nonce SHA256 HASH Calculation

# of Bytes	Input Data Param2 = 0x00 00	Input Data Param2 = 0x80 00
32	RandOut from random number generator	TempKey value from previous command
20	NumIn from input stream	NumIn from input stream
1	Opcode (always 0x16)	Opcode (always 0x16)
1	Mode (0x00 or 0x01)	Mode (0x00 or 0x01)
1	LSB of Param2 (always 0x00)	LSB of Param2 (always 0x00)

5.1.4.2 Nonce - Fixed

A fixed nonce is passed to the device and stored in one of the internal buffers. The size of the nonce may be either 32 or 64 bytes. The TempKey.SourceFlag is always set to input after this command is run. This mode of the `Nonce` command does not run a SHA256 calculation or generate a random number.

Table 5-16. Fixed Nonce Input Parameters

Opcode	Mode	Param2	Data	Description
0x16	0x03 0x43 0x83	0x00	32 bytes	• 32 bytes written to TempKey • 32 bytes written to Message Digest Buffer • 32 bytes written to Alternate Key Buffer
	0x23 0x63	0x00	64 bytes	• 64 bytes written to TempKey • 64 bytes written to Message Digest Buffer

Table 5-17. Fixed Nonce Output Response

Name	Size	Description
Response	1 byte	0x00 if the command is completed successfully. Otherwise an error code is received.

Note:
1. TempKey.Source is set to 1, indicating nonce value was input.
2. TempKey.Valid is set to 1, indicating the value is valid for other use.

5.1.5 `Random` Command

The `Random` command generates a random number to be used by the system. Random numbers are generated via the internal NIST 800-90 A/B/C random number generator. The output of the command is always a 32-byte number placed on the bus. The number cannot be stored in any data slot or SRAM location.

Table 5-18. Input Parameters - Random

OpCode (1 Byte)	Mode (1 Byte)	Param2 (2 Bytes)	Data (0 Bytes)	Description
0x1B	0x00	0x00 00	—	`Random` command

Table 5-19. Output Response - Random

Name	Size	Description
RandOut	32 bytes	The output of the RNG

5.1.6 Read Command

The read command can be used to access any of the EEPROM zones of the ATECC608A-TNGTLS device. Data zone access is limited based on the access policies set for each of the slots. Encrypted reads are possible only on the Data zone slots if specific access policies are set.

5.1.6.1 Clear Text Read

Clear text reads are always possible from the Configuration and OTP zones of the ATECC608A-TNGTLS device. Specific slots within the Data zone may be readable as clear text based on the access policies that have been set. Either 4-byte or 32-byte reads are possible on any of these zones.

For the ATECC608A-TNGTLS device, Slots 8 and 10-15 contain data that can be read as clear text.

Table 5-20. Input Parameters - Clear Text Read

Opcode (1 Byte)	Mode (1 Byte)	Address (2 Byte)	Description
0x02	0x00	See Section 4.4 Address Encoding.	4-byte Configuration zone read
	0x80	See Section 4.4 Address Encoding.	32-byte Configuration zone read
	0x01	See Section 4.4 Address Encoding.	4-byte OTP zone read
	0x81	See Section 4.4 Address Encoding.	32-byte OTP zone read
	0x02	See Section 4.4 Address Encoding.	4-byte Data zone read
	0x82	See Section 4.4 Address Encoding.	32-byte Data zone read

Table 5-21. Output Response - Clear Text Read

Name	Size	Description
Data Contents	4 bytes	4 bytes transmitted [0:3]
	32 bytes	32 bytes transmitted [0:31]

5.1.6.2 Encrypted Read

Encrypted reads are only possible on Data zone slots that have the access policies set for an encrypted read. Data in the Configuration zone and OTP zone can never be encrypted. All encrypted reads must be 32 bytes in length. Prior to doing the encrypted read an encryption key must be generated. This key can be unique every time an encrypted read is done on a given slot. Note that in order to read all of the contents of a slot, multiple reads may be required. A unique session key will need to be generated for each encrypted read.

For the ATECC608A-TNGTLS device, no slots require an encrypted read.

Procedure for an Encrypted Read
The following steps are required for each encrypted read:

1. Run the `Nonce` command. It is recommended that this be done in Random mode, 32 bytes. Output the value to TempKey.
2. Run the `GenDig` command. The Slot # of the Encryption key must be included in the GenDig Input parameters, as well as the output of the `Nonce` command which is stored in TempKey.
   Note: The output of these two commands is the encryption key and is stored in TempKey.
3. Issue the Read command.

- The contents of the Data zone slot will be encrypted by XOR'ing the data with the generated value stored in TempKey. This value in TempKey is the session key that was previously generated.
- The output of the command will be the encrypted data.

Table 5-22. Input Parameters - Encrypted Read

Opcode (1 Byte)	Mode (1 Byte)	Address (2 Byte)	Description
0x02	0x82	See Section 4.4 Address Encoding	32-byte Data zone read

Table 5-23. Output Response - Encrypted Read

Name	Size	Description
Data Contents	32 bytes	32 bytes of encrypted data [0:31]

The host system must also calculate the Encryption/Decryption key based on the output of the `Nonce` command and the SHA256 calculation used in the `GenDig` command. This allows for the host system to decrypt the data being sent.

5.1.7 `SelfTest` Command

The `SelfTest` command performs a test of one or more of the cryptographic engines within the ATECC608A-TNGTLS chip. Some or all of the algorithms will be tested depending on the input mode parameter.

For the ATECC608A-TNGTLS device, the `SelfTest` command has been disabled from running automatically after a Power-up or Wake event. However, the command may be executed by the system if so desired. There is no requirement to run this test.

If any self test fails, whether called automatically on power-up, wake or via this command, the chip will enter a Failure state, in which chip operation is limited. The stored Failure state is always cleared upon a wake or power cycle. Note that the self-test failure (error code: 0x07) is not the same as a health-test failure (error code: 0x08).

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

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

Table 5-24. Input Parameters Self Test

Opcode (1 Byte)	Mode (1 Byte)[1]							Param2 (2 Bytes)
	b[7:6]	b[5]	b[4]	b[3]	b[2]	b[1]	b[0]	
	`2'b00`	SHA	AES	ECDH	ECDSA (Sign, Verify)	0	RNG, DRBG	0x00 00

Note:
1. Any combination of tests can be run at one time. Setting the corresponding mode bit to '`1`' indicates that the test will be run. If the bit is '`0`', then the test will not be run.

Table 5-25. Output Response Self Test

Name	Size	Description
Success	1 byte	0x00 - All Tests Passed Failure Map - one for each test that failed. Failure bits align with bits in Mode byte.

5.1.8 `SHA` Command

The `SHA` command computes a SHA-256 or HMAC/SHA digest for general purpose use by the host system. The SHA computation is performed in a special section of internal ATECC608A-TNGTLS memory (Context Buffer) that is not read nor written by any other commands. Any arbitrary command can be interspersed between the various phases of the `SHA` command without problems. This SHA context is invalidated on power-up and wake. In most cases, if an error occurs during the execution of the `SHA` command, the context is retained without change.

5.1.8.1 SHA - SHA256 Digest

The `SHA` command utilizes the SHA256 Hash algorithm for computing the hash of a message. The primary input to the command is the actual message. The message is submitted to the device in blocks of 1 to 64 bytes. The following procedure is used when no context switching is required:

1. Issue the `SHA` command in Start mode. No message is included.
2. Issue the `SHA` command in Update mode with 1 to 64 bytes of the message.
3. Repeat step 2 until the total number of bytes of the message have been submitted.
4. Issue the `SHA` command in End mode to complete the SHA256 calculation.

Table 5-26. Input Parameters SHA Standard Mode

Opcode (1 Byte)	Mode (1 Byte)	Param2 (2 Bytes)	Data (Varies by Mode)	Description
0x47	0x00	0x00 00	0 bytes	Start mode
	0x01	0x00 [ByteCount]	1 to 64 bytes	Update mode
	0x02	0x00 [ByteCount]	0 to 64 bytes	Finalize mode: Digest placed in Output Buffer and TempKey
	0x42	0x00 [ByteCount]		Finalize mode: Digest placed in Output Buffer and Message Digest Buffer
	0xC2	0x00 [ByteCount]		Finalize mode: Digest placed in Output Buffer only

Table 5-27. Output Response SHA256 Standard

Name	Mode	Size	Description
Response	0x00, 0x01	1 byte	0x00 if successful. otherwise an error code is received
	0x02, 0x42, 0xC2	1 byte 32 bytes	If error code SHA256 Digest

5.1.8.2 SHA - HMAC Digest

The `SHA` command may be used to calculate an HMAC Digest instead of an SHA256 Digest. The procedure is essentially the same but requires the use of an internal key and the use of HMAC_START instead of START.

The following procedure is used when no context switching is required:

1. Issue the `SHA` command in HMAC_Start mode with key location indicated. No message is included.
2. Issue the `SHA` command in Update mode with 1 to 64 bytes of the message.
3. Repeat step 3 until the total number of bytes of the message has been submitted.
4. Issue the `SHA` command in End mode to complete the HMAC Digest calculation.

Table 5-28. Input Parameters SHA HMAC Mode

Opcode (1 Byte)	Mode (1 Byte)	Param2 (2 Bytes)	Data (Varies by Mode)	Description
0x47	0x04	0x00 0[slot]	0 bytes	HMAC Start mode and include a key from a data slot.
	0x04	0xFF FF	0 bytes	HMAC Start mode and include a key from TempKey
	0x01	0x00 [ByteCount]	1 to 64 bytes	Update mode
	0x02	0x00 [ByteCount]	0 to 64 bytes	Finalize mode: Digest placed in Output Buffer and TempKey
	0x42	0x00 [ByteCount]		Finalize mode: Digest placed in Output Buffer and Message Digest Buffer
	0xC2	0x00 [ByteCount]		Finalize mode: Digest placed in Output Buffer only

Table 5-29. Output Response SHA256 Standard

Name	Mode	Size	Description
Response	0x04, 0x01	1 byte	0x00 if successful, otherwise an error code is received
	0x02, 0x42, 0xC2	1 byte 32 bytes	If Error Code SHA256 Digest upon success

5.1.8.3 SHA - Context Switching

Context switching allows for the generation of a digest to be interrupted to do other functions or to generate other digests. Context switching can be used only in the SHA256 Digest mode, so it can only occur after a SHA Start has been issued and prior to a SHA `Finalize` command. Context switching may happen multiple times during the course of a digest generation.

Context switching involves two phases:

1. Read_Context - Reads a variable length context from the ATECC608A-TNGTLS while leaving the context valid within the chip. The total length of the output data parameter is always from 40 to 99 bytes and can either be determined from the length field in the output packet or computed as 40 plus the Least Significant six bits of the first byte in the output.
2. Write_Context - Writes a SHA256 context from the host to the ATECC608A-TNGTLS to allow subsequent update operations to be completed. This context must have previously been read from the chip with the Read_Context mode. The ATECC608A-TNGTLS determines the size of the context from the first 4 bytes of the data parameter.

After the context has been read, the device may perform any other operations as required. Upon completion of the other operations, the context may be written back to the ATECC608A-TNGTLS and the SHA256 Digest generation process may continue until it has completed.

Table 5-30. Input Parameters SHA Context Switching

Opcode (1 Byte)	Mode (1 Byte)	Param2 (2 Bytes)	Data (Varies by Mode)	Description
0x47	0x06	0x00 00	0 bytes	Read the current context
	0x07	0x00 [ByteCount]	40 to 99 bytes	Restore the current context from the prior session

Table 5-31. Output Response SHA Context Switching

Name	Mode	Size	Description
Response	0x06	1 byte 40-99 bytes	If error code context value
	0x07	1 byte	0x00 if successful, otherwise an error code is received

5.1.9 `UpdateExtra` Command

The `UpdateExtra` command is used to update the UpdateExtra and UpdateExtraAdd bytes, bytes 84 and 85 respectively in the Configuration zone. These bytes can only be updated by this command. These bytes are one-time updatable bytes and can only be updated if the current value is 0x00. Trying to update this byte if the value is not 0x00 will result in an error.

For the ATECC608A-TNGTLS device, the UpdateExtraAdd byte (byte 85) has been configured to be an alternate I^{2}C address.

Table 5-32. Input Parameters - UpdateExtra

OpCode (1 Byte)	Mode (1 Byte)	Param2 (2 Bytes)	Data (0 Bytes)	Description
0x20	0x00	0x00 [Value]	—	Write the UpdateExtra byte (byte 84) with the value in the LSB of Param2.
—	0x01	0x00 [Value]	—	Write the UpdateExtraAdd byte (byte 85) with the value in the LSB of Param2.

Table 5-33. Output Response - UpdateExtra

Name	Size	Description
Success	1 byte	0x00 - The byte is written successfully. An error code is received if the byte is not written successfully.

5.1.10 Write Command

For the ATECC608A-TNGTLS, the Configuration zone and OTP zone have been locked and no updates to these zones are possible. Limited write capability exists on the Data zone based on access policies of each slot. Slots that can be written are described in the submodes of this command.

5.1.10.1 Data Zone - Clear Text Write

Standard Clear Text Writes
Clear text writes to Data zone slots are only possible for slots so configured provided they have not been SlotLocked. Note that any given slot may allow for more than one block of data to be written to it based on the size of the slot. 4 or 32-byte writes are permissible for any block within the slot. The last block of any slot will not be 32 bytes. This can still be written as a 32-byte write and the additional bytes need to be padded with zeros. For the ATECC608A-TNGTLS device, Slots 6, 8-9, 13 and 15 can be written as clear text.

Invalidated Public Key Writes
Since ECC public keys are not secret values, they may be directly written as ClearText by the write command. If the key requires validation prior to use then the key cannot be overwritten without first invalidating the key. The Invalidate mode of the `Verify` command must first be used to invalidate the PublicKey prior to trying to write this key with the write command.

For the ATECC608A-TNGTLS device, Slot 14 contains a validated public key.

The input parameters for a standard clear text write and the invalidated public key write are identical once the public key has been invalidated.

Table 5-34. Clear Text Write Input Parameters

Opcode (1 Byte)	Mode (1 Byte)	Address (2 Bytes)	Data (4 or 32 Bytes)	Description
0x12	0x02	See Section 4.4 Address Encoding	4 bytes	4-byte write
	0x82	See Section 4.4 Address Encoding	32 bytes	32-byte write

Table 5-35. Clear Text Write Output Response

Name	Size	Description
Response	1 byte	If successful, it will return a value of 0x00. If unsuccessful, then an error code will be returned.

5.1.10.2 Data Zone - Encrypted Write

Writes to the Data zone may be encrypted if the slots have been so configured. Only data written to the Data zone may be encrypted. For the ATECC608A-TNGTLS device, Slot 5 can be written as encrypted text.

All encrypted writes must be done as 32-byte blocks. If a partial block at the end of the zone needs to be encrypted 32 bytes of input, data must still be sent and used as part of the MAC calculation. The address of the write is an actual memory location address and is not a Data slot number.

Table 5-36. Input Parameters - Encrypted Write

Opcode (1 Byte)	Mode (1 Byte)	Address (2 Bytes)	Input Data (32 Bytes)	MAC (32 Bytes)	Description
0x12	0x82	See Section 4.4 Address Encoding	32 bytes of encrypted input data	32 bytes of MAC	32-byte encrypted write

Table 5-37. Output Response - Encrypted Write

Name	Size	Description
Response	1 byte	If successful, it will return a value of 0x00. If unsuccessful, then an error code will be returned.

