Atmel

APPLICATION NOTE

Compressed Certificate Definition

ATECC CryptoAuthentication Device Family

Introduction

Provides the details required to integrate an Atmel® ATECC CryptoAuthentication™ device and the AT88CKECCSIGNER Module Kit into a third party certificate chain.

Features

- Compressed Certificate Details
- Certificate Template
- Support for X.509 Certificates
- Atmel ATECC Device Supported Authentication Chaining Overview

Table of Contents

1 Technical Overview 3

1.1 Provisioning Overview 3
1.2 Device Authentication Overview 4

2 Compressed Certificates 5

2.1 Supported ATECC Chain 5
2.2 P256 Compressed Certificate 6
2.3 Signature 6
2.4 Encoded Dates 6
2.5 Signer ID 8
2.6 Template ID 8
2.7 Chain ID 8
2.8 Serial Number Source 8
2.8.2 Stored Serial Number (SN Source = `0x0`) 9
2.8.3 Generated from Subject Public Key (SN Source = `0xA`) 9
2.8.4 Generated from Device Serial Number (SN Source = `0xB`) 10
2.9 Compressed Certificate Format Version 10
2.10 Reserved Byte 10

3 X.509 Certificate 11

3.1 X.509 Compressed Certificate Elements 11
3.2 Signature Reconstruction 12
3.2.1 Encoding steps 14
3.2.2 References 14

4 Revision History 14

1 Technical Overview

1.1 Provisioning Overview

Prior to deployment, the ATECC device needs to be provisioned. This consists of defining the configuration and the programming of device memory with secrets and other application specific data. Reference the Atmel Security Provisioning Kits documents at: www.atmel.com/tools/AT88CKECCROOT-SIGNER.aspx. The AT88CKECCROOT and AT88CKECCSIGNER Module Kit user guides direct links are listed below:

www.atmel.com/Images/Atmel-8967-CryptoAuth-AT88CKECCROOT-UserGuide.pdf

www.atmel.com/Images/Atmel-8969-CryptoAuth-AT88CKECCSIGNER-UserGuide.pdf

For shared secret data and keys, the protected memory of the device is written so the keys can be used in the secure execution environment of the device, and secure memory can be accessed with encrypted reads.

For ECC keys, the device internally creates a unique private key, stores the private key into a protected key slot, calculates the associated public key, then returns the public key. The private key can then be used to sign messages and the public key will be signed into a trusted chain.

Device Provisioning Steps

1. Send a command to the device to create a unique public-private key pair.
2. Construct a *To Be Signed* (TBS) certificate for the device using the device certificate template, the device public key, and other device-specific certificate elements.
3. A SHA-256 digest of the TBS certificate is signed by the Signer.
4. Sign the TBS digest using the signing module.
5. The compressed certificate of the device is written to the device. Compressed certificate includes:
   - Device public key
   - Signature of TBS by Signer
   - Device-specific certificate elements
6. The compressed certificate of the signer is written to the device. Compressed certificate includes:
   - Signer public key
   - Signature of Signer TBS by Issuer
   - Signer-specific certificate elements

