APPLICATION NOTE

Node Authentication Example Using Asymmetric PKI

ATECC508A

Introduction

The `node-auth-basic.atsln` project is an all-in-one example which demonstrates the various stages of the node authentication sequence using public key, asymmetric techniques of Atmel® CryptoAuthentication™ devices such as the Atmel ATECC508A.

The node authentication stages demonstrated are:

- Provisioning the ATECC508A with Device and Signer Certificates and Keys
- Reconstruct X.509 Certificates from Data Stored in the ATECC508A
- Chain Verify – Verify Device Certificate Linkage to the Root of Trust (RoT)
- Send Challenge to the Device
- Device Signs Challenge
- Verify Authenticity of Signed Challenge

Overview

The combined result of the chain verify stage and the device challenge/signature verify phase indicates if the node is authentic and can prove it is an original OEM device. It also shows an example of how a device may be provisioned to hold critical data for an X.509 certificate.

Prerequisites

- Software:
  - Atmel Studio 6.2 or 7.0
- Hardware:
  - Atmel SAM D21 Xplained Pro Evaluation Kit
  - Atmel AT88CK101 Development Board with Socket or Atmel CryptoAuth Xplained Pro Evaluation and Development Kit

Plug the CryptoAuthXplained Pro kit into the SAM D21 Ext1 or Ext2 header. The I^2C pins of the SAM D21 Xplained Pro kit automatically connects to the CryptoAuth Xplained Pro. Along with the firmware required in this project, the example is ready to run.

Table of Contents

1 For the Impatient – Where to Dive In?

For the impatient reader, the code that implements each of the stages kicks off in `node_auth.c`. The code uses the Atmel CryptoAuthLib library, a portable device driver to communicate with the ATECC508A. With this project example, a walk through of the entire sequence from top to bottom down to the driver level can be completed.

The HTML documentation for this example can be found in the `node-auth-basic/docs` directory. Load `index.html` with the browser to view the documentation for the node-auth-basic project.

The HTML documentation for CryptoAuthLib, the core crypto library for Atmel CryptoAuthentication devices, can be found in `node-auth-basic/src/cryptoauthlib/docs/`. Load `index.html` in the browser to begin the viewing of the API docs for CryptoAuthLib.

2 What is a Node?

“Node” in this use case refers to the device to be authenticated. It could be an accessory or even a sensor in a network.

3 What does “all-in-one” mean?

The “all-in-one” phrase implies that these stages are often not executed on the same device. For example, a node might be a 6LoWPAN device on a wireless network and the host is in a remote data-center. However, there are use cases where all runtime stages will be executed on the same host. For example, in a consumables use case such as a printer/printer-cartridge, the host in the printer would perform all stages shown here and the ATECC508A is in the cartridge which is in direct electrical contact with the host.

The “all-in-one” example is a convenient way to watch how all the roles work together in a system similar to the printer/print-cartridge use case. All-in-one minimizes hardware and is the easiest way to trace all the code paths from one tool, Atmel Studio.

The example will clearly differentiate which roles are being performed at each stage.

4 What are the roles demonstrated in this example?

The all-in-one example demonstrates the following roles:

- **Provisioner** The role that configures and programs the ATECC508A for runtime use.
- **Client** The device to be authenticated, such as an accessory.
- **Host** The device which would perform the authentication and verification steps to insure the device is authentic.

5 Stages of Authentication

5.1 Provisioning

Typically, the ATECC508A factory stage is carried out at the production facility, but is included here to demonstrate the basic process used to store certificates in the device.

5.2 Reconstruction

Reconstruction is the method used to take a small amount of data which is dynamically created as part of the certificate, stored within the ATECC508A, and reconstitute that data into a fully valid X.509 valid certificate.

5.3 Chain Verify, Root of Trust

Elliptic Curve Digital Signature Algorithm (ECDSA) verifies the Root of Trust (RoT) is one phase of a full verification process which insures that this device has been properly signed into the manufacturer's chain of certificates. This chain will fail if any certificate were invalid or contained an incorrect signer's signature or public key.

This verification process guards against an attacker forging a certificate within the chain to the RoT.

5.4 Signature Verify of Challenge

ECDSA verify of the signed challenge involves the host sending a challenge, a random number, to the ATECC508A to be cryptographically signed. A signature incorporates the private key held securely by the ATECC508A and the private key cannot be read from the hardware. The signature of the challenge is then verified using the public key of the device, the signature, and the challenge data itself. Once all verifications are complete, the device is determined to be authentic or not authentic, and the host can take an appropriate course based on that result.

Signing the random challenge from the host proves that the device really does own the private key associated with the public key of its certificate.