Data Encryption
Data must be encrypted by the host system prior to writing the data to the slot. The encryption algorithm simply XOR's the clear text data with the value stored in the TempKey. TempKey must be a result of a `GenDig` command. The host system will need to calculate this value that will be used in parallel with what the ATECC608A-TNGTLS calculates. The `GenDig` command can be used one or more times when calculating the XOR value. The final value will be the actual XOR value used for the encryption. Once the data are encrypted and written, the ATECC608A-TNGTLS decrypts the value with the value stored in TempKey. The encrypted write must occur before any other commands that can affect the TempKey value or before a timeout occurs. In order to validate the encrypted write, a 32-byte MAC value must also be sent with the command.

Input MAC Generation
The required Input MAC is generated by a SHA256 Hash over 96 bytes. This is calculated by the host system and sent as part of the encrypted write command.

32 bytes	TempKey
1 byte	OpCode = 0x12
1 byte	Mode
2 bytes	Address (LSB, MSB)
1 byte	SN[8] = 0x01
2 bytes	SN[0:1]=0x01 0x23
25 bytes	Zeros
32 bytes	Plain Text Data

5.2 Asymmetric Cryptography Commands

The Asymmetric Cryptography command set is made up of those commands that are specifically used to generate or use ECC keys. Keys are typically stored in Data zone slots, but for some commands could also be in the SRAM array.

Table 5-38. Asymmetric Cryptography Commands

Command Name	Opcode	Description
`ECDH`	0x43	Generates an ECDH pre-master secret using the stored private key and input public key.
`GenKey`	0x40	Generates an ECC private key or optionally generates an ECC public key from the stored private key.
`SecureBoot`	0x80	Validates code signature or code digest on power-up.
`Sign`	0x41	Signs an internal or external message digest using an ECC private key with an ECDSA signature calculation.
`Verify`	0x45	Verifies an internal or external message digest using an ECC public key with an ECDSA verify calculation.

5.2.1 `ECDH` Command

The `ECDH` command is used to generate a shared secret between two devices. By passing an ECC public key from another device and combining with the ECC private key stored in a slot or with an ephemeral key stored in TempKey and doing the reverse on the other device, both devices will generate the same shared master secret. This can then be further combined with other common data in both sides to generate a shared session key between the devices. The `KDF` command is often used with TLS sessions to further diversify the shared secret.

5.2.1.1 ECDH - Stored Key

The `ECDH` command may use an internal data slot as its ECC private key source. The slot must have its access policies configured such that the slot is an ECC private key and that the `ECDH` command is allowed. Access policies may also specify whether or not the output will be stored, encrypted or allow the command itself to determine whether the output is stored or encrypted. The IO protection key is used when encryption is required. Encryption can only occur when data is output to the output buffer.

For the ATECC608A-TNGTLS, the `ECDH` command may be run using the ECC private keys stored in Slots 0 and 2-4.

Table 5-39. Input Parameters ECDH Stored Key

Opcode (1 Byte)	Mode (1 Byte)	KeyId (2 Bytes)	Data		Description
			Data 1 (32 Bytes)	Data 2 (32 Bytes)	
0x43	0x0C	0x00 0[Slot]	X component of public key	Y component of public key	• Results go to the output buffer • Output is in the clear(1)
	0x0E	0x00 0[Slot]	X component of public key	Y component of public key	• Results go to the output buffer • Output is encrypted
	0x08	0x00 0[Slot]	X component of public key	Y component of public key	• Results go to the TempKey • Output is available for other operations but is not directly accessible.

Note:
1. When the ChipOptions.ECDHPROT value is 1, then the output of the `ECDH` command will be encrypted in this mode. For the ATECC608A-TNGTLS the ECDHPROT field is set to 0 and encryption will be dependent upon the mode of the ECDH command.

Table 5-40. Output Response ECDH Stored Key

Name	Mode	Size	Description
Response	0x0C or 0x0E	1 byte	Error code if command fails
Response	0x0C	32 bytes	Shared Master Secret as clear text
Response OutNonce	0x0E	32 bytes 32 bytes	Shared Master Secret as encrypted text nonce used for encryption
Response	0x08	1 byte	0x00 if successful, otherwise an error code is returned

5.2.1.2 ECDH - TempKey Source

The `ECDH` command may use the value in TempKey as its starting value for an `ECDH` command. The ECC private key value in TempKey must be generated by the `GenKey` command. Once the TempKey has been used by the `ECDH` command, the TempKey.Valid flag is reset. If the output is back to the TempKey location, then this flag will be set again.

Table 5-41. Input Parameters ECDH TempKey

Opcode (1 Byte)	Mode (1 Byte)	KeyId (2 Bytes)	Data		Description
			Data 1 (32 Bytes)	Data 2 (32 Bytes)	
0x43	0x0D	0x00 00	X component of public key	Y component of public key	• Results go to the output buffer • Output is in the clear[1]
	0x0F	0x00 00	X component of public key	Y component of public key	• Results go to the output buffer • Output is encrypted
	0x09	0x00 00	X component of public key	Y component of public key	• Results go to the TempKey • Output is available for other operations but is not directly accessible.
	0x05	0x00 0[Slot]	X component of public key	Y component of public key	• Results go to the specified slot

Note:
1. When the ChipOptions.ECDHPROT value is 1, then the output of the `ECDH` command will be encrypted in this mode. For the ATECC608A-TNGTLS the ECDHPROT field is set to 0 and encryption will be dependent upon the mode of the ECDH command.

Table 5-42. Output Response ECDH TempKey

Name	Mode	Size	Description
Response	0x0D or 0x0F	1 byte	Error code if command fails
Response	0x0D	32 bytes	Shared Master Secret as clear text
Response OutNonce	0x0F	32 bytes 32 bytes	Shared Master Secret as encrypted text nonce used for encryption

...........continued

Name	Mode	Size	Description
Response	0x03 or 0x09	1 byte	0x00 if successful, otherwise an error code is returned

5.2.2 `GenKey` Command

The `GenKey` command is used to generate ECC private keys, ECC public keys from private keys or generate a public key digest. This command is only applicable for those slots designated to be ECC private or public keys. Running this command on a non-ECC slot will result in an error.

5.2.2.1 Private Key - Stored in Slot

The `GenKey` command can be used to generate an ECC P256 private key and store it in a data slot that has been so designated as holding an ECC private key. When this command is run, the corresponding ECC public key is also generated. If the slot has been locked, this command returns an error. On rare occasions an invalid ECC private key is generated and this too will cause an error.

For the ATECC608A-TNGTLS, the `GenKey` command can be used to generate private keys only in Slots 2, 3 and 4. Authorization is not required prior to updating these keys.

Table 5-43. Input Parameters - Private Key Stored in Slot

Opcode (1 Byte)	Mode (1 Byte)	KeyID (2 Bytes)	OtherData (0 Bytes)	Description
0x40	0x04	0x00 0[Slot]	—	• Private key stored in [Slot] • Public key generated and output on the bus
0x40	0x0C	0x00 0[Slot]	—	• Private key stored in [Slot] • Public key generated and output on the bus • Public key digest is generated and stored in TempKey

Table 5-44. Output Response - Private Key Stored in Slot

Name	Size	Response
Response	1 byte	• ECC Fault Code if command fails
	64 bytes	• Public keys X and Y coordinate the command’s success

5.2.2.2 Private Key - Stored in TempKey

The `GenKey` command can be used to generate an ephemeral ECC private key and place it in SRAM where there is no limit on writing to a memory location. This key cannot be read out but may be used by the `ECDH` command.

Table 5-45. Command Parameters

Opcode (1 Byte)	Mode (1 Byte)	KeyID (2 Bytes)	OtherData (3 Bytes)	Comment
0x40	0x04	0xFF FF	0x00 00 00	• Private key stored in TempKey • Public key generated and output on the bus

Table 5-46. Output Response GenKey Stored in TempKey

Name	Size	Response
Response	1 byte	If an ECC Fault code has been generated due to a command error
	64 bytes	Public keys X and Y coordinate the command’s success

5.2.2.3 Public Key Generation

If the slot has been so configured, the ECC public key can be regenerated from a stored ECC private key and output on the bus. In addition, if so desired, a public key digest can also be generated and stored in TempKey at the same time.

For the ATECC608A-TNGTLS, the public key can be generated from the stored private key in Slots 0-5. Optional digest generation is also allowed.

Table 5-47. Input Parameters - Public Key Generation

Opcode (1 Byte)	Mode (1 Byte)	KeyID (2 Bytes)	OtherData (0 Bytes)	Description
0x40	0x00	0x00 0[Slot]	—	Public key generated and output on the bus
0x40	0x08	0x00 0[Slot]	—	• Public key generated and output on the bus • Public key digest generated and stored in TempKey

Table 5-48. Output Response - Public Key Generation

Name	Size	Response
Response	1 byte	ECC Fault Code if command fails
	64 bytes	Public keys X and Y coordinate the command's success

Public Key Digest Generation
A SHA256 Hash is performed over 128 bytes to generate a digest from the public key.

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

5.2.2.4 Public Key Digest Generation

A digest of a public key may be generated from a stored ECC public key and stored in TempKey. The Slot# must point to a stored public key, therefore this command is limited to Slots 8 and higher. Prior to running this command, the TempKey must be a valid value. The public key will not be output on the bus. The public key could however be read out using the read command. Note that in this mode the 3 bytes in OtherData will be used to generate the digest and the mode and KeyID bytes will be ignored.

For the ATECC608A-TNGTLS, a digest can be created from Slots 11, 14 and 15 and optionally Slot 13 if it contains a public key.

Table 5-49. Input Parameters - Public Key Digest Generation

Opcode (1 Byte)	Mode (1 Byte)	KeyID (2 Bytes)	OtherData (3 Bytes)	Comment
0x40	0x10	0x00 0[Slot]	0x[any value]	Public key digest is created and stored in TempKey

Table 5-50. Output Response - Public Key Digest Generation

Name	Size	Response
Response	1 byte	• 0x00 if command completes successfully • ECC Fault Code if command fails

Public Key Digest Creation
A SHA256 Hash is performed over 128 bytes to generate a digest from the public key.

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

5.2.3 `SecureBoot` Command

The `SecureBoot` command provides support for secure boot of an external MCU or MPU. The general approach is that the boot code within the system will use the ATECC608A-TNGTLS to assist in validating the application code that is to be subsequently executed. The ATECC608A-TNGTLS device has been configured to operate in the SecureBoot, Stored Digest mode. Following a successful digest/signature validation, the digest will be stored the digest will be stored in Slot 7 and the public key required to verify the SecureBoot is stored in Slot 15. The device has not been configured to use the persistent latch and SecureBoot will not be tied up to power-up in any way.

In lieu of a return code, a MAC can optionally be generated from a nonce written to TempKey, the IO protection secret and various other data, dependent upon the mode of the command, to prevent tampering with the wire between the host and the ATECC608A-TNGTLS.

5.2.3.1 SecureBoot - FullCopy

The FullCopy mode of the `SecureBoot` command is similar to the Full mode but will also copy either the signature or verified digest to the target slot. The target slot is defined as part of the access policies for SecureBoot and is not part of the command. This mode of the `SecureBoot` command must be run before the command can be run in FullStore Digest or Signature mode. Optionally a MAC can be generated with a nonce from the host using the IO protection secret to prevent tampering with the wire between the host and the ATECC608A-TNGTLS.

For the ATECC608A-TNGTLS device, the digest will be copied to Slot 7 upon successful completion of this command.

Table 5-51. Input Parameters - SecureBoot FullCopy

Opcode (1 Byte)	Mode (1 Byte)	Param2 (2 Bytes)	Data (96 Bytes)	Description
0x80	0x07	0x00 00	• 32-byte digest of the entire code • 64-byte signature	• Code digest is unencrypted • Code digest and signature to be verified by the public key
	0x87	0x00 00	• 32-byte encrypted digest of the entire code • 64-byte signature	• Code digest is encrypted • Code digest and signature to be verified by the public key • Output MAC is generated

Table 5-52. Output Response - SecureBoot FullCopy

Name	Mode	Size	Response
Success	0x07	1 byte	• 0x00 - Successful • 0x01 - Computation completed but mismatch in result • Error code for other values
MAC	0x87	32 bytes	If successful
		1 byte	• 0x01 - Computation completed but mismatch in result • Error code for other values

SecureBoot FullCopy MAC Calculation
Prior to generating the MAC in this mode, the `Nonce` command must be run to insure a valid value is stored in TempKey. The MAC is then calculated in two steps.

Step 1: Generate SHA256 digest over the IO protection key and the nonce

32 bytes	Content of the IO protection key
32 bytes	First 32 bytes of nonce stored in TempKey

Step 2: SHA256 digest of the following

32 bytes	Digest generated in step 1
32 bytes	Plaintext Message. Output of step 1 XORed with the input encrypted code digest (first 32 bytes of the input buffer)
64 bytes	Signature as passed from the input
4 bytes	Input parameters (Opcode, Mode, Param2) (0x80, 0x86, 0x00 00)

5.2.3.2 SecureBoot - FullStore (Digest)

In the FullStore Digest mode of the `SecureBoot` command, the verified digest will be stored in a slot. This mode improves the IO transfer and overall computation times associated with the command. To use this mode, the FullCopy validation Command mode needs to be initially executed and the device will receive both the digest and the signature and store the digest in the slot specified in the SecureBoot access policies. Optionally a MAC can be generated with a nonce from the host using the IO protection secret to prevent tampering with the wire between the host and the ATECC608A-TNGTLS.

Table 5-53. Input Parameters - SecureBoot FullStore

Opcode (1 Byte)	Mode (1 Byte)	Param2 (2 Bytes)	Data (32 Bytes)	Description
0x80	0x06	0x00 00	• 32-byte digest of the entire code	• Code digest and signature to be verified by the public key
	0x86	0x00 00	• 32-byte encrypted digest of the entire code	• Code digest is encrypted • Code digest and signature to be verified by the public key • Output MAC is generated

Table 5-54. Output Response - SecureBoot FullStore

Name	Mode	Size	Response
Success	0x06	1 byte	0x00 - Successful 0x01 - Computation completed but mismatch in result. Error code for other values.
MAC	0x86	32 bytes	If successful
		1 byte	0x01 - Computation completed but mismatch in result. Error code for other values.

SecureBoot Stored Digest MAC Calculation
Prior to generating the MAC in this mode, the `Nonce` command must be run to insure a valid value is stored in TempKey. The MAC is then calculated in two steps.

Step 1: Generate SHA256 digest over the IO protection key and the nonce

32 bytes	Content of the IO protection key
32 bytes	First 32 bytes of nonce stored in TempKey

Step 2: SHA256 digest of the output of step 1 and the additional information shown below:

32 bytes	Digest generated in step 1
32 bytes	Plaintext Message. Output of step 1 XORed with the input encrypted code digest (first 32 bytes of the input buffer)
4 bytes	Input parameters (Opcode, Mode, Param2) (0x80, 0x86, 0x00 00)

5.2.4 `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. Multiple modes of this device are available depending on what tries to be signed, validated or invalidated.

5.2.4.1 Sign - Internal Message

The `Sign` command in the Internal Message mode is used to sign a message that was internally generated. The command calculates the internal message digest and then signs the digest using the ECDSA sign algorithm with the private ECC key specified in KeyID. Internally generated messages must always reside in TempKey. The value in TempKey must be generated using either the `GenDig` or the `GenKey` command. If TempKey is not valid an error will occur. Typical uses include:

- Signing an internally generated random key. This is typically generated by the `GenKey` command.
- The output of a `GenKey` or `GenDig` commands, provided the output is located in TempKey.

For the ATECC608A-TNGTLS device, only Slot 1 is capable of signing internally generated messages.