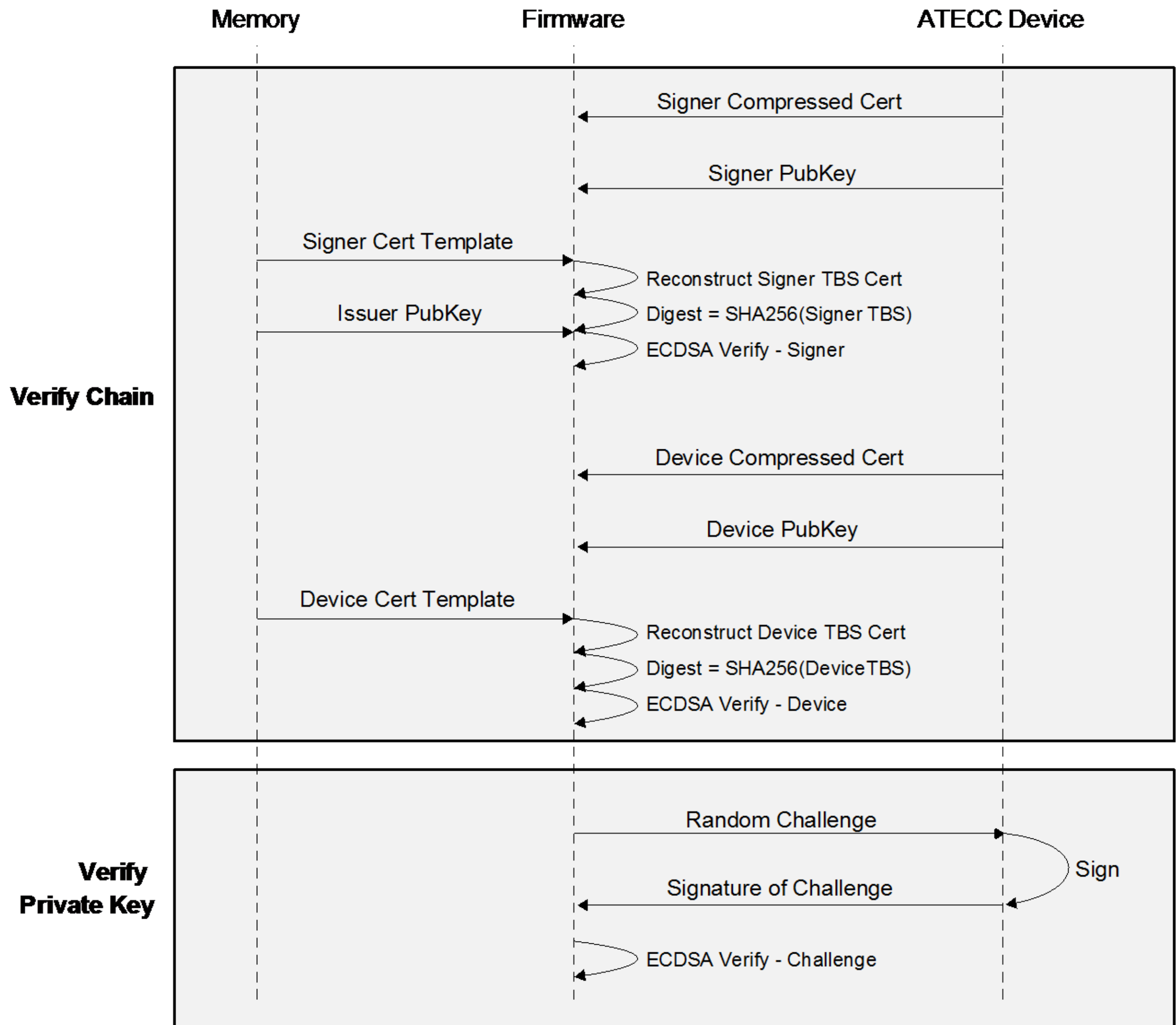
The provisioning results in the device containing two certificates for the device itself and the signer that signed it.

1.2 Device Authentication Overview

Since the signing of the device is performed by the signer, it allows Atmel to produce devices that can be validated into the certificate chain while managing the other production considerations.

With this scheme, the device authentication only needs the issuer certificate to be installed on the platform. All remaining components of the device certificate and its signer certificate can be produced via information stored within the device. The following sequence diagram shows the authentication of the device.

Figure 1-1. Device Authentication Sequence Diagram



2 Compressed Certificates

Since the ATECC device does not have enough storage for two full X.509 certificates, the concept of a Compressed Certificate and a Certificate Template is introduced. The Compressed Certificate includes the Public Key, Certificate Elements, and the Signature. These elements can be combined with the Certificate Template to construct the original Certificate.

This certificate reconstruction is required for both the device and the signer of the device; therefore, the details are defined for both certificates types. Each device stores the elements to reconstruct both the Signer Certificate and the Device Certificate.

2.1 Supported ATECC Chain

The device is preconfigured as shown in the previous Section 1.2, “Device Authentication Overview” of this document. All elements required for authenticating the chain through the Issuer are stored on the device.