5.5 Building the Example Source Code

If using Atmel Studio 6.2, load the project file: `node-auth-basic_6_2.atsln`

If using Atmel Studio 7.0, load the project file: `node-auth-basic.atsln`

Once the project has been loaded, build it with ***Rebuild Solution*** under the ***Build*** menu. Flash the SAM D21 Xplained Pro kit using the standard Atmel Studio device programming tools.

5.6 Using the Node Authentication Basic Example

There are two USB ports on the SAM D21 Xplained Pro. One is labeled, “EDBG USB” and is used to flash the code into the MCU with Atmel Studio. The second USB port is labeled “Target USB”, a CDC USB port and is used for the console interface to the example.

1. Connect the host computer to the EDBG USB to program it.
2. Connect the host computer to the Target USB CDC port in order to see a console interface you can use to exercise the example after it has been programmed.
3. Use a terminal program on the host and connect it to the virtual COMM port of the SAM D21 Xplained Pro which should be created when the Target USB CDC port is plugged into your PC, Linux, or OS X machine. This particular step will vary on each computer and operating system.

The communication parameters are:

- 115,200 baud
- 8 bit word
- No parity
- 1 stop bit

Help Command

Once connected to the serial USB, type `help` and the command line console will list as follows:

```
$ help
Usage:
client-provision  - Configure and load certificate data onto ATECC device.
client-build      - Read certificate data off ATECC device and rebuild full
signer and device certificates.
host-chain-verify - Verify the certificate chain from the client.
host-gen-chal     - Generate challenge for the client.
client-gen-resp   - Generate response to challenge from host.
host-verify-resp  - Verify the client response to the challenge.
Utility functions:
lockstat - zone lock status
lockcfg  - lock config zone
lockdata - lock data and OTP zones
info     - get the chip revision
sernum   - get the chip serial number

$
```

Check ATECC508A Connectivity

Using the command console, use the `info` or `sernum` command to display the device's revision and serial numbers. These are both good tests to insure your board and ATECC508A can communicate with each other.

Below is an example session of what to expect. Your serial number will be different, of course.

```
$ info
revision:
00 00 50 00
$ sernum
serial number:
01 23 61 12 D9 2C A5 71 EE
$
```

You must be able to perform this step successfully before proceeding to the next steps.

If you do not see identical revision or similar serial number, then check your connections to the CryptoAuthXplained Pro extension board or the socketed top-board connected to the I^2C pins of the SAM D21 Xplained Pro kit.

Step 1 Provision the ATECC508A: `client-provision`

Type the command: `client-provision`

This is a one-time step which generates the keys in the ATECC508A, as well as construct the certificates required later to complete the verification steps. The certificates created and stored in this step are the device's certificate and the signer's certificate.

Once the command is complete, all the certificates and keys will be stored and locked in the device. The device cannot be changed thereafter.