Table 5-55. Input Parameters - Sign Internal Message

Opcode (1 Byte)	Mode (1 Byte)	KeyId (2 Bytes)	Description
0x41	0x00 or 0x20	0x00 0[Slot]	Serial number is not included in the message digest calculation
	0x40 or 0x60	0x00 0[Slot]	Serial number is included in the message digest calculation

Table 5-56. Output Response - Sign Internal Message

Name	Size	Description
Response	1 byte	Error code if the command fails
	64 bytes	The signature composed of R and S values

Internal Message Generation
The internal message is generated based on a 55 Byte Field as shown below.

Byte Count	Serial Number Not Included	Serial Number Included
32 bytes	TempKey(1)	TempKey(1)
1 byte	Opcode	Opcode
1 byte	Mode	Mode
2 bytes	KeyID	KeyID
2 bytes	SlotConfig (of TempKeyFlags.KeyID)	SlotConfig (of TempKeyFlags.KeyID)
2 bytes	KeyConfig (of TempKeyFlags.KeyID)	KeyConfig (of TempKeyFlags.KeyID)
1 byte	TempKeyFlags(2)	TempKeyFlags(2)
2 bytes	Zeros	Zeros
1 byte	SN[8] = 0x01	SN[8] = 0x01
4 bytes	Zeros	SN[4:7]
2 bytes	SN[0:1] = 0x01 0x23	SN[0:1] = 0x01 0x23
2 bytes	Zeros	SN[2:3]
1 byte	SlotLocked:TempKeyFlags.KeyID	SlotLocked:TempKeyFlags.KeyID
1 byte	0x00	0x00
1 byte	0x00	0x00

Note:
1. TempKey must be generated by `GenKey` or `GenDig` commands prior to this calculation.
2. TempKeyFlags Consists of: (b[7]: NoMacFlag, b[6]: GenKeyData, b[5]: GenDigData, b[4]: SourceFlag,b[3:0] TempKeyFlags)

5.2.4.2 Sign - External Message

The `Sign` command can be used to sign the digest of an external message by an ECC private key. The message must be compiled and the digest of the message generated by the host system. The message can be loaded into either the TempKey or Message Digest Buffer via the `Nonce` command run in fixed mode and is always 32 bytes in length. The message always resides in the lower 32 bytes of these locations.

For the ATECC608A-TNGTLS device, Slots 0 and 2-4 are enabled to sign external messages.

Table 5-57. Input Parameters - Sign External

Opcode (1 Byte)	Mode (1 Byte)	KeyId (2 Bytes)	Description
0x41	0x80 or 0xC0	0x00 0[Slot]	External message digest stored in TempKey
	0xA0 or 0xD0	0x00 0[Slot]	External message digest stored in Message Digest Buffer

Table 5-58. Output Response - Sign External

Name	Size	Description
Response	1 byte	Error code if the command fails
	64 bytes	The signature composed of R and S values

5.2.5 `Verify` Command

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

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

5.2.5.1 Verify - External Public Key Mode

The `Verify` command may be used to verify a message generated externally to the ATECC608A-TNGTLS with a public key that is passed to the command. The output of the command will either be a code indicating success, failure or error or a 32-byte MAC. Prior to this command being run, the message should be written using the `Nonce` command in Fixed mode to either TempKey or the Message Digest Buffer. In this mode, the device merely accelerates the public key computation and returns a boolean result.

Procedure to Verify a Message with an External Public Key

1. Write the 32-byte digest of the message to either the TempKey or Message Digest Buffer using the `Nonce` command in Fixed mode.
2. Optional: System Nonce - Nonce generated by the system.
   - 2.1. If the external message digest is stored in TempKey, the nonce generated by the system must be stored in the lower 32 bytes of the Message Digest Buffer.
   - 2.2. If the external message is stored in the MessageDigestBuffer[31:0], then the System Nonce must be stored in the upper 32 bytes of the MessageDigest Buffer[63:32]. To do this, the external message and nonce value should be written as a 64-byte value.
3. Issue the `Verify` command. Include the Mode, KeyID, which specifies the P256 ECC Curve, the 64-byte signature and the 64-byte external public key.
4. The output will return:
   - 4.1. One byte success, fail or error code if MAC is not required.
   - 4.2. A 32-byte MAC if specified by the mode.

Table 5-59. Command Parameters

Opcode (1 Byte)	Mode (1 Byte)	Key ID (2 Bytes)	Data Field (128 Bytes)		Comment
			Signature (64 Bytes)	Public Key (64 Bytes)	
0x45	0x02	0x00 04	R value S value	X value Y value	Message stored in TempKey
	0x22	0x00 04	R value S value	X value Y value	Message stored in Message Digest Buffer
	0xA2	0x00 04	R value S value	X value Y value	• Message stored in TempKey • System Nonce stored in MDB[31:0] • Validation MAC is returned
	0x82	0x00 04	R value S value	X value Y value	• Message stored in Message Digest Buffer • System Nonce stored in MDB[63:32] • Validation MAC is returned

Table 5-60. Output Response - Verify External

Name	Mode	Size	Response
Response	0x02 or 0x22	1 byte	• 0x00 - If signature is verified • 0x01 - If signature does not match • Error code - If there is a failure due to some other reason
	0x82 or 0xA2	1 byte or 32 bytes	• Validation MAC - If signature is verified • 0x01 - If signature does not match • Error code - If there is a failure due to some other reason

Table 5-61. Validation MAC - Verify External

Size (Bytes)	Message in TempKey	Message in Message Digest Buffer
32	Contents of the IO protection key	Contents of the IO protection key
32	Message stored in TempKey	Message stored in the first 32 bytes of the Message Digest Buffer
32	System Nonce stored in the first 32 bytes of the Message Digest Buffer	System Nonce stored in the second 32 bytes of the Message Digest Buffer
32	R Data of the passed signature	R Data of the passed signature
32	S Data of the passed signature	S Data of the passed signature
1	Opcode	Opcode
1	Mode	Mode
2	Param2 [LSB,MSB]	Param2 [LSB,MSB]

5.2.5.2 Verify - Stored Public Key Mode

When using the `Verify` command in Stored mode, the public key to be used is stored in a data slot and does not need to be passed. Prior to this command being run, the message should be written to TempKey or the Message Digest Buffer using the `Nonce` command.

Procedure to Verify a Message with a Stored Key

1. If so required, validate the public key before using the `Verify` command.
2. If so required, authorize the public key before use.
3. Write the 32-byte digest of the message to either the TempKey or Message Digest Buffer using the `Nonce` command in Fixed mode.
4. Write the System Nonce to either the lower or upper 32 bytes of the Message Digest Buffer.
   - 4.1. If TempKey contains the message digest, then store the System Nonce in the lower 32 bytes of the Message Digest Buffer.
   - 4.2. If the message digest is stored in the lower 32 bytes of the Message Digest Buffer, then store the System Nonce in the upper 32 bytes of the Message Digest Buffer. The user needs to write both the message digest and the System Nonce at one time using the `Nonce` command.
5. Issue the `Verify` command. Include the Mode, KeyID, which specifies the public key slot, the 64-byte signature and the slot # of the internal public key.
6. The output will return:
   - 6.1. One byte success, fail or error code if MAC is not required.
   - 6.2. A 32-byte MAC, if specified by the mode or an error code, if the command fails.

Table 5-62. Command Parameters

Opcode (1 Byte)	Mode (1 Byte)	Public Key (2 Bytes)	Data Field (64 Bytes)	Comment
			Signature (64 Bytes)	
0x45	0x00	<0x00, 0[Slot]	R value S value	- Message stored in TempKey
	0x20	<0x00, 0[Slot]	R value S value	- Message stored in Message Digest Buffer
	0x80	<0x00, 0[Slot]	R value S value	- Message stored in TempKey Validation MAC is returned
	0xA0	<0x00, 0[Slot]	R value S value	- Message stored in Message Digest Buffer Validation MAC is returned

Table 5-63. Output Response - Verify Stored

Name	Mode	Size	Response
Response	0x00 or 0x20	1 byte	• 0x00 - If signature is verified • 0x01 - If signature does not match • Error code - If there is a failure due to some other reason
	0x80 or 0xA0	32 bytes 1 byte	• Validation MAC - If signature is verified • 0x01 - If signature does not match • Error code - If there is a failure due to some other reason

Table 5-64. Validation MAC - Verify Stored

The validation MAC input locations vary based on where the message is stored.

Size (Bytes)	Message in TempKey	Message in Message Digest Bufffer
32	Contents of the IO protection key	Contents of the IO protection key
32	Message from TempKey	First 32 bytes message from the Digest Buffer
32	System Nonce in the first 32 bytes of the Message Digest Buffer	System Nonce stored in the second 32 bytes of the Message Digest Buffer
32	R Data of the passed signature	R Data of the passed signature
32	S Data of the passed signature	S Data of the passed signature
1	Opcode	Opcode
1	Mode	Mode
2	Param2 [LSB,MSB]	Param2 [LSB,MSB]

5.2.5.3 Verify - Validate and Invalidate

The `Verify` command can be used to validate or invalidate a public key. Only those public keys whose access policies require validation need to go through this process. Prior to a public key being used to verify a signature, it must be validated. If a validated public key needs to be updated, then it needs to be invalidated prior to being written. Only internally stored public keys can be validated or invalidated. The status of a public key is stored in the most significant nibble of byte 0 of the public key slot.

For the ATECC608A-TNGTLS device, Slot 14 contains a validated public key.

Procedure for Validating or Invalidating a Public Key

1. Using GenKey, generate a digest of the public key to be validated or invalidated and store it in TempKey.
2. OtherData[18:0] bytes must be the same as the bytes that were used when calculating the original signature.
   – OtherData[17][0] = 0 if you are going to validate the key
   – OtherData[17][0] = 1 if you are going to invalidate the key
   – This bit must match the Mode[2] value of the `Verify Validate` or `Invalidate` command or an error will occur.

   Note: The message is created in the same manner as for the Internal mode of the `Sign` command, but it uses the OtherData[18:0] bytes.
3. Issue the `Verify Validate` or `Invalidate` command, including the signature R and S values and the OtherData bytes.
4. Upon successful validation or invalidation, a code of 0x00 will be returned and bits [7:4] of the LSB of the slot will be set.

Table 5-65. Input Parameters - Verify Validate/Invalidate

Opcode (1 Byte)	Mode (1 Byte)	Key ID (2 Bytes)	Data Field (83 Bytes)		Comment
			Signature (64 Bytes)	Other Data[(1)] (19 Bytes)	
0x45	0x03	0x00 0[Slot]	R value S value	OtherData[17][0] = 0	Validates public key
	0x07	0x00 0[Slot]	R Value S Value	OtherData[17][0] = 1	Invalidates public key

Note:

1. Other Data byte values must align with the data used to generate the original message.

Table 5-66. Output Response - Verify Validate/Invalidate

Name	Size	Description
Response	1 byte	• 0x00 - If signature is verified • 0x01 - If signature does not match • Error code - If there is a failure due to some other reason
ValidateNibble of Public Key	4 bits.	Slot[n][0] [7:4] will be updated of the public key • 0x5 - If the public key has been validated • 0xA - If the public key has been invalidated

Table 5-67. Generated Message

32 bytes	TempKey digest of the PublicKey (must be generated by GenKey)
1 byte	Sign Opcode
10 bytes	OtherData[0:9][(1)]
1 byte	SN[8] = 0x01
4 bytes	OtherData[10:13][(1)]
2 bytes	SN[0:1] = 0x01 0x23
5 bytes	OtherData[14:18][(1)]

Note:

1. These bytes should match the bytes used in the original message that generated the signature. The original message calculation can be found in Section 5.2.4.1 Internal Message Generation. The only exception is for bit 0 of byte 17, as described above.

5.3 Symmetric Cryptography Commands

5.3.1 AES Command

The AES Command can be used to encrypt and/or decrypt a 16-byte block of data utilizing an AES key. Note that the key is stored in a 16 Byte (128 bit) location with a given slot or within the first 16 bytes of TempKey. Multiple keys may be stored in a given slot and accessed in successive 16-byte boundaries, starting with 0-15 up to the size of the slot, but not exceeding four keys in any slot. For the ATECC608A-TNGTLS, an AES key may be stored in either Slot 5 or Slot 9. Slot 5 can accommodate up to two AES keys and Slot 9 can accommodate up to four AES keys.

In addition to AES encryption and decryption, the AES command may be used to generate a Galois Field Multiply (GFM) in support of other cryptographic operations.

5.3.1.1 AES-ECB Encrypt

In the AES-ECB Encrypt mode, 16 bytes of clear text are expected in the input stream and the device will output 16 bytes of encrypted text.

Table 5-68. AES-ECB Encrypt

Opcode (1 Byte)	Mode (1 Byte)	KeyID (2 Bytes)	Data (16 Bytes)	Comments
0x51	0x00 0x40 0x80 0xC0	0x00 0[Slot]	Any 16 bytes of ClearText data	Encrypt key in Position 0 Encrypt key in Position 1 Encrypt key in Position 2 Encrypt key in Position 3
	0x00	0xFF FF	Any 16 bytes of ClearText data	Encryption key located in TempKey

Table 5-69. AES Encrypt Output Response

Name	Size (Bytes)	Notes
Response	1	If the operation fails, the ouptut is a one byte error code.
	16	If the operation is successful, the device will output 16 bytes of encrypted text.

5.3.1.2 AES-ECB Decrypt

The AES-ECB Decrypt mode of the AES command is used to convert encrypted text back to clear text.

Table 5-70. AES-ECB Decrypt

Opcode (1 Byte)	Mode (1 Byte)	KeyID (2 Bytes)	Data (16 Bytes)	Comments
0x51	0x01 0x41 0x81 0xC1	0x00 0[Slot]	Any 16 bytes of AES Encrypted data	Decrypt key in Position 0 Decrypt key in Position 1 Decrypt key in Position 2 Encrypt key in Position 3
	0x01	0xFF FF	Any 16 bytes of ClearText data	Decryption key located in TempKey

Table 5-71. AES Decrypt Output Response

Name	Size (Bytes)	Notes
Response	1	If the operation fails, the ouptut is a one byte error code.
	16	If the operation is successful, the device will output 16 bytes of clear text data.

5.3.1.3 AES-GFM

The GFM operation is often used as part of various AES cryptographic operations. This function has been provided to aid in the creation of operations that are not directly supported by the ATECC608A-TNGTLS. The output of this operation can be used in the AES-GCM AEAD functionality. This mode does not involve secrets or anything stored on the chip. If this mode is selected, the remaining mode bits are ignored.

Table 5-72. AES Galois Field Multiply (GFM)

Opcode (1 Byte)	Mode (1 Byte)	KeyID (2 Bytes)	Data (32 Bytes)	Description
0x51	0x03	0x00 00	• First 16 bytes - H-Field • Second 16 bytes input data	

Table 5-73. AES GFM Output Response

Name	Size (Bytes)	Notes
Response	1	If the operation fails, the output is a one byte error code.
	16	If the operation is successful, the device will output 16 bytes result of the GFM calculation.

5.3.2 `CheckMac` Command

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

If a value in TempKey is used as an input to the `CheckMac`, then a `Nonce` and/or `GenDig` command must be run prior to the `CheckMac` command.

Table 5-74. Input Parameters CheckMac

Opcode (1 Byte)	Mode (1 Byte)[(2)]	KeyID (2 Bytes)	Data (77 Bytes)[(1)]	Description
0x28	0x00	0x00 0[Slot]	• 32-byte client challenge • 32-byte response generated by the client • 13 bytes other data	
	0x01	0x00 0[Slot]	• 32 bytes ignored but must be present • 32-byte client response • 13 bytes other data	Use if TempKey.Source was random
	0x05	0x00 0[Slot]		Use if TempKey.Source was fixed
	0x02	0x00 00	• 32-byte client challenge • 32 bytes client response • 13 bytes other data	Use if TempKey.Source was random
	0x06	0x00 00		Use if TempKey.Source was fixed

Note:
1. OtherData[0:12] values must match the values used in the original `MAC` command.
2. For modes other than 0x00, Mode[2] must match the TempKey.Source flag.