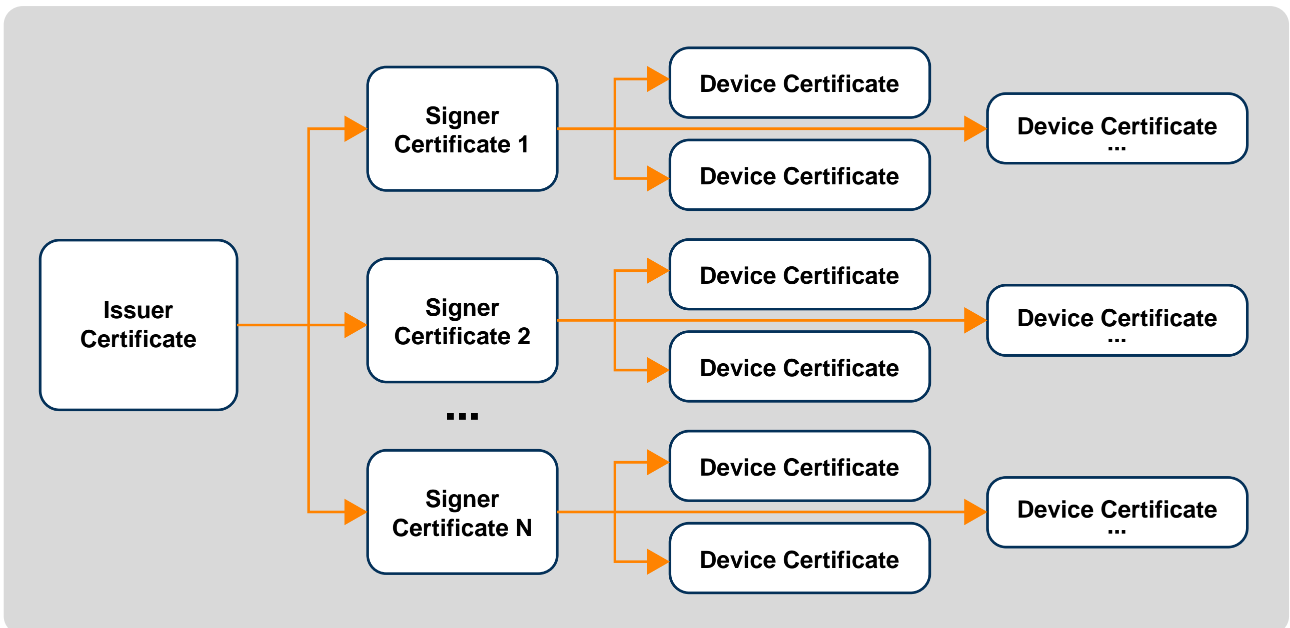
The certificate chain that is supported by the device is defined in the table below.

Table 2-1. Certificate Chain Definition

Certificate	Signed By	Signer For	Notes
Device	Signer	Challenge	A challenge is sent to the device to demonstrate possession of the Private Key that is associated with the Public Key in the Device Certificate.
Signer	Issuer	Device	The Signer Private Key is contained in the Signing Module that is used in production.
Issuer	Root	Signer	Used as the root in some systems and can be a Root Signing Module.
Optional Root	Optional Root	Issuer	Represents a chain of one or more levels above the Issuer to include Certificate Authorities. These certificates may use larger ECC curves.

Note: The Issuer, Signer, and Device Certificates *must* use the same ECC curve.

Figure 2-1. Certificate Chain Diagram



2.2 P256 Compressed Certificate

For P256 curves, the Compressed Certificate can be stored with the signature in one of the 72 byte slots (8 to15) of the ATECC device.

The P256 Compressed Certificate is defined as follows:

Figure 2-2. P256 Compressed Certificate

<table>
<tr><td colspan="10">Signature: 64 bytes
Bytes 0 to 63</td></tr>
<tr><td>Byte 64</td><td>Byte 65</td><td>Byte 66</td><td>Byte 67</td><td>Byte 68</td><td colspan="2">Byte 69</td><td colspan="2">Byte 70</td><td>Byte 71</td></tr>
<tr><td colspan="3">Encoded Dates
(24 bits)</td><td colspan="2">Signer ID
(16 bits)</td><td>Template ID
(4 bits)</td><td>Chain ID
(4 bits)</td><td>SN Source
(4 bits)</td><td>Format Version
(4 bits)</td><td>Reserved
(8 bits)</td></tr>
</table>