Example: `client-provision` Session

An example session with `client-provision` might look similar to this. Don't worry about the exact bytes shown, yours will be different; the main point is that you see the various components have been created and have data.

```
Signer CA Public Key:
02 54 9E 50 2F 7C 13 1E C5 DA 7A 8B BF 5E 0D 05
E1 3D 8E 11 F4 F1 04 D2 F6 CE 41 44 FA 40 E6 D4
02 3C A0 80 30 B1 DE F1 4A A7 CE A3 FF 12 4B 4B
A5 91 E0 F1 59 EF 67 A9 68 E5 CC 5C 0B FD E8 7A
Signer Public Key:
A3 AC C0 2F 35 17 15 08 68 B1 10 43 24 F9 EA 30
17 2C B1 11 AB A1 F0 B5 0B 4B 85 77 2B F3 14 08
70 C0 69 8E AF AA 6A 58 F9 8E 22 0F 3A 9E F8 35
C0 6A 5D FB C5 25 F4 56 5A A7 AB A9 E9 B1 44 E6
Device Public Key:
B9 17 F9 9F BA A0 AF 3C 67 61 B8 DB D8 2F 8E 6B
C1 CB D0 CF 87 82 08 0E 2B D3 EC EF E8 E9 C5 3B
E2 1C 2E 5D CC A1 92 A5 A1 22 68 EA FF 94 68 F5
C0 54 DD 32 40 F9 F6 C2 9B AF 0D 46 36 EC 5F 26
Signer Certificate:
30 82 01 B1 30 82 01 57 A0 03 02 01 02 02 03 40
```

```
C4 8B 30 0A 06 08 2A 86 48 CE 3D 04 03 02 30 36
31 10 30 0E 06 03 55 04 0A 0C 07 45 78 61 6D 70
6C 65 31 22 30 20 06 03 55 04 03 0C 19 45 78 61
6D 70 6C 65 20 41 54 45 43 43 35 30 38 41 20 52
6F 6F 74 20 43 41 30 1E 17 0D 31 34 30 38 30 32
32 30 30 30 30 30 5A 17 0D 33 34 30 38 30 32 32
30 30 30 30 30 5A 30 3A 31 10 30 0E 06 03 55 04
0A 0C 07 45 78 61 6D 70 6C 65 31 26 30 24 06 03
55 04 03 0C 1D 45 78 61 6D 70 6C 65 20 41 54 45
43 43 35 30 38 41 20 53 69 67 6E 65 72 20 43 34
38 42 30 59 30 13 06 07 2A 86 48 CE 3D 02 01 06
08 2A 86 48 CE 3D 03 01 07 03 42 00 04 A3 AC C0
2F 35 17 15 08 68 B1 10 43 24 F9 EA 30 17 2C B1
11 AB A1 F0 B5 0B 4B 85 77 2B F3 14 08 70 C0 69
8E AF AA 6A 58 F9 8E 22 0F 3A 9E F8 35 C0 6A 5D
FB C5 25 F4 56 5A A7 AB A9 E9 B1 44 E6 A3 50 30
4E 30 0C 06 03 55 1D 13 04 05 30 03 01 01 FF 30
1D 06 03 55 1D 0E 04 16 04 14 BB 5C 3D F7 4D 4C
93 D4 2B 50 D1 7F B3 23 C3 3A B0 2C 27 BA 30 1F
06 03 55 1D 23 04 18 30 16 80 14 14 B0 97 8A 1D
57 50 FF 52 F9 DF A8 90 60 77 60 C5 3C 6B 50 30
0A 06 08 2A 86 48 CE 3D 04 03 02 03 48 00 30 45
02 21 00 FB 08 10 99 B3 F0 A8 E5 D5 19 3F 1A A2
20 94 06 A1 63 D9 4A CE 18 6A 80 C6 6A E7 91 42
6C 58 7D 02 20 46 85 5F 9D 71 F2 B9 48 84 75 2E
49 2F D7 58 AD 1B EB BD 36 A5 74 64 2B 6B EA 02
26 5A 72 13 3F
Device Certificate:
30 82 01 8A 30 82 01 30 A0 03 02 01 02 02 0A 40
01 23 6F 12 D9 2C A5 71 EE 30 0A 06 08 2A 86 48
CE 3D 04 03 02 30 3A 31 10 30 0E 06 03 55 04 0A
0C 07 45 78 61 6D 70 6C 65 31 26 30 24 06 03 55
04 03 0C 1D 45 78 61 6D 70 6C 65 20 41 54 45 43
43 35 30 38 41 20 53 69 67 6E 65 72 20 43 34 38
42 30 1E 17 0D 31 35 30 39 30 33 32 31 30 30 30
30 5A 17 0D 33 35 30 39 30 33 32 31 30 30 30 30
5A 30 35 31 10 30 0E 06 03 55 04 0A 0C 07 45 78
61 6D 70 6C 65 31 21 30 1F 06 03 55 04 03 0C 18
45 78 61 6D 70 6C 65 20 41 54 45 43 43 35 30 38
41 20 44 65 76 69 63 65 30 59 30 13 06 07 2A 86
48 CE 3D 02 01 06 08 2A 86 48 CE 3D 03 01 07 03
42 00 04 B9 17 F9 9F BA A0 AF 3C 67 61 B8 DB D8
2F 8E 6B C1 CB D0 CF 87 82 08 0E 2B D3 EC EF E8
E9 C5 3B E2 1C 2E 5D CC A1 92 A5 A1 22 68 EA FF
94 68 F5 C0 54 DD 32 40 F9 F6 C2 9B AF 0D 46 36
EC 5F 26 A3 23 30 21 30 1F 06 03 55 1D 23 04 18
30 16 80 14 BB 5C 3D F7 4D 4C 93 D4 2B 50 D1 7F
B3 23 C3 3A B0 2C 27 BA 30 0A 06 08 2A 86 48 CE
3D 04 03 02 03 48 00 30 45 02 20 35 96 2E 3F F4
1A 3A DA E7 6F E1 FE 9D 7A 83 BE 36 FA 06 C5 01
79 55 F2 2C 8C FE 1D 43 38 19 CC 02 21 00 E8 53
87 83 A6 98 21 8E 43 A0 08 73 B3 FD B4 4B 7E 1C
EC FB 61 33 52 59 99 DF B1 E1 79 3E D7 8B
$
```

Step 2 Read ATECC508A Certificates: `client-build`

Type the command: `client-build`

`client-build` reads the certificate data from the ATECC508A and reconstructs them into X.509 DER format certificates. For this demonstration, you won't have to parse the full certificate; the demo code will use the X.509 DER formats for its verification and validation steps.