Table 5-75. Output Response CheckMac

Name	Size	Description
Response	1 byte	• 0x00 - If successful • 0x01 - If there is a mismatch • Error Code - If there is a failure

Table 5-76. SHA256 CheckMac Hash

# of Bytes	Mode 0x00	Mode 0x01 or 0x05	Mode 0x02 or 0x06
32	Key[KeyID]	Key[KeyID]	TempKey
32	Input Client Challenge	TempKey	Input Client Challenge
4	OtherData[0:3]	OtherData[0:3]	OtherData[0:3]
8	Zeros	Zeros	Zeros
3	OtherData[4:6]	OtherData[4:6]	OtherData[4:6]
1	SN[8] = 0x01	SN[8] = 0x01	SN[8] = 0x01
4	OtherData[7:10]	OtherData[7:10]	OtherData[7:10]
2	SN[0:1] = 0x01 0x23	SN[0:1] = 0x01 0x23	SN[0:1] = 0x01 0x23
2	OtherData[11:12]	OtherData[11:12]	OtherData[11:12]

5.3.3 `GenDig` Command

The `GenDig` command uses a SHA-256 Hash to combine a stored or input value with the contents of TempKey, which must be validated 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 the monotonic counters. The specific mode of the device determines which data is to be included in the GenDig calculation.

In some cases, it is required to run the `GenDig` prior to the execution of some commands. The command can be run multiple times to include more data in the digest prior to executing a given command. The resulting digest is retained in TempKey and can be used in one of four ways:

1. It can be included as part of the message used by the `MAC`, `Sign` or `CheckMac` commands. Because the MAC response output incorporates both the data used in the GenDig calculation and the secret key from the `MAC` command, it serves to authenticate the data stored in the Data and/or OTP zones.
2. A subsequent read or write command can use the digest to provide authentication and/or confidentiality for the data, in which case it is known as a data protection digest.
3. The command can be used for secure personalization by using a value from the transport keyarray. 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.

5.3.3.1 GenDig - Config

Data from the Configuration zone can be included in the GenDig calculation. Data are always included in 32-byte blocks and only one block may be included on any given GenDig calculation. The `Nonce` command must be run prior to the first `GenDig` command to load a value into TempKey. Subsequent `GenDig` commands will use the value stored in TempKey from the previous GenDig operation.

Table 5-77. Input Parameters GenDig Config

Opcode (1 Byte)	Mode (1 Byte)	KeyID[(1)] (2 Bytes)	Data (0 Bytes)	Description
0x15	0x00	0x00 00	—	Use Configuration Block 0
		0x00 01	—	Use Configuration Block 1
		0x00 02	—	Use Configuration Block
		0x00 03	—	Use Configuration Block 3

Note:
1. KeyId specifies the Configuration zone block to be used in the TempKey Calculation.

Table 5-78. Output Response - GenDig Config

Name	Size	Description
Response	1 byte	0x00 - If successful. Error code - If the command fails.

Note: Flag Bits
1. TempKey.Valid flag will be set to 1 if successful, otherwise it is 0.
2. TempKey.GenDigData will be set to 0.

Table 5-79. TempKey Calculation - GenDig Config

32 bytes	Configuration Zone Block
1 byte	Opcode = 0x15
1 byte	Mode = 0x00
2 bytes	KeyID[0:1] = 0x0[block] 0x00
1 byte	SN[8] = 0x01
2 bytes	SN[0:1] = 0x01 0x23
25 bytes	All Zeros
32 bytes	TempKey.Value

5.3.3.2 GenDig - OTP

Data from the OTP zone can be included in the GenDig calculation. Data are always included in 32-byte blocks and only one block may be included on any given GenDig calculation. The `Nonce` command must be run prior to the first `GenDig` command to load a value into TempKey. Subsequent `GenDig` commands will use the value from the previous GenDig operation.

Table 5-80. Input Parameters - GenDig OTP

Opcode (1 Byte)	Mode (1 Byte)	KeyID[(1)] (2 Bytes)	Data (0 Bytes)	Description
0x15	0x01	0x00 00	—	Use OTP Block 0 as the KeyID
		0x00 01	—	Use OTP Block 1 as the KeyID

Note:
1. KeyId specifies the OTP zone block to be used in the TempKey calculation.

Table 5-81. Output Response - GenDig OTP

Name	Size	Description
Response	1 byte	0x00 - If successful. Error code - If the command fails.

Note: Flag Bits
1. TempKey.Valid flag will be set to 1 if successful, otherwise it is 0.
2. TempKey.GenDigData will be set to 0.

Table 5-82. TempKey Calculation - GenDig OTP

32 bytes	OTP Zone Block
1 byte	Opcode = 0x15
1 byte	Mode = 0x00
2 bytes	KeyID[0:1] = 0x0[block] 0x00
1 byte	SN[8] = 0x01
2 bytes	SN[0:1] = 0x01 0x23
25 bytes	All Zeros
32 bytes	TempKey.Value

5.3.3.3 GenDig - Data

Data from the Data zone slots can be included in the GenDig calculation. Data are always included in 32-byte blocks and only the lowest block of a slot is included in the GenDig calculation. The `Nonce` command must be run prior to the first `GenDig` command to load a value into TempKey. If the slot requires a random nonce, then the data must be generated by the `Nonce`command versus passing it into the device. Subsequent `GenDig` commands will use the value from the previous GenDig operation.

If a slot is set for NoMAC, then it is not allowed to use the output of this GenDig in a `MAC` command. If multiple GenDigs are run, if any slot has NoMAC set, then the output in TempKey cannot be used in executing a `MAC` command.

Table 5-83. Input Parameters - GenDig Data

Opcode (1 Byte)	Mode (1 Byte)	KeyID(1) (2 Bytes)	Data (0 or 4 Bytes)	Description
0x15	0x02	0x00 0[Slot]	—	Use data from slot # in the calculation.
		0x00 0[Slot]	4 bytes	Use data from slot # in the calculation and other data when using a NoMAC key slot

Note:
1. KeyId specifies the Data zone slot to be used in the TempKey calculation. Only the lowest 32 bytes will be used.

Table 5-84. Output Response - GenDig Data

Name	Size	Description
Response	1 byte	0x00 - If successful. Error code - If the command fails.

Note: Flag Bits
1. TempKey.Valid flag will be set to 1 if successful, otherwise it is 0.
2. TempKey.GenDigData will be set to 1 indicating a DataZone slot was used in the calculation.
3. TempKey.KeyID will set to the slot specified in the command.
4. TempKey.NoMacFlag will be set to 0 if the `MAC` command is allowed and 1 if it is not.

Table 5-85. TempKey Calculation - GenDig Data

Size	Parameters - MAC	Parameters - NoMAC
32 bytes	DATA.slot[KeyID]	DATA.slot[KeyID]
4 byte	Opcode, Mode, KeyID = 0x15, 0x02, 0x0[Slot] 0x00	OtherData[0:3]
1 byte	SN[8] = 0x01	SN[8] = 0x01

...........continued		
Size	**Parameters - MAC**	**Parameters - NoMAC**
2 bytes	SN[0:1] = 0x01 0x23	SN[0:1] = 0x01 0x23
25 bytes	All Zeros	All Zeros
32 bytes	TempKey.Value	TempKey.Value

5.3.3.4 GenDig - Shared Nonce

In Shared Nonce mode 32 bytes of data are input to this command. This mode is used when a nonce value must be shared between two devices. The `Nonce` command must be run prior to the first `GenDig` command to load a value into TempKey. Subsequent `GenDig` commands will use the value from the previous GenDig operation.

Table 5-86. Input Parameters GenDig Shared Nonce

Opcode (1 Byte)	Mode (1 Byte)	KeyID(1) (2 Bytes)	Data (32 Bytes)	Description
0x15	0x03	0x00 0[Slot]	Input data	
		0x80 0[Slot]	Input data	

Note:
1. KeyId specifies the DataZone slot to be used in the TempKey calculation. Only the lowest 32 bytes will be used.

Table 5-87. Output Response - GenDig Data

Name	Size	Description
Response	1 byte	0x00 - If successful. Error code - If the command fails.

Note: Flag Bits
1. TempKey.Valid flag will be set to 1 if successful, otherwise it is 0.
2. TempKey.GenDigData will be set to 1 indicating a DataZone slot was used in the calculation.
3. TempKey.KeyID will set to the slot specified in the command.

Table 5-88. TempKey Calculation - GenDig Shared Nonce

Size	Parameters - KeyID MSB 0x00	Parameters - KeyID MSB 0x80
32 bytes	Input Data	TempKey.value
1 byte	Opcode = 0x15	Opcode = 0x15
1 byte	Mode = 0x03	Mode = 0x03
1 byte	LSB of KeyID = 0x0[Slot]	LSB of KeyID = 0x0[Slot]
1 byte	0x00	0x00
1 byte	SN[8] = 0x01	SN[8] = 0x01
2 bytes	SN[0:1] = 0x01 0x23	SN[0:1] = 0x01 0x23
25 bytes	All Zeros	All Zeros
32 bytes	TempKey.Value	Input Data

5.3.3.5 GenDig - Counter

In the Counter mode of the `GenDig` command, the binary value of the counter is included in the TempKey calculation. The `Nonce` command must be run prior to the first `GenDig` command to load a value into TempKey. Subsequent `GenDig` commands will use the value from the previous GenDig operation.

Table 5-89. Input Parameters - GenDig Counter

Opcode (1 Byte)	Mode (1 Byte)	KeyID (2 Bytes)	Data (0 Bytes)	Description
0x15	0x04	0x00 00	—	Include Counter[0] value
		0x00 01	—	Include Counter[1] value

Note:
1. KeyId specifies the monotonic counter value to be used in the TempKey calculation.

Table 5-90. Output Response - GenDig Counter

Name	Size	Description
Response	1 byte	0x00 - If successful. Error code - If the command fails.

Note: Flag Bits
1. TempKey.Valid flag will be set to 1 if successful, otherwise it is 0.
2. TempKey.GenDigData will be set to 0.

Table 5-91. TempKey Calculation - GenDig Counter

32 bytes	All Zeros
1 byte	Opcode = 0x15
1 byte	Mode = 0x04
2 bytes	KeyID[0:1] = 0x0[Counter#] 0x00
1 byte	SN[8] = 0x01
2 bytes	SN[0:1] = 0x01 0x23
1 byte	Zero
4 bytes	Counter[KEYID] - Binary value as reported by the `Counter` command
20 bytes	All Zeros
32 bytes	TempKey.Value

5.3.3.6 GenDig - Key Config

In the Key Config mode of the `GenDig` command, the slot configuration and key configuration of the key specified by KeyID are included in the GenDig TempKey calculation. The `Nonce` command must be run prior to the first `GenDig` command to load a value into TempKey. Subsequent `GenDig` commands will use the value from the previous GenDig operation.

Table 5-92. Input Parameters - GenDig Key Config

Opcode (1 Byte)	Mode (1 Byte)	KeyID(1) (2 Bytes)	Data (0 or 4 Bytes)	Description
0x15	0x05	0x00 0[Slot]	—	Includes slot configuration information

Note:
1. KeyId specifies the slot # that will have its configuration information included in the TempKey calculation. The actual slot value is not included.

Table 5-93. Output Response - GenDig Key Config

Name	Size	Description
Response	1 byte	0x00 - If successful. Error code - If the command fails.

Note: Flag Bits

1. TempKey.Valid flag will be set to 1 if successful, otherwise it is 0.
2. TempKey.GenDigData will be set to 0.

Table 5-94. TempKey Calculation - GenDig Key Config

32 bytes	TempKey.value
1 byte	Opcode = 0x15
1 byte	Mode = 0x05
2 bytes	KeyID[0:1] = 0x0[Slot] 0x00
1 byte	SN[8] = 0x01
2 bytes	SN[0:1] = 0x01 0x23
1 byte	0x00
2 bytes	SlotConfig[KeyID]
2 bytes	KeyConfig[KeyID]
1 byte	SlotLocked[KeyID]
20 bytes	All Zeros

5.3.4 `KDF` Command

The `KDF` command implements several Key Derivation Functions (KDF). Generally, this function combines a source key with an input string and creates a symmetric resultant key/digest/array. The input key may be located in either the TempKey, the Alternate Key Buffer, or an EEPROM Slot and can be either 32 or 64 bytes in length.

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

5.3.4.1 KDF - PRF

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

Table 5-95. Input Parameters - KDF PRF

Opcode (1 Byte)	Mode (1 Byte)	KeyId[(1)] (2 Bytes)	Details (4 Bytes)	Data (0 to 127) Bytes	Description	
					Input Source	Output Result
0x56	0x00 0x04 0x08 0x0C 0x10 0x14	0x00 00 0x00 00 0x0[SO] 00 0x00 00 0x00 00 0x00 00	See Table 5-96	Specified by DataLen in Table 5-96	TempKey (32 or 64 Bytes)	TempKey Temp Key - Upper Block EEPROM Slot Alternate Key Buffer OutBuf - Clear Text OutBuf - Encrypted
	0x01 0x05 0x09 0x0D 0x11 0x15	0x00 00 0x00 00 0x0[SO] 00 0x00 00 0x00 00 0x00 00	See Table 5-96	Specified by DataLen in Table 5-96	Temp Key Upper Block - 32 Bytes	TempKey Temp Key - Upper Block EEPROM Slot Alternate Key Buffer OutBuf - Clear Text OutBuf - Encrypted
	0x02 0x06 0x0A 0x0E 0x12 0x16	0x00 0[SI] 0x00 0[SI] 0x0[SO] 0[SI] 0x00 0[SI] 0x00 0[SI] 0x00 0[SI]	See Table 5-96	Specified by DataLen in Table 5-96	EEPROM Slot	TempKey Temp Key - Upper Block EEPROM Slot Alternate Key Buffer Outtuf - Clear Text OutBuf - Encrypted
	0x03 0x07 0x0B 0x0F 0x13 0x17	0x00 00 0x00 00 0x0[SO] 00 0x00 00 0x00 00 0x00 00	See Table 5-96	Specified by DataLen in Table 5-96	Alternate Key Buffer	TempKey Temp Key - Upper Block EEPROM Slot Alternate Key Buffer OutBuf - Clear Text OutBuf - Encrypted

Note:
1. [SO] = Output Slot #, [SI] = Input Slot #.

Table 5-96. Detail Parameter Encoding for PRF

Bits	Name	Description
31-24	DataLen	Length in bytes of the input parameter (Label | Seed in TLS lingo).
23-11	Zero	All bits must be 0.
10-9	Aead	`00` = no special AEAD processing. `01` = TargetLen must be 1 to generate 64 bytes. • First 32 bytes go to the target which must not be output • Second 32 remaining bytes go to the output buffer, never encrypted `10` and `11` forbidden, will return a parse error.
8	TargetLen	The number of 32-byte blocks to be placed in the target location. 0 = 1 block (32 bytes) 1 = 2 blocks (64 bytes)
7-2	Zero	All bits must be 0.

...........continued

Bits	Name	Description
1-0	KeyLen	The length of the source key in 16 byte blocks. 0 = 1 block (16 bytes) 1 = 2 blocks (32 bytes) 2 = 3 blocks (48 bytes) 3 = 4 blocks (64 bytes)

Table 5-97. Output Response - KDF - PRF

Name	Modes	Size	Description
OutData	0x00 to 0x0F	1	Success or failure code for all modes where output is sent to internal location.
	0x10 to 0x17	32 or 64	Output data for modes where output is placed in the output buffer.
OutNonce	0x00 to 0x13	0	If the output is not encrypted.
	0x14 to 0x17	32	If the output is encrypted, a random nonce will be generated and output.