Table 2-2. P256 Compressed Certificate Definition

Element	Size (bits)	Description
Signature	512	Certificate signature stored as the 32 byte R and S unsigned big-endian integers.
Encoded Dates	24	Certificate issue and expiration dates in a bit-packed format.
Signer ID	16	ID of the specific signer used to sign the certificate (device cert) or of the signer itself (signer cert).
Template ID	4	ID the certificate template to be used to reconstruct the full X.509 certificate.
Chain ID	4	ID of the certificate chain being used.
SN Source	4	Indicates where to find or how to generate the certificate serial number.
Format Version	4	Version of the compressed certificate format. `0` is the only version.
Reserved	8	Reserved byte.

2.3 Signature

An elliptic curve signature has two components, R and S. For a P256 curve, these components are 32 bytes. The signature in the compressed certificate stores the R integer first, then S in an unsigned big-endian format.

2.4 Encoded Dates

The encoded dates are three bytes that represent the issue and expiration dates of the certificate in a compressed (bit packed) format. The issue date is assumed to be UTC.

Table 2-3. Encoded Date Format

<table>
<tr><td colspan="8">Byte 0</td><td colspan="8">Byte 1</td><td colspan="8">Byte 2</td></tr>
<tr><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td colspan="5">Year
5 bits</td><td colspan="4">Month
4 bits</td><td colspan="5">Day
5 bits</td><td colspan="5">Hour
5 bits</td><td colspan="5">Expire Years
5 bits</td></tr>
</table>

Table 2-4. Encoded Dates Definition

Element	Size (bits)	Description	Range
Year	5	Issue date year starting from 2000.	0 to 31 (2000 to 2031)
Month	4	Issue date month.	1 to 12
Day	5	Issue date day.	1 to 31
Hour	5	Issue date hour.	0 to 23
Expire Years	5	How many years the certificate is valid for.	0 to 31

The expiration date is a set number of years (expire years field) from the issue date. Use `0` for expire years to indicate no expiration date.

The issue date only has a resolution of hours. Minutes and seconds are assumed to be zero.

Example 1: Parsing Encoded Dates Example

The example encoded date represents the following dates:

Issue date: 2014-10-15 16:00:00 UTC
Expire date: 2028-10-15 16:00:00 UTC (14 years after issue date)

```
unsigned char enc_dates[] = {0x75, 0x3E, 0x0E};

int issue_date_year  = (enc_dates[0] >> 3) + 2000;
int issue_date_month = ((enc_dates[0] & 0x07) << 1) | ((enc_dates[1] & 0x80) >> 7);
int issue_date_day   = ((enc_dates[1] & 0x7C) >> 2);
int issue_date_hour  = ((enc_dates[1] & 0x03) << 3) | ((enc_dates[2] & 0xE0) >> 5);
int expire_years     = (enc_dates[2] & 0x1F);
```

If the expiration in years is `0`, then the certificate is supposed to have no expiration date. Since X.509 certificates have no provision for this, the maximum possible date is used. X.509 defines two different date formats in RFC 5280.

For the UTCTime format as defined by section 4.1.2.5.1, the maximum possible date is:

```
491231235959Z      2049-12-31 23:59:49 UTC
```

For the GeneralizedTime format as defined by section 4.1.2.5.2, the maximum possible date is:

```
99991231235959Z    9999-12-31 23:59:59 UTC
```

2.5 Signer ID

The Signer ID is a two byte identifier for the specific signer that is used to sign the certificate. The Atmel internal provisioning system uses the following format.

Figure 2-3. Signer ID Format

Byte 0								Byte 1							
7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Module ID 8 bits								ECC ID 4 bits				Slot ID 4 bits			

Table 2-5. Signer ID Definition

Element	Size (bits)	Description
Module ID	8	Specific signing module used.
ECC ID	4	ECC device on the specified signing module used.
Slot ID	4	Slot on the specified ECC device used.

2.6 Template ID

The Template ID provides the necessary information to expand a compressed certificate into a full X.509 certificate. A typical chain may have a different template for the signer certificate than for the device certificate. The Template ID allows for up to 16 different templates to be used for unique templates at each level.

