Atmel CryptoAuthentication

ATECC508A Public Key Validation

APPLICATION NOTE

Introduction

The Atmel® CryptoAuthentication™ ATECC508A device supports validated public keys if keyConfig.PubInfo is set to one for the slot containing the public key. If configured in this manner, the public key is invalid by default when loaded and can only be used after the appropriate validation operation has been successfully completed. Reasons why this capability is useful are:

- To store the validation status of a client public key to speed up subsequent authorizations. In this manner the certificate chain needs be verified only once, and subsequent authorizations will only require validation of the signature of the random nonce.
- To allow controlled updates of public keys used to verify software or other data, as would be appropriate if the key is revoked.

The validity status is stored in protected memory which can be modified only in accordance with the security policies set in the device's Configuration zone.

This following functionality is optional:

- If PubInfo is set to zero, then public keys can be used by the `Verify(External)` command immediately after they are loaded.

The remainder of this document discusses handling of public keys for which the validation function is intended and PubInfo is set to one. In addition, this document refers to the operation after the Data zone is locked. Prior to locking, most slots can be written without restriction. This document describes the command sequences in terms of three keys:

- **Child Key** —The public key to be written and validated. Must be a slot number from 8 to 15.
- **Encryption Key** — The symmetric key used to authorize the write. Its slot number is SlotConfig[Child].WriteKey. It may reside in any slot.
- **Parent Key** — The public key used to validate the child key. Its slot number is SlotConfig[Child].ReadKey. It must resolve to slot number from 8 to 15.

These sequences can be used hierarchically to validate a longer certificate chain if the various key links are so configured.

Table of Contents

1. Overview

There are three ways to write public keys into the device depending on SlotConfig[Child].WriteConfig. Command details are in the Public Key Write Sequences section. All writes invalidate the public key:

- **Standard Open Write** — No special authorization is required.
- **Authenticated Write** — Requires knowledge of a symmetric secret to generate an authorizing MAC for the data. It uses the `Write(Encrypt)` command which takes the input of the encrypted public key and authorizing MAC.
- **PubValid Write** — The PubValid Write Can be written in the clear without authorization only if the existing public key in the slot is invalid. If the PubKey is valid, then writes are forbidden.

There are two ways to validate a public key depending upon the contents of the X.509 formatting bytes within the Configuration zone, X509format[KeyConfig[Child].X509id]. Both use the `Verify` command. Command details are in the Public Key Validation Sequences section.

- **Verify(Validate)** — With this method, the device receives a signature generated by the parent over an Atmel-defined message covering the child public key. Depending upon the details of the message, this command can be used to invalidate a key to allow the PubValid write mode to be executed.
- **Verify(ValidateExternal)** — With this method, the device receives an X.509 certificate generated by the parent over a standard ASN.1 message covering the child public key. Keys validated in this manner can only be invalidated with the Write command

Additional ATECC508A features that come into play regarding public key updates are Random Nonce and Authorization which are addressed below.

1.1. Random Nounce

The KeyConfig.ReqRandom bit is designed to allow or deny use of a fixed cryptographic sequence. Another way to think of this is that if ReqRandom is zero then a stored token can be used to repeat the sequence at various times or in various places if the public keys are repeated. If ReqRandom is one, then the sequence must be individually computed for a specific device at that specific time.

If ReqRandom is set for the Child, then a random nonce will be required for execution of the Verify(ValidateExternal) sequence described in the Public Key Validation Sequences section, but not the Verify(Validate) sequence. If ReqRandom is set for the Parent, then a random nonce will be required for both sequences.

The `Write` command does not honor ReqRandom. In the case of the Authorized Write sequence, the `GenDig` command listed in the Public Key Write Sequences section implements this restriction regarding the encryption key.

All sequences are listed in the following sections assuming that the ReqAuth bits are set to zero.

1.2. Authorization

The `Verify` command does not observe the ReqAuth for the child slot to be validated. ReqAuth is honored for the parent key (KeyConfig[Child].AuthKey). In the validation sequences described below in the Public Key Validation Sequences section, multiple commands are utilized to prepare for the final `Verify` command. Those that access a public key DO honor the relevant ReqAuth bit for either Child or Parent.

The `Write` command does not observe the ReqAuth bit on the child slot. Authorization can be used for the authenticated write operation if the ReqAuth bit is set for the Encryption key. The authorization is required prior to the `GenDig` operation listed in the Public Key Write Sequences section.

All sequences are listed below assuming that the ReqAuth bits are set to zero.

2. Public Key Write Sequences

2.1. Open Write

Table 2-1. Open Write

Command	Mode	Param2	Notes
Write	0x82	Child	First half of public key. No MAC required.
Write	0x82	Child	Second half. No MAC required.

2.2. Authenticated Write

Table 2-2. Authenticated Write

Command	Mode	Param2	Notes
Nounce	Fixed	—	
GenDig	Data	Enc Key	Load decryption value into TempKey.
Write	0x82	Child	First half. MAC is required.
Nonce	Fixed	—	
GenDig	Data	Enc Key	Load decryption value into TempKey.
Write	0x82	Child	Second half. MAC is required.

2.3. PubValid Write

This sequence includes both the invalidation commands for the currently stored public key and writing of the new public key.

Table 2-3. PubValid Write

Command	Mode	Param2	Notes
Nonce	Fixed	—	
GenKey	0x10	Child	Add digest of Child public key to TempKey.
Verify	Invalidate	Child	Data is invalidation (revocation) signature.
Write	0x82	Child	First half of new public key. No MAC required.
Write	0x82	Child	Second half. No MAC required.