5.3.4.2 KDF - AES

The AES mode of the `KDF` command calculates AES-ECB on a single block of input data. The upper 16 bytes of the output are padded to 0 so the result is always 32 bytes.

Table 5-98. Input Parameters - KDF AES

Opcode (1 Byte)	Mode (1 Byte)	KeyId[1] (2 Bytes)	Details (4 Bytes)	Data (16 Bytes)	Description	
					Input Key Source (16 Bytes)	Output Result
0x56	0x20 0x24 0x28 0x2C 0x30 0x34	0x00 00 0x00 00 0x0[SO] 00 0x00 00 0x00 00 0x00 00	See Table 5-99	Data to be encrypted	TempKey (32 or 64 bytes)	TempKey Temp Key - Upper Block EEPROM Slot Alternate Key Buffer OutBuf - Clear Text OutBuf - Encrypted
	0x21 0x25 0x29 0x2D 0x31 0x35	0x00 00 0x00 00 0x0[SO] 00 0x00 00 0x00 00 0x00 00	See Table 5-99	Data to be encrypted	Temp Key Upper Block - 32 bytes	TempKey Temp Key - Upper Block EEPROM Slot Alternate Key Buffer OutBuf - Clear Text OutBuf - Encrypted
	0x22 0x26 0x2A 0x2E 0x32 0x36	0x00 0[SI] 0x00 0[SI] 0x0[SO] 0[SI] 0x00 0[SI] 0x00 0[SI] 0x00 0[SI]	See Table 5-99	Data to be encrypted	EEPROM Slot	TempKey Temp Key - Upper Block EEPROM Slot Alternate Key Buffer Outtuf - Clear Text OutBuf - Encrypted
	0x23 0x27 0x2B 0x2F 0x33 0x37	0x00 00 0x00 00 0x0[SO] 00 0x00 00 0x00 00 0x00 00	See Table 5-99	Data to be encrypted	Alternate Key Buffer	TempKey Temp Key - Upper Block EEPROM Slot Alternate Key Buffer OutBuf - Clear Text OutBuf - Encrypted

Note:
1. [SO] = Output Slot #, [SI] = Input Slot #.

Table 5-99. Detail Parameter Encoding for AES

Bits	Name	Description
31-2	Zero	All bits must be 0.
1-0	KeyLoc	The AES key is located at Src[KeyLoc*16] within the source key material. • For 64-byte locations, the key may be located in one of the 4 blocks. • For 32-byte locations, the key may be located in block 0 or 1.

Table 5-100. Output Response KDF - AES

Name	Modes	Size	Description
OutData	0x20 to 0x2F	1	Success or failure code for all modes where output is sent to internal location.
	0x30 to 0x37	32	Output data for modes where output is placed in the output buffer.
OutNonce	0x20 to 0x33	0	If the output is not encrypted.
	0x34 to 0x37	32	If the output is encrypted, a random nonce will be generated and output.

5.3.4.3 KDF - HKDF

The HKDF function within the `KDF` command is intended to support the necessary key derivation operations as specified in TLS 1.3 and other protocols. It always computes a single iteration of HMACSHA256 using the key and message as specified in the mode and detail parameters. Multiple iterations of this command can be used to implement the HKDF extract and expand functions per the HKDF specification. 64-byte results can be created by separately writing to both the upper and lower halves of the TempKey register.

For the ATECC608A-TNGTLS, the special IV functionality has been disabled in the Configuration zone and cannot be used.

Table 5-101. Input Parameters - KDF HKDF

Opcode (1 Byte)	Mode (1 Byte)	KeyId[2] (2 Bytes)	Details (4 Bytes)	Data (0 to 127) Bytes	Description	
					Input Source	Output Result
0x56	0x00[1] 0x44 0x48 0x4C 0x50 0x54	0x00 00 0x00 00 0x0[SO] 00 0x00 00 0x00 00 0x00 00	See Table 5-102	Specified by DataLen in Table 5-102	TempKey (32 or 64 bytes)	TempKey Temp Key - Upper Block EEPROM Slot Alternate Key Buffer OutBuf - Clear Text OutBuf - Encrypted
	0x41 0x45 0x49 0x4D 0x51 0x55	0x00 00 0x00 00 0x0[SO] 00 0x00 00 0x00 00 0x00 00	See Table 5-102	Specified by DataLen in Table 5-102	Temp Key Upper Block - 32 bytes	TempKey Temp Key - Upper Block EEPROM Slot Alternate Key Buffer OutBuf - Clear Text OutBuf - Encrypted
	0x42 0x46 0x4A 0x4E 0x52 0x56	0x00 0[SI] 0x00 0[SI] 0x0[SO] 0[SI] 0x00 0[SI] 0x00 0[SI] 0x00 0[SI]	See Table 5-102	Specified by DataLen in Table 5-102	EEPROM Slot	TempKey Temp Key - Upper Block EEPROM Slot Alternate Key Buffer Outtuf - Clear Text OutBuf - Encrypted
	0x43 0x47 0x4B 0x4F 0x53 0x57	0x00 00 0x00 00 0x0[SO] 00 0x00 00 0x00 00 0x00 00	See Table 5-102	Specified by DataLen in Table 5-102	Alternate Key Buffer	TempKey Temp Key - Upper Block EEPROM Slot Alternate Key Buffer OutBuf - Clear Text OutBuf - Encrypted

⚠ CAUTION

1. If the output and input are pointing to the lower 32 bytes of TempKey, then the MsgLoc portion of the details parameter must not be set to TempKey (MsgLoc = `01`). Results are unpredictable if this occurs. This combination is forbidden.
2. [SO] = Output Slot #, [SI] = Input Slot #.

Table 5-102. Detail Parameter Encoding for HKDF

Bits	Name	Description
31-25	DataLen	Length in bytes of the HKDF message. If this value is zero, then the message will be 32 bytes of 0x00.
24-12	Zero	All bits must be 0.

...........continued

Bits	Name	Description
11-8	msgKey	The key slot for the message if in EEPROM.
7-3	Zero	All bits must be 0.
2	ZeroKey	If 1, the key is 32 bytes of 0x00.
1-0	MsgLoc	The location of the message. `00` = EEPROM slot `01` = TempKey `10` = Input Parameter `11` = Disabled for the ATECC608A-TNGTLS.

Table 5-103. Output Response - KDF HKDF

Name	Modes	Size	Description
OutData	0x40 to 0x4F	1	Success or failure code for all modes where output is sent to internal location.
	0x50 to 0x57	32 or 64	Output data for modes where output is placed in the output buffer.
OutNonce	0x40 to 0x53	0	If the output is not encrypted.
	0x54 to 0x57	32	If the output is encrypted, a random nonce will be generated and output.

5.3.4.4 KDF Output Encryption

For all of the KDF algorithms, output buffer encryption has been implemented in a manner similar to the `ECDH` command. While encryption is not required, it may be useful in providing additional system security. For the ATECC608A-TNGTLS, the output buffer encryption has been enabled. The IO protection key used for encryption has been stored in Slot 6. Refer to each of the modes of operation to determine the specific mode values that require encryption.

The following procedure is used in generating an encrypted output:

1. The first 32 bytes of the IO protection key slot (Config.ChipOptions[15:12]) are copied to a SHA256 buffer.
2. The internal RNG generates a 32-byte random number and appends the first 16 bytes of that nonce to the SHA256 buffer
3. The SHA256 buffer is hashed and the digest is XORed with the first 32 bytes of the clear text KDF result. If there are only 16 bytes in the result, then the output buffer will contain only those 16 bytes and the second 16 bytes of the SHA digest will be ignored.
4. If there are more than 32 bytes in the output, then a new digest is created via the SHA256 hash of the IO protection key (32 bytes), followed by the second 16 bytes of the random nonce from step #2. The resulting digest is XORed with the next 32 bytes of the result.
5. The output buffer consists of the encrypted KDF result followed by the 32 byte nonce. All 32 bytes of the nonce are output even if only the first 16 have been used.

5.3.5 `MAC` Command

The Message Authentication Code (`MAC`) command is used to generate a SHA256 digest of a message, which consists 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.

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

1. Run the `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 the `GenDig` command one or more times to combine 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 1 (and step 2 if desired) with an EEPROM key to generate an output response (i.e., digest).

Alternatively, data in any slot (which does not have to 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.

5.3.5.1 Non-Diversified MAC

The MAC is always calculated over a total of 88 bytes and always creates a 32-byte SHA256 digest. A non-diversified MAC does not include the serial number of the device and will therefore be the same across all devices if the input parameters are the same.

Table 5-104. Input Parameters - Non-Diversified MAC

Opcode (1 Byte)	Mode (1 Byte)	KeyID (2 Bytes)	Data(2) (0-32 Bytes)	Mode Descriptions
0x08	0x00	0x00 0[Slot]	32 bytes	• First 32 bytes loaded from data slot • Second 32 bytes are taken from the input challenge
	0x01 or 0x05(1)	0x00 0[Slot]	0 bytes	• First 32 bytes loaded from data slot • Second 32 bytes are taken from TempKey
	0x02 or 0x06(1)	0x00 00	32 bytes	• First 32 bytes loaded with TempKey • Second 32 bytes are taken from the input challenge

Note:

(1) Mode[2] must match the TempKey.SourceFlag.
(2) When present, the Data parameter corresponds to the input challenge.

Table 5-105. Output Response - Non-Diversified MAC

Name	Size	Description
Response	1 byte	If the command fails
	32 bytes	SHA-256 digest

Table 5-106. Non-Diversified MAC Calculation

# of Bytes	Mode 0x00	Mode 0x01 or 0x05	Mode 0x02 or 0x06
32	Data Slot	Data Slot	TempKey
32	Input Challenge	TempKey	Input Challenge
1	Opcode (0x08)	Opcode (0x08)	Opcode (0x08)
1	Mode	Mode	Mode
2	KeyID	KeyID	KeyID
11	Zeros	Zeros	Zeros
1	SN[8] 0x01	SN[8] 0x01	SN[8] 0x01
4	Zeros	Zeros	Zeros
2	SN[0:1] 0x01 0x23	SN[0:1] 0x01 0x23	SN[0:1] 0x01 0x23
2	Zeros	Zeros	Zeros

5.3.5.2 Diversified MAC

A diversified MAC includes the serial number of the device. The serial number will be unique for each device and therefore will always generate a unique SHA256 digest. The MAC is always calculated over a total of 88 bytes and always creates a 32 byte SHA256 digest.

Table 5-107. Input Parameters - Diversified MAC

Opcode (1 Byte)	Mode (1 Byte)	KeyID (2 Bytes)	Data[2] (0-32 Bytes)	Mode Descriptions
0x08	0x40	0x00 0[Slot]	32 bytes	• First 32 bytes loaded from data slot • Second 32 bytes are taken from the input challenge
	0x41 or 0x45[1]	0x00 0[Slot]	0 bytes	• First 32 bytes loaded from data slot • Second 32 bytes are taken from TempKey
	0x42 or 0x46[1]	0x00 00	32 bytes	• First 32 bytes loaded with TempKey • Second 32 bytes are taken from the input challenge

Note:

(1) Mode[2] must match the TempKey.SourceFlag.
(2) When present, the Data parameter corresponds to the input challenge.

Table 5-108. Output Response - Diversified MAC

Name	Size	Description
Response	1 byte	If the command fails
	32 bytes	SHA-256 digest

Table 5-109. Diversified MAC Calculation

# of Bytes	Mode 0x40	Mode 0x41 or 0x45	Mode 0x42 or 0x46
32	Data Slot	Data Slot	TempKey
32	Input Challenge	TempKey	Input Challenge
1	Opcode (0x08)	Opcode (0x08)	Opcode (0x08)
1	Mode	Mode	Mode
2	KeyID	KeyID	KeyID
11	Zeros	Zeros	Zeros
1	SN[8]0x01	SN[8] 0x01	SN[8] 0x01
4	SN[4:7]	SN[4:7]	SN[4:7]
2	SN[0:1] 0x01 0x23	SN[0:1] 0x01 0x23	SN[0:1] 0x01 0x23
2	SN[2:3]	SN[2:3]	SN[2:3]

6. Application Information

The ATECC608A-TNGTLS is a member of the Microchip's Trust&GO CryptoAuthentication™ family of products. The Trust&GO products are easy to use, simple to implement and allow even low volume customers to implement security into their end system while leveraging Microchip's expertise and infrastructure in secure provisioning.

The ATECC608A-TNGTLS device has been developed to take the guesswork out of adding security to an IoT-connected product. The product has been pre-configured to readily connect to the IoT Cloud through TLS connections and to provide support for multiple other security use cases including Secure Boot, Disposable and Accessory Authentication and User Data and I/P Protection.

In addition to the actual security device, Microchip has developed a series of tools that seamlessly integrate with our hardware devices to provide an easy path to develop your entire security solution. When the developers use Microchip's software security tools, they eliminate the complexity of setting up their own infrastructure and provide a rapid path to initial prototypes and production.

6.1 Use Cases

The ATECC608A-TNGTLS has been defined to specifically address the IoT market. The device has been set to directly support the Google IoT Cloud™. Support for Amazon AWS® or other TLS servers can be done by adding certificates to Slot 8. Other use cases can also be supported by this device. A brief description of the primary use case for this device is provided below.

Secure TLS Connection
The ATECC608A-TNGTLS allows the creation of secure TLS connections using a variety of protocols. The device is capable of establishing secure connections to the Google Cloud, to AWS and other cloud providers. Through the various modes of the Key Derivation Function (KDF), appropriate keys can be generated to support TLS1.2, TLS1.3 and earlier secure connection internet protocols.

6.2 Development Tools

The ATECC608A-TNGTLS is supported with multiple hardware and software tools and backend services that provide a path to rapidly develop applications. Initial development can start by using a family of easy-to-use use case tools. These tools provide a graphical way to implement your use case and end with the C code necessary to implement your application.

If your application differs from what the predefined use case tools can provide, then through use of the CryptoAuthLib or the Python® version of CryptoAuthLib and CryptoAuthTools, an application can be developed. CryptoAuthLib is also the backbone of the code that is output from the use case tools.

Full verification of your application can be implemented via hardware tools along with samples of the ATECC608A-TNGTLS device. Since the access policies of the device have already been set, the focus revolves just around developing the system level code.

Once the application is complete, the ATECC608A-TNGTLS devices can be ordered through Microchip Direct.

6.2.1 Use Case Tools

To simplify the implementation process Microchip has developed web-based use case tools that will allow developers to go from concept to production via a guided flow. The tools allow you to develop and construct the transaction diagrams and code necessary to implement a particular application within the constraints of the configuration and defined access policies of the ATECC608A-TNGTLS.

More information on these tools can be found under Microchip's CryptoAuthentication Products found under the Security IC's section of the webpage.

6.2.2 Hardware Tools

There are multiple hardware tools that can help in developing with the ATECC608A-TNGTLS. Check the Microchip website for the availability of additional tools that are not mentioned here. Specific tools are also mentioned with the specific use case examples.

DM320109 - CryptoAuthentication Starter Kit
The DM320109 consists of an ATSAMD21-XPRO development board pre-programmed with firmware that can work with CryptoAuthentication Devices. The kit comes with the AT88CKSCKTSOIC-XPRO socket board but you will need to obtain the UDFN version of the board to work with the sample devices which are currently provided only in the UDFN package. Specific samples of the ATECC608A-TNGTLS will need to be obtained separately. This board readily connects with the use case tools to develop applications.

AT88CKSCKTUDFN(SOIC)-XPRO
The AT88CKSCKTUDFN-XPRO and AT88CKSCKTSOIC-XPRO are generic CryptoAuthentication socket kits that can be used with any Microcontroller development board with an XPRO interface. Specific samples of the ATECC608A-TNGTLS must be acquired to be used with these kits.