Table 2-6. Conventions

Template ID	Description
0	Use the Device Template.
1	Use the Signer Template.
n	Issuer Template or higher.

2.7 Chain ID

The Chain ID provides the necessary information to allow for multiple certificate chains to be defined for a particular customer. Most customers have only one certificate chain that uses a Chain ID of `0`.

2.8 Serial Number Source

The Serial Number Source determines where the serial number comes from when reconstructing the X.509 certificate from the compressed certificate. A typical X.509 certificate serial number is at least 8 bytes, can be up to 20 bytes, and must be unique. Each certificate must have a different serial number and there is no room in the compressed certificate slot to store it. Therefore, the serial number needs to be either stored in another slot on the ATECC device or generated from already known data that would result in a unique value. In all of the options listed, the most significant two bits should be forced to `01`. This ensures the serial number is positive and un-trimmable per RFC 5280 Section 4.1.2.2 and ASN.1 integer encoding.

Table 2-7. Serial Number Source

SN Source	Value	Description
Stored SN	`0x0`	Random number generated and written to a slot.
Generated with Public Key	`0xA`	Use the Public Key and encoded dates to generate the certificate serial number.
Generated with Device SN	`0xB`	Use the unique device serial number and encoded dates to generate the certificate serial number.

2.8.2 Stored Serial Number (SN Source = `0x0`)

During certificate generation, the serial number is a random number. The resulting serial number is stored in a slot on the ATECC device. The two upper most significant bits are set to `01`. When reconstructing the certificate, the serial number is read from the slot where it was saved and inserted into the certificate template.

```
const int cert_sn_size = 16;
unsigned char cert_sn[cert_sn_size];
// Fill the serial number with random data (pseudo-function)
random_bytes(cert_sn, cert_sn_size);
// Set two most significant bits to 01 to ensure a positive, untrimmable serial number
cert_sn[0] &= 0x7F;
cert_sn[0] |= 0x40;
```

2.8.3 Generated from Subject Public Key (SN Source = `0xA`)

Serial number is generated from a hash of the subject public key (device public key for the device certificate and signer public key for the signer certificate) and encoded dates from the compressed certificate:

- SHA256(subject public key [64 bytes] + encoded dates [3 bytes])
- Two upper most significant bits are set to `01`

This method uses already known information to generate the serial number and doesn't need to be saved to a slot. Serial number should be truncated to the size of the serial number specified for the certificate. Refer to the ECC Key Formatting section in the datasheet for information on how ECC public keys are stored in the EEPROM slots.

```
const int cert_sn_size = 16;
unsigned char cert_sn[cert_sn_size];
unsigned char subject_public_key[72];
unsigned char comp_cert[72];
unsigned char msg[67];
unsigned char digest[32];
// Read the subject key from the relevant slot (pseudo-function)
get_subject_public_key(subject_public_key);
// Read the compressed certificate from the relevant slot (pseudo-function)
get_compressed_cert(comp_cert);
// Build the input message for the hash
memcpy(&msg[0], &subject_public_key[4], 32);   // Subject public key, X
memcpy(&msg[32], &subject_public_key[40], 32); // Subject public key, Y
memcpy(&msg[64], &comp_cert[64], 3);           // Encoded dates from comp cert
// Perform the SHA256 hash on the message and put the results into digest
sha256(msg, 67, digest);
// Copy only the portion of the digest for the serial number size
memcpy(cert_sn, digest, cert_sn_size);
// Set two most significant bits to 01 to ensure a positive, untrimmable serial number
cert_sn[0] &= 0x7F;
cert_sn[0] |= 0x40;
```

2.8.4 Generated from Device Serial Number (SN Source = `0xB`)

Serial number is generated from a hash of the device's serial number and encoded dates from the compressed certificate:

- SHA256(device SN [9 bytes] + encoded dates [3 bytes])
- Two upper most significant bits are set to `01`