3. Public Key Validation Sequences

3.1. Verify(Validate)

The Validate mode is a more efficient mode for public key validation as compared to ValidateExternal, requiring fewer commands. Its input is a signature computed from the child public key by the parent. Because the entire message template is controlled by the device, it can be used for both validation and invalidation.

Table 3-1. Verify(Validate)

Command	Mode	Param2	Notes
Nonce	Fixed	—	
GenKey	0x10	Child	Add digest of Child public key to TempKey.
Verify	Validate	Child	Signed by Parent. Reference the Message Generation for Verify(Validate) section for message format.

3.2. Verify(ValidateExternal)

The ValidateExternal mode is designed to permit the public keys to be validated with standard X.509 formatted certificates. The X509format field in the Configuration zone constrains the construction of the ASN.1 template to the particular template expected to be used. The X509id field in the KeyConfig field permits different templates at different locations within the key hierarchy to have different structures.

Note: The SHA(Update) command requires the full 64 byte SHA blocks to be inputted into the device. The X.509 template should be adjusted to make sure that complete blocks occur before the public key. This can usually be achieved by adjusting the length of the names.

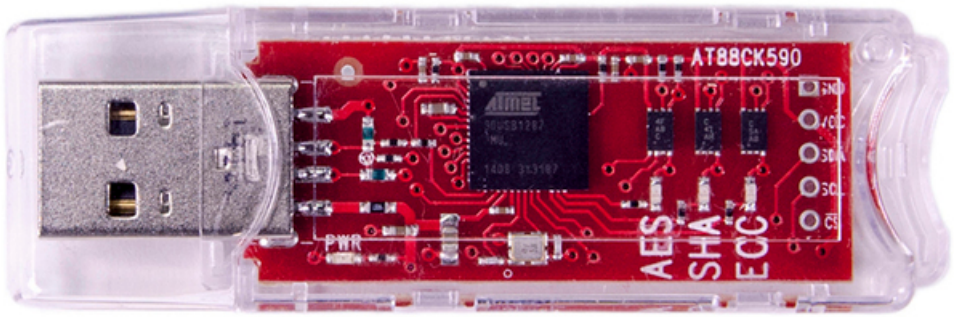
Table 3-2. Verify(Validate)

Command	Mode	Param2	Notes
SHA	Start	—	
SHA	Update		Repeated (X509format.PublicPosition) times.
SHA	Public	Child	Add digest of child public key to digest.
SHA	Update		Repeated (X509format.TemplateLength – X509format.PublicPosition - 1) times.
SHA	End		
Verify	ValExt	Child	

4. Message Generation for Verify(Validate)

The message that should be signed to create the validation signature for Verify(Validate) sequence is described below. This signature can be generated by any system containing the Parent private key, or it can be generated by an ATECC508A that contains the Parent private key. A board such as the Atmel AT88CK590 USB Evaluation Kit can serve this purpose.

Figure 4-1. AT88CK590 Evaluation Kit



Note: It is critical that if the ATECC508A is used to store the parent private key, it must be backed up externally and written to the ATECC508A device using PrivWrite. In this manner if there is a board or device failure signatures can still be generated.

The device should be configured to have a Child public key slot in which PubInfo is one. The slot containing the Parent private key should be marked (via SlotConfig[ParentPriv].ReadKey to allow internal signatures. The slot number for the Child public key should be the same in the signing module as in the devices that are in the field. The particular slot numbers into which the Parent Key is written does not matter. The sequence below assumes that the Child public key slot can be written without a MAC; SlotConfig[Child].WriteConfig = ALWAYS.

Table 4-1. Verify(Validate)

Command	Mode	Param2	Notes
Write	0x82	Child	First half of public key. No MAC required.
Write	0x82	Child	Second half. No MAC required.
Nonce	Fixed	—	
GenKey	0x10	Child	Add digest of Child public key to TempKey.
Sign	Internal	ParentPriv	Generates Validation signature.
Verify	Validate	Child	Use signature from above. Validates Child public key.
Nonce	Fixed	—	
GenKey	0x10	Child	Add digest of Child public key to TempKey.
Sign	Internal	ParentPriv	Generates Invalidation signature.

The message signed by the Sign(Internal) steps above is generated as listed below. The preceding `GenKey` command generates the TempKey value from the following message:

Bytes	Description
32	Input value to `Nonce(Fixed)` command
1	GenKey Opcode (0x40)
1	GenKey Opcode (0x40)
2	GenKey Child
1	SN[8]
2	SN[0:1]
25	Zeros
64	X and Y coordinates of the Child public key

Then this TempKey value is used by Sign(Internal) to create a second level message that is then hashed using SHA-256 to create the digest that is signed.

Bytes	Description
32	TempKey
1	Sign Opcode (0x41)
1	Mode (0x80)
2	ParentPriv
2	SlotConfig[Child]
2	KeyConfig[Child]
1	TempKeyFlags
1	UseFlag[Child]
1	UpdateCount[Child]
1	SN[8]
4	0x00
2	SN[0:1]
2	0x00
1	SlotLocked:TempKeyFlags.Child
1	0x00 (for Validation signature, 0x01 for Invalidation signature)
1	0x00

The values of the 19 bytes highlighted in bold above should be passed to the corresponding `Verify(Validate)` command on the signing module or device in the field in order to allow the validation to succeed. Other than the state of key validation byte (OtherData[17], the second from the last byte) these values are not critical to the success of the `Verify(Validate)` command so long as they match those used to generate the signatures.

5. Revision History

Table 5-1. Revision History

Doc. Rev.	Date	Comments
8932A	01/2016	Initial document release.

Atmel | Enabling Unlimited Possibilities®

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