6.2.3 CryptoAuthLib

CryptoAuthLib is a software library that supports Microchip's family of CryptoAuthentication devices. Microchip recommends working with this library when developing with the ATECC608A-TNGTLS. The library implements the API calls necessary to execute the commands detailed in this data sheet.

The library has been implemented to readily work with many of Microchip's microcontrollers but can easily be extended through a Hardware Abstraction Layer (HAL) to other microcontrollers including those made by other vendors.

For more information on these tools, check the information on:

- CryptoAuthLib - Web Link
- CryptoAuthLib - GitHub

API Calls
Each of the commands in the data sheet have one or more API calls that are associated with it. Typically, there is a base API call of the command where all input parameters can be specified. The parameter shown in the commands and subsections can be used with this command. There are also mode variants of each of the API calls. The table below shows examples of commands and base API calls. For the most accurate API information, refer to the GitHub information.

Table 6-1. Example Commands to CryptoAuthLib API Calls

Device Command	API Call	Comments
Info	atcab_info()	
Write	atcab_write()	
Read	atcab_read()	
SHA	atcab_sha()	
Sign	atcab_sign()	
Random	atcab_random()	
Verify	atcab_verify()	

6.3 TrustFLEX vs. Trust&GO

Trust&GO products have been defined for customers with low volumes that can use a secure off-the-shelf solution. The product's simple onboarding procedure leverages Microchip's secure manufacturing solution and infrastructure. By using this flow, customers do not have to create their own secure manufacturing environment.

However, at times customers may want to have more control over their security environment while still enjoying the simplicity of the Trust&GO product use. The TrustFLEX product was created for these customers. TrustFLEX products are still capable of implementing the use cases of the Trust&GO product but provide additional flexibility for some of the security keys and certificates.

- Same locked configuration as that of the Trust&GO product.
- Same data slot definition as that of the Trust&GO.
- Ability to provision the customer's public key into the device for Secure Boot implementation.
- Ability to do Symmetric Key Authentication. Customers can securely provide their desired Symmetric Keys as part of the secure provisioning process.
- Ability to customize certificate elements and link the certificate chain to the customer's desired PKI.
- Option for either I^{2}C or SWI interface devices.

For more information on TrustFLEX products and other provisioning options, see the Microchip CryptoAuthentication webpages.

7. I^{2}C Interface

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

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

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

7.1 I/O Conditions

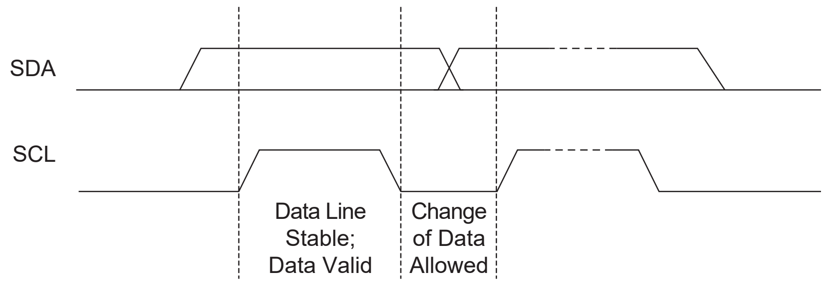
The device responds to the following I/O conditions:

7.2 Device is Awake

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

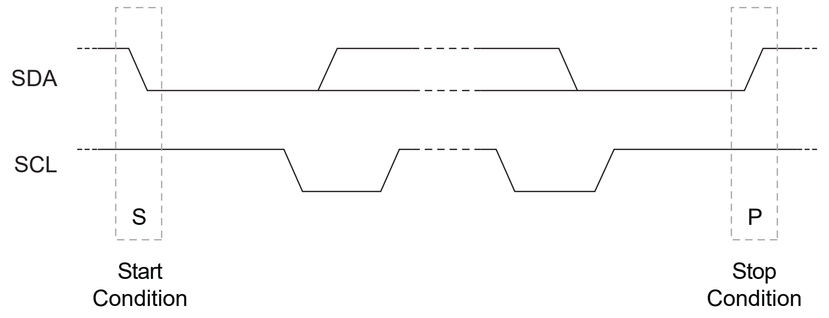
- **DATA Zero**: If SDA is low and stable while SCL goes from low to high to low, then a zero bit is being transferred on the bus. SDA can change while SCL is low.
- **DATA One**: If SDA is high and stable while SCL goes from low to high to low, then a one bit is being transferred on the bus. SDA can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

Figure 7-1. Data Bit Transfer on I^{2}C Interface



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

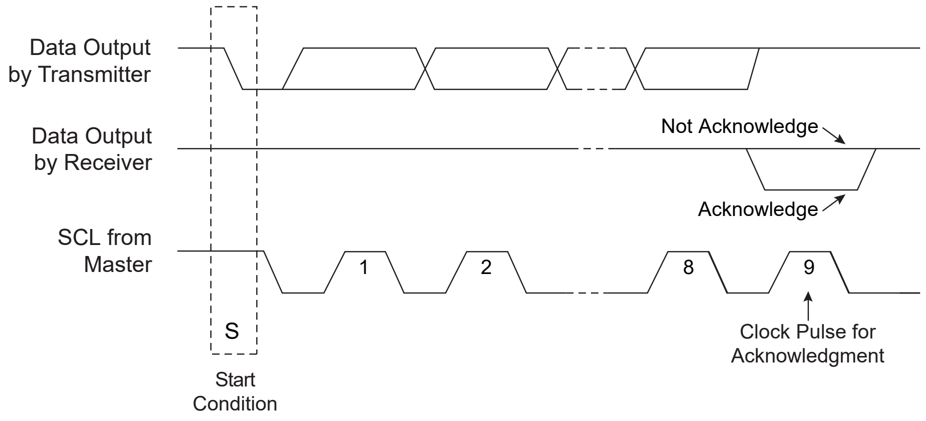
Figure 7-2. Start and Stop Conditions on I^{2}C Interface



- **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 (NACK)**: Alternatively, on the ninth clock cycle after every address or data byte is transferred, the receiver can leave the SDA pin high to indicate that there was a problem with the reception of the byte or that this byte completes the group transfer.

Figure 7-3. NACK and ACK Conditions on I^{2}C Interface



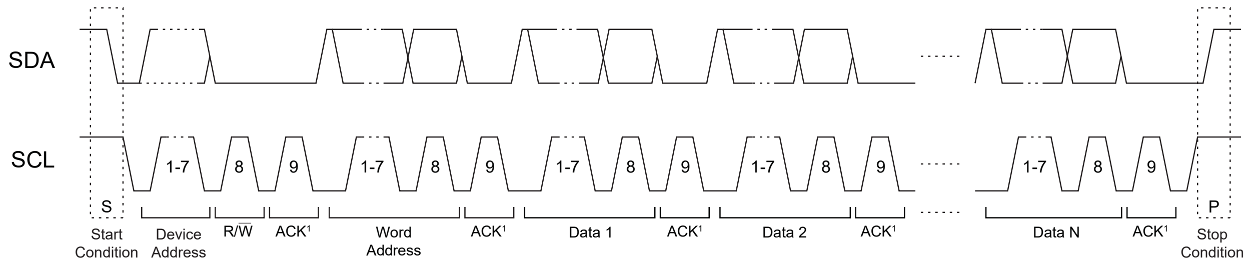
Multiple ATECC608A-TNGTLS devices can easily share the same I^{2}C interface signals if the I2C_Address byte in the Configuration zone is programmed differently for each device on the bus. Because all seven of the bits of the device address are programmable, ATECC608A-TNGTLS can also share the I^{2}C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

7.3 I^{2}C Transmission to ATECC608A-TNGTLS

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

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

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



SDA is driven low by ATECC608A-TNGTLS ACK periods.

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

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

Name	I^2C Name	Description
Device Address	Device Address	This byte selects a particular device on the I^2C interface. ATECC608A-TNGTLS is selected if bits 1 through 7 of this byte match bits 1 through 7 of the I2C_Address byte in the Configuration zone. Bit 0 of this byte is the standard I^2C 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 Section 7.3.1 Word Address Values 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 4.1 I/O Transactions.

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

The system must send a Stop condition after the last command byte to ensure that ATECC608A-TNGTLS will start the computation of the command. Failure to send a Stop condition may eventually result in a loss of synchronization; see Section 7.3.2 I2C Synchronization for recovery procedures.

7.3.1 Word Address Values

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

Table 7-2. Word Address Values

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

7.3.2 I^2C Synchronization

It is possible for the system to lose synchronization with the I/O port on the ATECC608A-TNGTLS, perhaps due to a system reset, I/O noise, or other conditions. Under this circumstance, the ATECC608A-TNGTLS 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 can be followed:

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

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

If the device does ACK the device address, the system should reset the internal address counter to force the ATECC608A-TNGTLS 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 must send a complete Wake token and wait t_{WHI} after the rising edge. The system may then send another read sequence, and if synchronization is complete, 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 must wait the longest t_{EXEC} (max) and then send the read sequence, which will be acknowledged by the device.

7.4 Sleep Sequence

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

7.5 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 Idle mode prior to completion of the watchdog interval. When the device receives the Wake token, it will then restart the Watchdog Timer and execution can be continued.

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

7.6 I^{2}C Transmission from the ATECC608A-TNGTLS

When the ATECC608A-TNGTLS is awake and not busy, the bus master can retrieve the current output buffer contents from the device using an I^{2}C Read. If valid command results are available, the size of the group returned is determined by the particular command which has been run. Otherwise, the size of the group (and the first byte returned) will always be four: count, status/error, and 2-byte CRC. The bus timing is shown in Figure 8-2.

Table 7-3. I^{2}C Transmission from the ATECC608A-TNGTLS

Name	I^{2}C Name	Direction	Description
Device Address	Device Address	To slave	This byte selects a particular device on the I2C interface and ATECC608A-TNGTLS will be selected if bits 1 through 7 of this byte match bits 1 through 7 of the I2C_Address byte in the Configuration zone. Bit 0 of this byte is the standard I^{2}C 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. See Section 4.1 I/O Transactions.

The status, error, or command outputs can be read repeatedly by the master. Each time a read command is sent to the ATECC608A-TNGTLS along the I^{2}C interface, the device transmits the next sequential byte in the output buffer. See the following section for details on how the device handles the address counter.

If the ATECC608A-TNGTLS is busy, idle, or asleep, it will NACK the device address on a read sequence. If a partial command has been sent to the device and a read sequence `[Start + DeviceAddress(R/W == R)]` is sent to the device, then the ATECC608A-TNGTLS will NACK the device address to indicate that no data is available to be read.

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	5.0 mA
Voltage on any pin -0.5V to (VCC + 0.5V)	-0.5V to (VCC + 0.5V)
ESD Ratings:	
Human Body Model(HBM) ESD	>4kV
Charge Device Model(CDM) ESD	>1kV

Note: 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 ATECC608A-TNGTLS is fabricated with Microchip's high reliability CMOS EEPROM manufacturing technology.

Table 8-1. EEPROM Reliability

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

8.3 AC Parameters: All I/O Interfaces

Figure 8-1. AC Timing Diagram: All Interfaces

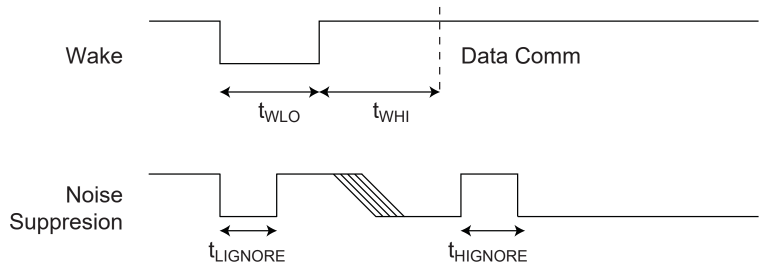


Table 8-2. AC Parameters: All I/O Interfaces

Parameter	Sym.	Direction	Min.	Typ.	Max.	Units	Conditions
Power-Up Delay[2]	t_{PU}	To Crypto Authentication	100	—	—	μs	Minimum time between V_{CC} > V_{CC} min prior to start of t_{WLO}.
Wake Low Duration	t_{WLO}	To Crypto Authentication	60	—	—	μs	
Wake High Delay to Data Comm	t_{WHI}	To Crypto Authentication	1500	—	—	μs	SDA should be stable high for this entire duration unless polling is implemented. SelfTest is not enabled at power-up.
Wake High Delay when SelfTest is Enabled	t_{WHIST}	To Crypto Authentication	20	—	—	ms	SDA should be stable high for this entire duration unless polling is implemented.
High-Side Glitch Filter at Active	$t_{HIGNORE\_A}$	To Crypto Authentication	45[1]	—	—	ns	Pulses shorter than this in width will be ignored by the device, regardless of its state when active.
Low-Side Glitch Filter at Active	$t_{LIGNORE\_A}$	To Crypto Authentication	45[1]	—	—	ns	Pulses shorter than this in width will be ignored by the device, regardless of its state when active.
Low-Side Glitch Filter at Sleep	$t_{LIGNORE\_S}$	To Crypto Authentication	15[1]	—	—	μs	Pulses shorter than this in width will be ignored by the device when in Sleep mode.
Watchdog Timeout	$t_{WATCHDOG}$	To Crypto Authentication	0.7	1.3	1.7	s	Time from wake until device is forced into Sleep mode if Config.ChipMode[2] is 0.

Note:
1. These parameters are characterized, but not production tested.
2. The power-up delay will be significantly longer if power-on self test is enabled in the Configuration zone.

8.3.1 AC Parameters: I²C Interface

Figure 8-2. I²C Synchronous Data Timing

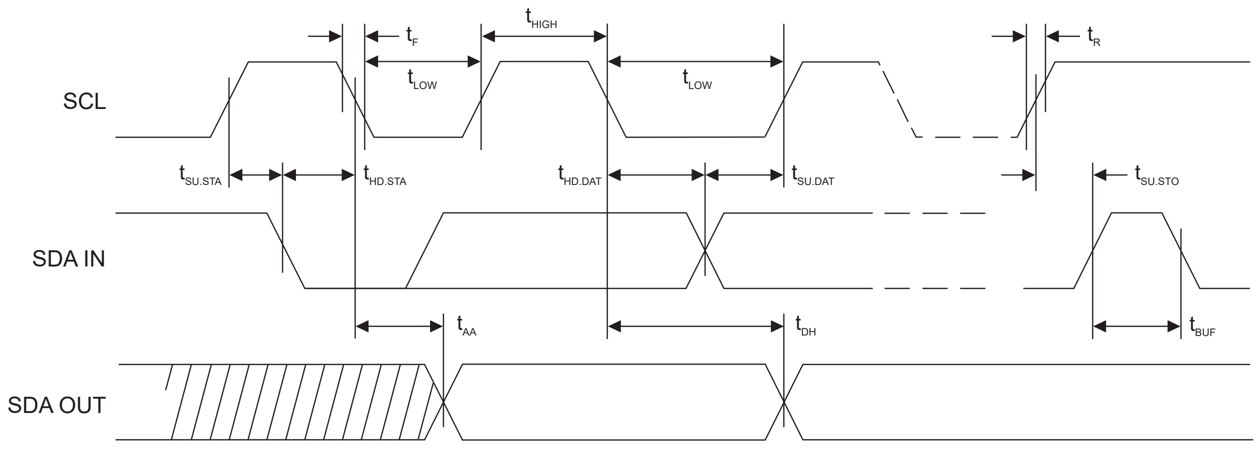


Table 8-3. AC Characteristics of I2C Interface

Unless otherwise specified, applicable over recommended operating range from T_A = -40°C to +85°C, V_{CC} = +2.0V to +5.5V, C_L = 1 TTL Gate and 100 pF

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

Note:
1. Values are based on characterization and are not tested.
2. AC measurement conditions:
   - R_L (connects between SDA and V_{CC}): 1.2 kΩ (for V_{CC} = +2.0V to +5.0V)
   - Input pulse voltages: $0.3V_{CC}$ to $0.7V_{CC}$
   - Input rise and fall times: ≤ 50 ns
   - 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	Sym.	Min.	Typ.	Max.	Units	Conditions
Ambient Operating Temperature	T_A	-40	—	+85	°C	
Power Supply Voltage	V_{CC}	2.0	—	5.5	V	
Active Power Supply Current	I_{CC}	—	2	3	mA	Waiting for I/O during I/O transfers or execution of non-ECC commands. Independent of Clock Divider value.
		—	—	14	mA	During ECC command execution. Clock divider = 0x0
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} ≤ 3.6V, V_{SDA} and V_{SCL} < 0.4V or > V_{CC} – 0.4, T_A ≤ +55°C
		—	—	2	μA	When device is in Sleep mode. Over full V_{CC} and temperature range.