This method uses already known information to generate the serial number and doesn't need to be saved to a slot. Serial number should be truncated to the size of the serial number specified for the certificate. This method is also only available to the device certificate, since the signer has no "device serial number".

```
const int cert_sn_size = 16;
unsigned char cert_sn[cert_sn_size];
unsigned char device_config[128];
unsigned char comp_cert[72];
unsigned char msg[12];
unsigned char digest[32];
// Read the config zone from the device (pseudo-function)
get_device_config(device_config);
// Read the compressed certificate from the relevant slot (pseudo-function)
get_compressed_cert(comp_cert);
// Build the input message for the hash
memcpy(&msg[0], &device_config[0], 4); // Device SN[0:3] from config zone
memcpy(&msg[4], &device_config[8], 5); // Device SN[4:8] from config zone
memcpy(&msg[9], &comp_cert[64], 3);    // Encoded dates from compressed cert
// Perform the SHA256 hash on the message and put the results into digest
sha256(msg, 12, digest);
// Copy only the portion of the digest for the serial number size
memcpy(cert_sn, digest, cert_sn_size);
// Set most significant bit to 0 to ensure a positive serial number
cert_sn[0] &= 0x7F;
// Set two most significant bits to 01 to ensure a positive, untrimmable serial number
cert_sn[0] &= 0x7F;
cert_sn[0] |= 0x40;
```

2.9 Compressed Certificate Format Version

This is a 4-bit value that can be used to indicate a change in the format of the compressed certificate. There's only one version for now, so this is just set to `0x0`.

2.10 Reserved Byte

This byte is reserved for future use and should be set to zero.

3 X.509 Certificate

This section describes how the full X.509 certificates are formatted for the two certificates (Signer and Device) stored within the ATECC device.

3.1 X.509 Compressed Certificate Elements

The elements to recreate the Signer TBS certificate and the Device TBS are stored on the Device. The storage locations for each of the elements can be the Certificate Template, a Device Slot, or a Calculated value.

The table below shows the location of each element of the Signer Certificate.

Table 3-1. Signer Certificate Location

Element	Storage Location	Size	Transform
Serial Number[1]	Device Slot	var	None
	Calculated	var	See Section 2.8, “Serial Number Source”.
IssueDate	Compressed Cert	13	See Section 2.4, “Encoded Dates”. In ASCII YYMMDDHHMMSSZ
ExpireDate	Compressed Cert	13	See Section 2.4. In ASCII YYMMDDHHMMSSZ
Signer ID	Compressed Cert	4	In ASCII as four upper-case hex digits within the Subject field, Common Name attribute.
Public Key X	Device Slot	32	None
Public Key Y	Device Slot	32	None
authorityKeyIdentifier	Calculated	20	SHA1(04 + Issuer Public Key X&Y)
subjectKeyIdentifier	Calculated	20	SHA1(04 + Signer Public Key X&Y)
Signature R	Compressed Cert	32	None
Signature S	Compressed Cert	32	None

Note: 1. Customer chooses whether the serial number is stored or calculated

Table 3-2. Device Certificate Location

Element	Storage Location	Size	Transform
Serial Number[1]	Device Slot	var	None
	Calculated	var	See Section 2.8, “Serial Number Source”.
Signer ID	Compressed Cert	4	In ASCII as four upper-case hex digits within the Issuer field, Common Name attribute.
IssueDate	Compressed Cert	13	See Section 2.4. In ASCII YYMMDDHHMMSSZ
ExpireDate	Compressed Cert	13	See Section 2.4. In ASCII YYMMDDHHMMSSZ
Public Key X	Device Slot	32	None
Public Key Y	Device Slot	32	None
authorityKeyIdentifier	Calculated	20	SHA1(04 + Signer Public Key X&Y)
subjectKeyIdentifier	Calculated	20	SHA1(04 + Device Public Key X&Y)
Signature R	Compressed Cert	32	None
Signature S	Compressed Cert	32	None

Note: 1. Customer chooses whether the serial number is stored or calculated.

3.2 Signature Reconstruction

The X.509 signature presents a unique case as it is not possible to perform a simple “copy and paste” into the template as with the other elements. This is because the signature’s R and S components are stored as ASN.1 integers, whose format and length can change depending on the actual value. Since signature values are different for each certificate, the rules have to be followed closely.

ASN.1 Integers are stored in a big-endian signed format. ASN.1 encoding rules (see references below) require that if the upper-most nine bits are all ones or zeros, then the upper-most byte must be trimmed, reducing the encoded size of the integer.

Furthermore, since the R and S integers for a signature are unsigned, if either integer has a one as its uppermost bit, then a zero byte must be added to the pad the integer and prevent it from being interpreted as a negative number.

Example 1: ASN.1 Signature With Padded Integer

The following example shows all the elements of an ASN.1 signature with the S integer requiring padding. Below are the raw bytes from a signature. This starts from the signature offset.

```
03 48 00 30 45 02 20 37 4A DD 5A B5 7E 48 F8 EA
59 AB C6 E6 09 54 E8 46 25 8C CA 1E 63 25 F4 A4
86 55 20 B0 FA 48 AE 02 21 00 9C 92 55 1E 8B 85
5E 30 EA A0 9B C8 47 3C 79 27 A4 60 E8 16 11 93
5D 60 C2 D6 D8 34 BF 99 B5 CF
```

- **Bold** bytes are tags; *italics* bytes are lengths; underlined are the actual R and S integers.

ASN.1 data has a tag, length, value format, and tends to follow a tree structure. The above data can be broken down into the following structure:

BIT STRING (tag=03, length=48 (72 bytes), unused bits=00)

SEQUENCE (tag=30, length=45 (69 bytes))

INTEGER (tag=02, length=20 (32 bytes)) – R integer

INTEGER (tag=02, length=21 (33 bytes)) – S integer

This ASN.1 signature encodes the raw signature bytes:

```
37 4A DD 5A B5 7E 48 F8 EA 59 AB C6 E6 09 54 E8      R
46 25 8C CA 1E 63 25 F4 A4 86 55 20 B0 FA 48 AE
9C 92 55 1E 8B 85 5E 30 EA A0 9B C8 47 3C 79 27      S
A4 60 E8 16 11 93 5D 60 C2 D6 D8 34 BF 99 B5 CF
```

Notice the S integer is 33 bytes long in the ASN.1 encoding. That is because its upper-most bit is a one (`0x9C` = `0b10011100`) and the entire integer is interpreted as a negative number unless padded with a `00` byte.

Example 2: ASN.1 Signature With Trimmable Bytes

The next example illustrates when bytes need to be trimmed, since dealing only with unsigned numbers, it is only required to look at the case where nine or more `0` bits are in the uppermost position.

The following signature has too many 0 bits in the upper-most position.

```
00 55 DD 5A B5 7E 48 F8 EA 59 AB C6 E6 09 54 E8
46 25 8C CA 1E 63 25 F4 A4 86 55 20 B0 FA 48 AE     R
00 00 7F 1E 8B 85 5E 30 EA A0 9B C8 47 3C 79 27
A4 60 E8 16 11 93 5D 60 C2 D6 D8 34 BF 99 B5 CF     S
```

This would get encoded as the following ASN.1 signature:

03 *44* 00 **30** *41* **02** *1F* 55 DD 5A B5 7E 48 F8 EA 59
AB C6 E6 09 54 E8 46 25 8C CA 1E 63 25 F4 A4 86
55 20 B0 FA 48 AE **02** *1E* 7F 1E 8B 85 5E 30 EA A0
9B C8 47 3C 79 27 A4 60 E8 16 11 93 5D 60 C2 D6
D8 34 BF 99 B5 CF

- **Bold** bytes are tags; *italics* bytes are lengths; underlined are the actual R and S integers.

The above data can be broken down into the following structure:

- BIT STRING (tag=03, length=44 (68 bytes), unused bits=00)
  - SEQUENCE (tag=30, length=45 (65 bytes))
    - INTEGER (tag=02, length=1F (31 bytes)) – R integer
    - INTEGER (tag=02, length=1E (30 bytes)) – S integer

- The bit string and sequence lengths have adjusted for the different integer sizes.

Because the R integer has at least nine upper-most `0` bits (`0x00 0x55` = 0b**00000000 0**1010101), the upper byte gets trimmed. Because the S integer has at least 17 upper-most `0` bits (`0x00 0x00 0x7F` = 0b**00000000 00000000 0**11111111), the upper two bytes get trimmed.