...........continued

Parameter	Sym.	Min.	Typ.	Max.	Units	Conditions
Output Low Voltage	V_{OL}	—	—	0.4	V	When device is in Active mode, V_{CC} = 2.5 to 5.5V
Output Low Current	I_{OL}	—	—	4	mA	When device is in Active mode, V_{CC} = 2.5 to 5.5V, V_{OL} = 0.4V
Theta JA	θ_{JA}	—	166	—	°C/W	SOIC (SSH)
		—	173	—	°C/W	UDFN (MAH)
		—	146	—	°C/W	RBH

8.5 V_{IH} and V_{IL} Specifications

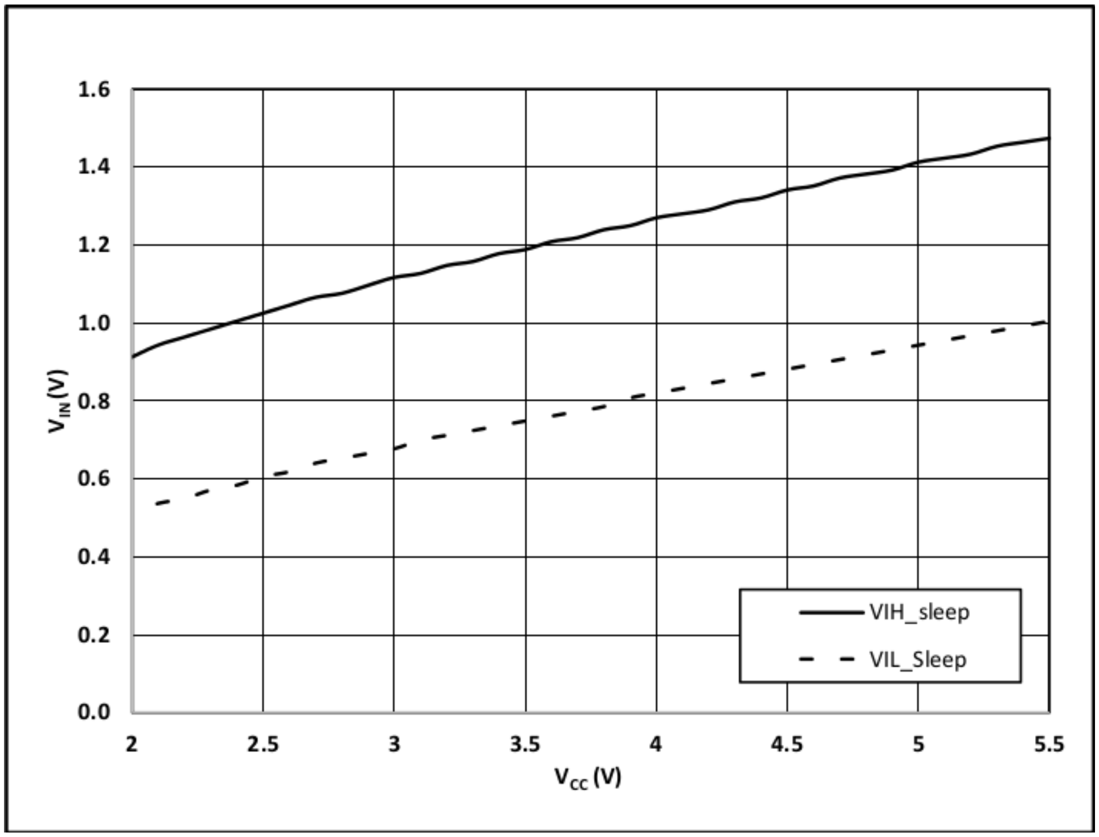
The input levels of the device will vary dependent on the mode and voltage of the device. The input voltage thresholds when in Sleep or Idle mode are dependent on the V_{CC} level as shown in Figure 8-3. When in Sleep or Idle mode the TTLenable bit has no effect.

The active input levels of the ATECC608A-TNGTLS are fixed and do not vary with the V_{CC} level. The input levels transmitted to the device must comply with the table below.

Table 8-5. V_{IL}, V_{IH} on All I/O Interfaces (TTLenable = 0)

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

Figure 8-3. V_{IH} and V_{IL} in Sleep and Idle Mode



9. Package Drawings

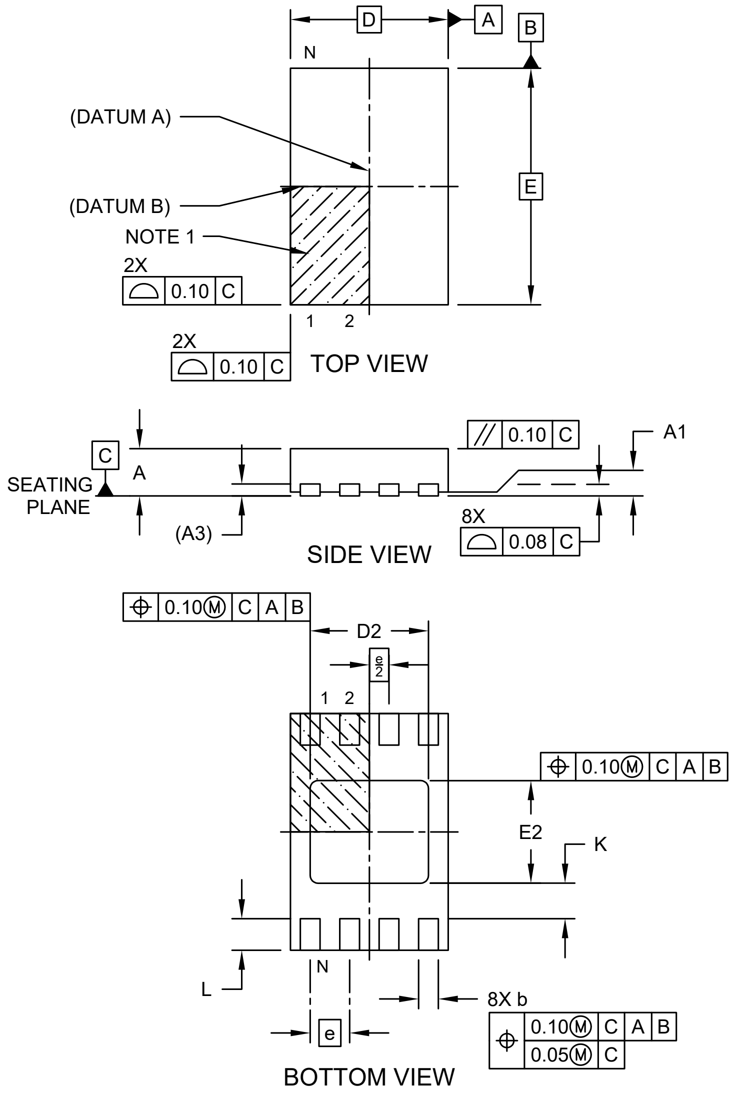
9.1 Package Marking Information

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

9.2 8-pad UDFN

8-Lead Ultra Thin Plastic Dual Flat, No Lead Package (Q4B) - 2x3 mm Body [UDFN]
Atmel Legacy YNZ Package

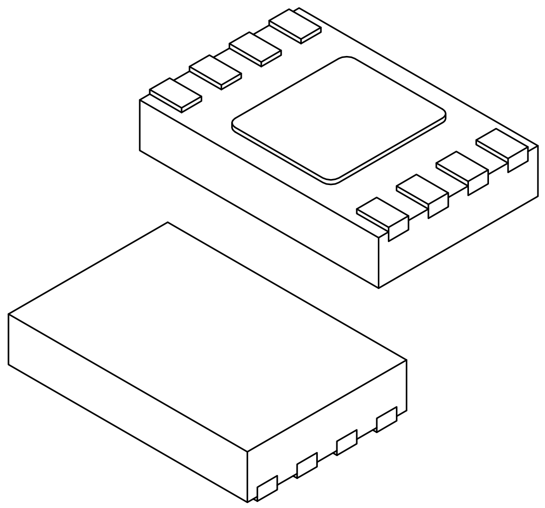
Note:	For the most current package drawings, please see the Microchip Packaging Specification located at http://www.microchip.com/packaging



Microchip Technology Drawing C04-21355-Q4B Rev A Sheet 1 of 2

8-Lead Ultra Thin Plastic Dual Flat, No Lead Package (Q4B) - 2x3 mm Body [UDFN] Atmel Legacy YNZ Package

Note: For the most current package drawings, please see the Microchip Packaging Specification located at http://www.microchip.com/packaging



	Units	MILLIMETERS		
	Dimension Limits	MIN	NOM	MAX
Number of Terminals	N		8	
Pitch	e		0.50 BSC	
Overall Height	A	0.50	0.55	0.60
Standoff	A1	0.00	0.02	0.05
Terminal Thickness	A3		0.152 REF	
Overall Length	D		2.00 BSC	
Exposed Pad Length	D2	1.40	1.50	1.60
Overall Width	E		3.00 BSC	
Exposed Pad Width	E2	1.20	1.30	1.40
Terminal Width	b	0.18	0.25	0.30
Terminal Length	L	0.35	0.40	0.45
Terminal-to-Exposed-Pad	K	0.20	-	-

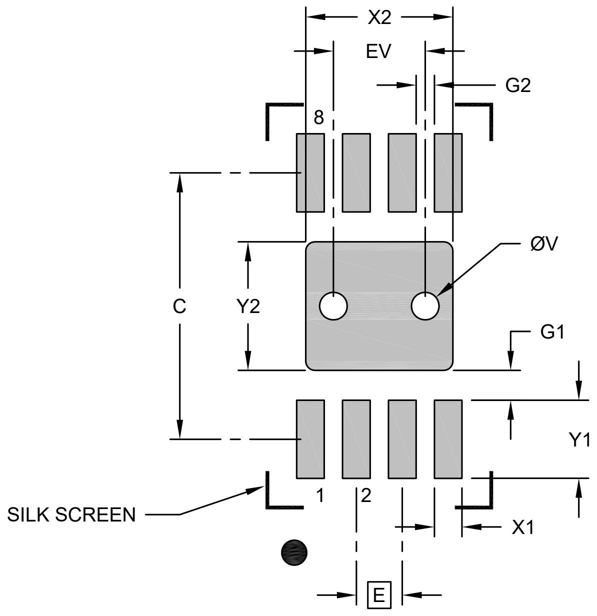
Notes:

1. Pin 1 visual index feature may vary, but must be located within the hatched area.
2. Package is saw singulated
3. Dimensioning and tolerancing per ASME Y14.5M
   BSC: Basic Dimension. Theoretically exact value shown without tolerances.
   REF: Reference Dimension, usually without tolerance, for information purposes only.

Microchip Technology Drawing C04-21355-Q4B Rev A Sheet 2 of 2

**8-Lead Ultra Thin Plastic Dual Flat, No Lead Package (Q4B) - 2x3 mm Body [UDFN]
Atmel Legacy YNZ Package**

Note: For the most current package drawings, please see the Microchip Packaging Specification located at http://www.microchip.com/packaging



RECOMMENDED LAND PATTERN

	Units	MILLIMETERS		
	Dimension Limits	MIN	NOM	MAX
Contact Pitch	E		0.50 BSC	
Optional Center Pad Width	X2			1.60
Optional Center Pad Length	Y2			1.40
Contact Pad Spacing	C		2.90	
Contact Pad Width (X8)	X1			0.30
Contact Pad Length (X8)	Y1			0.85
Contact Pad to Center Pad (X8)	G1	0.20		
Contact Pad to Contact Pad (X6)	G2	0.33		
Thermal Via Diameter	V		0.30	
Thermal Via Pitch	EV		1.00	

Notes:

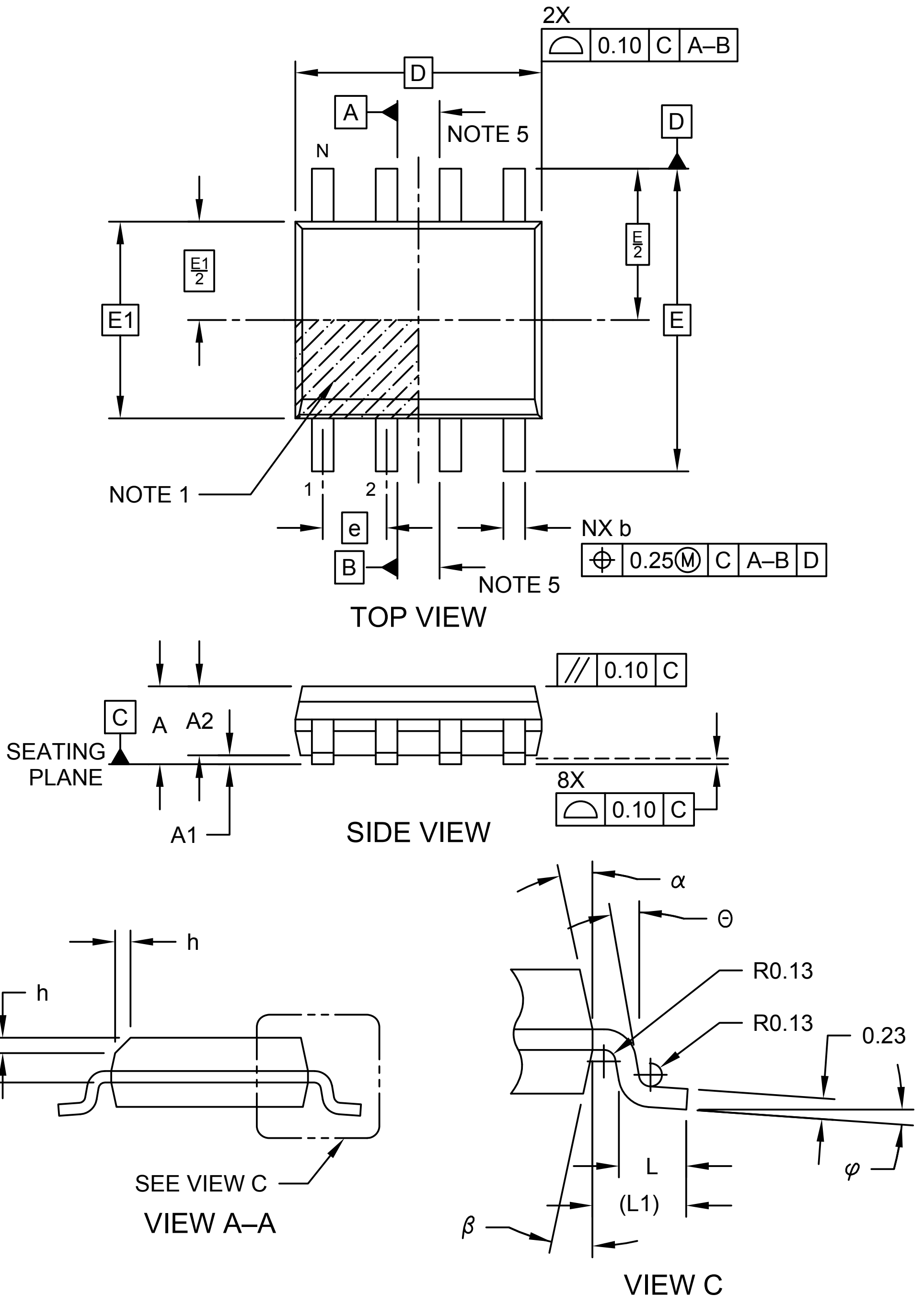
1. Dimensioning and tolerancing per ASME Y14.5M
   BSC: Basic Dimension. Theoretically exact value shown without tolerances.
2. For best soldering results, thermal vias, if used, should be filled or tented to avoid solder loss during reflow process