The last piece that needs to be considered is the size of the certificate as a whole. When the signature changes size, the length field for the whole certificate needs to be adjusted as well. The length of the certificate is a 2-byte big endian number found as the third and fourth bytes of the certificate (index 2 and 3). For example, the following is the first 4 bytes of an example certificate template:

30 82 **01 B8**

The bytes in bold are the length (`0x01B8` = 440 bytes). If a signature changed from the first example above to the second example, where bytes had to be trimmed, then the ASN.1 signature shrunk in size from 74 bytes to 70 bytes. This change in four bytes would need to be reflected in the certificate size:

30 82 **01 B4**

The certificate size is now 436 bytes, reflecting the ASN.1 signature change.

3.2.1 Encoding steps

The encoding steps can be summarized as such:

1. Encode the R integer
   - If the uppermost bit is a one, pad the integer with a zero (encoded integer is now 33 bytes).
   - Otherwise, while the upper-most nine bits are zeros, trim the upper-most byte.
2. Encode the S integer
   - If the uppermost bit is a one, pad the integer with a zero (encoded integer is now 33 bytes).
   - Otherwise, while the upper-most nine bits are zeros, trim the upper-most byte.
3. Set the sequence and bit string length fields based on the encoded R and S integer sizes.
4. Adjust the certificate length field by the change in signature size.

3.2.2 References

ASN.1 Length encoding	X.690 (www.itu.int/rec/T-REC-X.690/e) Section 8.1.3
ASN.1 Integer encoding	X.690 (www.itu.int/rec/T-REC-X.690/e) Section 8.3
	X.680 (www.itu.int/rec/T-REC-X.680/e) Section 19.8 for tag value
ECDSA-Sig-Value encoding	RFC 5480 Appendix A (tools.ietf.org/html/rfc5480)
	SECG SEC1 (www.secg.org/sec1-v2.pdf)

4 Revision History

Doc Rev.	Date	Comments
8974A	11/2015	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**

© 2015 Atmel Corporation. / Rev.:Atmel-8974A-CryptoAuth-ATECC-Compressed-Certificate-Definition-ApplicationNote_112015.

Atmel®, Atmel logo and combinations thereof, Enabling Unlimited Possibilities®, and others are registered trademarks or trademarks of Atmel Corporation in U.S. and other countries. Other terms and product names may be trademarks of others.

DISCLAIMER: The information in this document is provided in connection with Atmel products. No license, express or implied, by estoppel or otherwise, to any intellectual property right is granted by this document or in connection with the sale of Atmel products. EXCEPT AS SET FORTH IN THE ATMEL TERMS AND CONDITIONS OF SALES LOCATED ON THE ATMEL WEBSITE, ATMEL ASSUMES NO LIABILITY WHATSOEVER AND DISCLAIMS ANY EXPRESS, IMPLIED OR STATUTORY WARRANTY RELATING TO ITS PRODUCTS INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL ATMEL BE LIABLE FOR ANY DIRECT, INDIRECT, CONSEQUENTIAL, PUNITIVE, SPECIAL OR INCIDENTAL DAMAGES (INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS AND PROFITS, BUSINESS INTERRUPTION, OR LOSS OF INFORMATION) ARISING OUT OF THE USE OR INABILITY TO USE THIS DOCUMENT, EVEN IF ATMEL HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. Atmel makes no representations or warranties with respect to the accuracy or completeness of the contents of this document and reserves the right to make changes to specifications and products descriptions at any time without notice. Atmel does not make any commitment to update the information contained herein. Unless specifically provided otherwise, Atmel products are not suitable for, and shall not be used in, automotive applications. Atmel products are not intended, authorized, or warranted for use as components in applications intended to support or sustain life.

SAFETY-CRITICAL, MILITARY, AND AUTOMOTIVE APPLICATIONS DISCLAIMER: Atmel products are not designed for and will not be used in connection with any applications where the failure of such products would reasonably be expected to result in significant personal injury or death ("Safety-Critical Applications") without an Atmel officer's specific written consent. Safety-Critical Applications include, without limitation, life support devices and systems, equipment or systems for the operation of nuclear facilities and weapons systems. Atmel products are not designed nor intended for use in military or aerospace applications or environments unless specifically designated by Atmel as military-grade. Atmel products are not designed nor intended for use in automotive applications unless specifically designated by Atmel as automotive-grade.