Microchip Technology Drawing C04-21355-Q4B Rev A

9.3 8-lead SOIC

8-Lead Plastic Small Outline - Narrow, 3.90 mm (.150 In.) Body [SOIC]
Atmel Legacy Global Package Code SWB

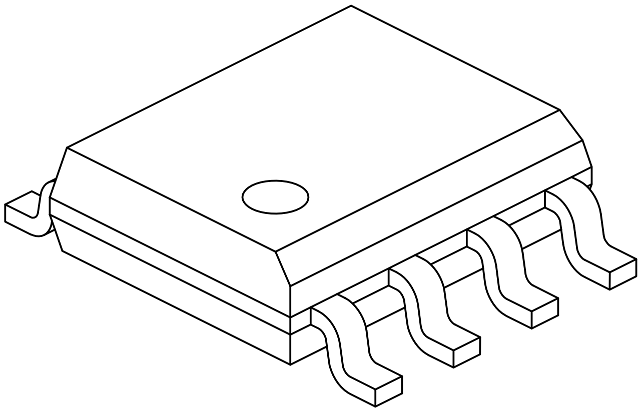
Note:	For the most current package drawings, please see the Microchip Packaging Specification located at http://www.microchip.com/packaging



Microchip Technology Drawing No. C04-057-SWB Rev E Sheet 1 of 2

8-Lead Plastic Small Outline - Narrow, 3.90 mm (.150 In.) Body [SOIC] Atmel Legacy Global Package Code SWB

Note: For the most current package drawings, please see the Microchip Packaging Specification located at http://www.microchip.com/packaging



Units		MILLIMETERS		
Dimension Limits		MIN	NOM	MAX
Number of Pins	N		8	
Pitch	e		1.27 BSC	
Overall Height	A	-	-	1.75
Molded Package Thickness	A2	1.25	-	-
Standoff §	A1	0.10	-	0.25
Overall Width	E		6.00 BSC	
Molded Package Width	E1		3.90 BSC	
Overall Length	D		4.90 BSC	
Chamfer (Optional)	h	0.25	-	0.50
Foot Length	L	0.40	-	1.27
Footprint	L1		1.04 REF	
Foot Angle	φ	0°	-	8°
Lead Thickness	c	0.17	-	0.25
Lead Width	b	0.31	-	0.51
Mold Draft Angle Top	α	5°	-	15°
Mold Draft Angle Bottom	β	5°	-	15°

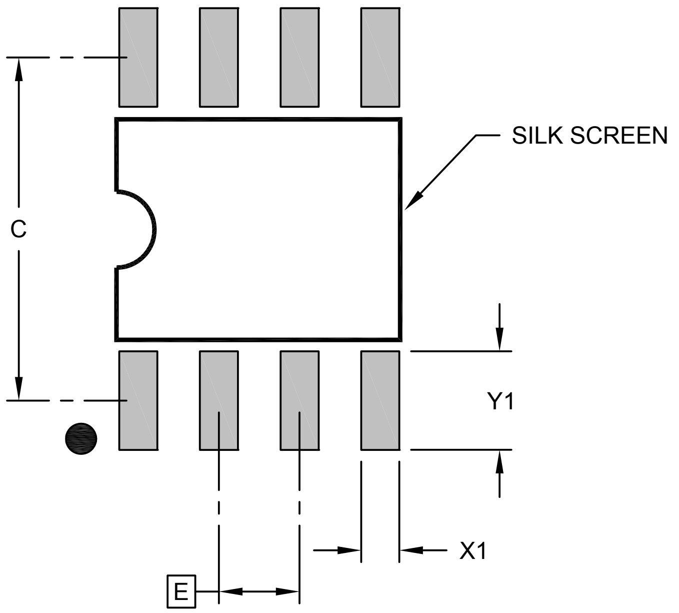
Notes:

1. Pin 1 visual index feature may vary, but must be located within the hatched area.
2. § Significant Characteristic
3. Dimensions D and E1 do not include mold flash or protrusions. Mold flash or protrusions shall not exceed 0.15mm per side.
4. Dimensioning and tolerancing per ASME Y14.5M
   BSC: Basic Dimension. Theoretically exact value shown without tolerances.
   REF: Reference Dimension, usually without tolerance, for information purposes only.
5. Datums A & B to be determined at Datum H.

Microchip Technology Drawing No. C04-057-SWB Rev E Sheet 2 of 2

8-Lead Plastic Small Outline - Narrow, 3.90 mm (.150 In.) Body [SOIC] Atmel Legacy Global Package Code SWB

Note:	For the most current package drawings, please see the Microchip Packaging Specification located at http://www.microchip.com/packaging



RECOMMENDED LAND PATTERN

	Units	MILLIMETERS		
	Dimension Limits	MIN	NOM	MAX
Contact Pitch	E	1.27 BSC		
Contact Pad Spacing	C		5.40	
Contact Pad Width (X8)	X1			0.60
Contact Pad Length (X8)	Y1			1.55

Notes:

1. Dimensioning and tolerancing per ASME Y14.5M
   BSC: Basic Dimension. Theoretically exact value shown without tolerances.

Microchip Technology Drawing C04-2057-SWB Rev E

10. Revision History

Revision A (August 2019)
Original release of the document.

The Microchip Website

Microchip provides online support via our website at http://www.microchip.com/. This website is used to make files and information easily available to customers. Some of the content available includes:

- **Product Support** – Data sheets and errata, application notes and sample programs, design resources, user's guides and hardware support documents, latest software releases and archived software
- **General Technical Support** – Frequently Asked Questions (FAQs), technical support requests, online discussion groups, Microchip design partner program member listing
- **Business of Microchip** – Product selector and ordering guides, latest Microchip press releases, listing of seminars and events, listings of Microchip sales offices, distributors and factory representatives

Product Change Notification Service

Microchip's product change notification service helps keep customers current on Microchip products. Subscribers will receive email notification whenever there are changes, updates, revisions or errata related to a specified product family or development tool of interest.

To register, go to http://www.microchip.com/pcn and follow the registration instructions.

Customer Support

Users of Microchip products can receive assistance through several channels:

- Distributor or Representative
- Local Sales Office
- Embedded Solutions Engineer (ESE)
- Technical Support

Customers should contact their distributor, representative or ESE for support. Local sales offices are also available to help customers. A listing of sales offices and locations is included in this document.

Technical support is available through the website at: http://www.microchip.com/support

Product Identification System

To order or obtain information, e.g., on pricing or delivery, refer to the factory or the listed sales office.

PART NO.	X	-X
Device	Package Type	Tape and Reel

Device:	ATECC608A-TNGTLS: Pre-configured Cryptographic Co-processor with secure hardware-based key storage for the IoT market.	
Package Options	U	8-Pad 2 x 3 x 0.6 mm Body, Thermally Enhanced Plastic Ultra Thin Dual Flat No Lead Package (UDFN)
	S	8-Lead (0.150” Wide Body), Plastic Gull Wing Small Outline (JEDEC SOIC)
Tape and Reel Options	G	2k Reel
	C	100 Unit Reel
	B	10 Unit Bulk- Prototype units

Examples:
- ATECC608A-TNGTLSU-G: Trust&GO TLS, Provisioned, 8-UDFN, I^2C, 2K Reel
- ATECC608A-TNGTLSU-C: Trust&GO TLS, Provisioned, 8-UDFN, I^2C, 100 Unit Reel
- ATECC608A-TNGTLSU-B: Trust&GO TLS, Proto Typing, 8-UDFN, I^2C, 10 Unit Bulk
- ATECC608A-TNGTLSS-G: Trust&GO TLS, Provisioned, 8-SOIC, I^2C, 2K Reel
- ATECC608A-TNGTLSS-C: Trust&GO TLS, Provisioned, 8-SOIC, I^2C, 100 Unit Reel
- ATECC608A-TNGTLSS-B: Trust&GO TLS, Proto Typing, 8-SOIC, I^2C, 10 Unit Bulk

Note:

1. Tape and Reel identifier only appears in the catalog part number description. This identifier is used for ordering purposes and is not printed on the device package. Check with your Microchip Sales Office for package availability with the Tape and Reel option.

Microchip Devices Code Protection Feature

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

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

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

Legal Notice

Information contained in this publication regarding device applications and the like is provided only for your convenience and may be superseded by updates. It is your responsibility to ensure that your application meets with your specifications. MICROCHIP MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND WHETHER EXPRESS OR IMPLIED, WRITTEN OR ORAL, STATUTORY OR OTHERWISE, RELATED TO THE INFORMATION, INCLUDING BUT NOT LIMITED TO ITS CONDITION, QUALITY, PERFORMANCE, MERCHANTABILITY OR FITNESS FOR PURPOSE. Microchip disclaims all liability arising from this information and its use. Use of Microchip devices in life support and/or safety applications is entirely at the buyer's risk, and the buyer agrees to defend, indemnify and hold harmless Microchip from any and all damages, claims, suits, or expenses resulting from such use. No licenses are conveyed, implicitly or otherwise, under any Microchip intellectual property rights unless otherwise stated.

Trademarks

The Microchip name and logo, the Microchip logo, Adaptec, AnyRate, AVR, AVR logo, AVR Freaks, BesTime, BitCloud, chipKIT, chipKIT logo, CryptoMemory, CryptoRF, dsPIC, FlashFlex, flexPWR, HELDO, IGLOO, JukeBlox, KeeLoq, Kleer, LANCheck, LinkMD, maXStylus, maXTouch, MediaLB, megaAVR, Microsemi, Microsemi logo, MOST, MOST logo, MPLAB, OptoLyzer, PackeTime, PIC, picoPower, PICSTART, PIC32 logo, PolarFire, Prochip Designer, QTouch, SAM-BA, SenGenuity, SpyNIC, SST, SST Logo, SuperFlash, Symmetricom, SyncServer, Tachyon, TempTrackr, TimeSource, tinyAVR, UNI/O, Vectron, and XMEGA are registered trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

APT, ClockWorks, The Embedded Control Solutions Company, EtherSynch, FlashTec, Hyper Speed Control, HyperLight Load, IntelliMOS, Libero, motorBench, mTouch, Powermite 3, Precision Edge, ProASIC, ProASIC Plus, ProASIC Plus logo, Quiet-Wire, SmartFusion, SyncWorld, Temux, TimeCesium, TimeHub, TimePictra, TimeProvider, Vite, WinPath, and ZL are registered trademarks of Microchip Technology Incorporated in the U.S.A.

Adjacent Key Suppression, AKS, Analog-for-the-Digital Age, Any Capacitor, AnyIn, AnyOut, BlueSky, BodyCom, CodeGuard, CryptoAuthentication, CryptoAutomotive, CryptoCompanion, CryptoController, dsPICDEM, dsPICDEM.net, Dynamic Average Matching, DAM, ECAN, EtherGREEN, In-Circuit Serial Programming, ICSP, INICnet, Inter-Chip Connectivity, JitterBlocker, KleerNet, KleerNet logo, memBrain, Mindi, MiWi, MPASM, MPF, MPLAB Certified logo, MPLIB, MPLINK, MultiTRAK, NetDetach, Omniscient Code Generation, PICDEM, PICDEM.net, PICkit, PICtail, PowerSmart, PureSilicon, QMatrix, REAL ICE, Ripple Blocker, SAM-ICE, Serial Quad I/O, SMART-I.S., SQI, SuperSwitcher, SuperSwitcher II, Total Endurance, TSHARC, USBCheck, VariSense, ViewSpan, WiperLock, Wireless DNA, and ZENA are trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

SQTP is a service mark of Microchip Technology Incorporated in the U.S.A.

The Adaptec logo, Frequency on Demand, Silicon Storage Technology, and Symmcom are registered trademarks of Microchip Technology Inc. in other countries.

GestIC is a registered trademark of Microchip Technology Germany II GmbH & Co. KG, a subsidiary of Microchip Technology Inc., in other countries.

All other trademarks mentioned herein are property of their respective companies.

© 2019, Microchip Technology Incorporated, Printed in the U.S.A., All Rights Reserved.

ISBN: 978-1-5224-4864-8

Quality Management System

For information regarding Microchip's Quality Management Systems, please visit http://www.microchip.com/quality.

Worldwide Sales and Service

AMERICAS

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

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

Austin, TX
Tel: 512-257-3370

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

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

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

Detroit
Novi, MI
Tel: 248-848-4000

Houston, TX
Tel: 281-894-5983

Indianapolis
Noblesville, IN
Tel: 317-773-8323
Fax: 317-773-5453
Tel: 317-536-2380

Los Angeles
Mission Viejo, CA
Tel: 949-462-9523
Fax: 949-462-9608
Tel: 951-273-7800

Raleigh, NC
Tel: 919-844-7510

New York, NY
Tel: 631-435-6000

San Jose, CA
Tel: 408-735-9110
Tel: 408-436-4270

Canada - Toronto
Tel: 905-695-1980
Fax: 905-695-2078

ASIA/PACIFIC

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

China - Beijing
Tel: 86-10-8569-7000

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

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

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

China - Guangzhou
Tel: 86-20-8755-8029

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

China - Hong Kong SAR
Tel: 852-2943-5100

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

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

China - Shanghai
Tel: 86-21-3326-8000

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

China - Shenzhen
Tel: 86-755-8864-2200

China - Suzhou
Tel: 86-186-6233-1526

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

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

China - Xiamen
Tel: 86-592-2388138

China - Zhuhai
Tel: 86-756-3210040

ASIA/PACIFIC

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

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

India - Pune
Tel: 91-20-4121-0141

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

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

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

Korea - Seoul
Tel: 82-2-554-7200

Malaysia - Kuala Lumpur
Tel: 60-3-7651-7906

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

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

Singapore
Tel: 65-6334-8870

Taiwan - Hsin Chu
Tel: 886-3-577-8366

Taiwan - Kaohsiung
Tel: 886-7-213-7830

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

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

Vietnam - Ho Chi Minh
Tel: 84-28-5448-2100

EUROPE

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

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

Finland - Espoo
Tel: 358-9-4520-820

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

Germany - Garching
Tel: 49-8931-9700

Germany - Haan
Tel: 49-2129-3766400

Germany - Heilbronn
Tel: 49-7131-72400

Germany - Karlsruhe
Tel: 49-721-625370

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

Germany - Rosenheim
Tel: 49-8031-354-560

Israel - Ra'anana
Tel: 972-9-744-7705

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

Italy - Padova
Tel: 39-049-7625286

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

Norway - Trondheim
Tel: 47-72884388

Poland - Warsaw
Tel: 48-22-3325737

Romania - Bucharest
Tel: 40-21-407-87-50

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

Sweden - Gothenberg
Tel: 46-31-704-60-40

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

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