Example: A typical `client-build` session would look like this.

Optionally, compare this output to the certificates shown during the client provisioning step. They should be the same. In this step, the certificate data was read from the device and reconstructed to match what was intended when the part was provisioned.

```
1  |   CLIENT: Rebuilt Signer Certificate:
2  |   30 82 01 B1 30 82 01 57 A0 03 02 01 02 02 03 40
3  |   C4 8B 30 0A 06 08 2A 86 48 CE 3D 04 03 02 30 36
4  |   31 10 30 0E 06 03 55 04 0A 0C 07 45 78 61 6D 70
5  |   6C 65 31 22 30 20 06 03 55 04 03 0C 19 45 78 61
6  |   6D 70 6C 65 20 41 54 45 43 43 35 30 38 41 20 52
7  |   6F 6F 74 20 43 41 30 1E 17 0D 31 34 30 38 30 32
8  |   32 30 30 30 30 30 5A 17 0D 33 34 30 38 30 32 32
9  |   30 30 30 30 30 5A 30 3A 31 10 30 0E 06 03 55 04
10 |  0A 0C 07 45 78 61 6D 70 6C 65 31 26 30 24 06 03
11 |  55 04 03 0C 1D 45 78 61 6D 70 6C 65 20 41 54 45
12 |  43 43 35 30 38 41 20 53 69 67 6E 65 72 20 43 34
13 |  38 42 30 59 30 13 06 07 2A 86 48 CE 3D 02 01 06
14 |  08 2A 86 48 CE 3D 03 01 07 03 42 00 04 A3 AC C0
15 |  2F 35 17 15 08 68 B1 10 43 24 F9 EA 30 17 2C B1
16 |  11 AB A1 F0 B5 0B 4B 85 77 2B F3 14 08 70 C0 69
17 |  8E AF AA 6A 58 F9 8E 22 0F 3A 9E F8 35 C0 6A 5D
18 |  FB C5 25 F4 56 5A A7 AB A9 E9 B1 44 E6 A3 50 30
19 |  4E 30 0C 06 03 55 1D 13 04 05 30 03 01 01 FF 30
20 |  1D 06 03 55 1D 0E 04 16 04 14 BB 5C 3D F7 4D 4C
21 |  93 D4 2B 50 D1 7F B3 23 C3 3A B0 2C 27 BA 30 1F
22 |  06 03 55 1D 23 04 18 30 16 80 14 14 B0 97 8A 1D
23 |  57 50 FF 52 F9 DF A8 90 60 77 60 C5 3C 6B 50 30
24 |  0A 06 08 2A 86 48 CE 3D 04 03 02 03 48 00 30 45
25 |  02 21 00 FB 08 10 99 B3 F0 A8 E5 D5 19 3F 1A A2
26 |  20 94 06 A1 63 D9 4A CE 18 6A 80 C6 6A E7 91 42
27 |  6C 58 7D 02 20 46 85 5F 9D 71 F2 B9 48 84 75 2E
28 |  49 2F D7 58 AD 1B EB BD 36 A5 74 64 2B 6B EA 02
29 |  26 5A 72 13 3F
30 |  CLIENT: Rebuilt Device Certificate:
31 |  30 82 01 8A 30 82 01 30 A0 03 02 01 02 02 0A 40
32 |  01 23 6F 12 D9 2C A5 71 EE 30 0A 06 08 2A 86 48
33 |  CE 3D 04 03 02 30 3A 31 10 30 0E 06 03 55 04 0A
34 |  0C 07 45 78 61 6D 70 6C 65 31 26 30 24 06 03 55
35 |  04 03 0C 1D 45 78 61 6D 70 6C 65 20 41 54 45 43
36 |  43 35 30 38 41 20 53 69 67 6E 65 72 20 43 34 38
37 |  42 30 1E 17 0D 31 35 30 39 30 33 32 31 30 30 30
38 |  30 5A 17 0D 33 35 30 39 30 33 32 31 30 30 30 30
39 |  5A 30 35 31 10 30 0E 06 03 55 04 0A 0C 07 45 78
40 |  61 6D 70 6C 65 31 21 30 1F 06 03 55 04 03 0C 18
41 |  45 78 61 6D 70 6C 65 20 41 54 45 43 43 35 30 38
```

```
41 20 44 65 76 69 63 65 30 59 30 13 06 07 2A 86
48 CE 3D 02 01 06 08 2A 86 48 CE 3D 03 01 07 03
42 00 04 B9 17 F9 9F BA A0 AF 3C 67 61 B8 DB D8
2F 8E 6B C1 CB D0 CF 87 82 08 0E 2B D3 EC EF E8
E9 C5 3B E2 1C 2E 5D CC A1 92 A5 A1 22 68 EA FF
94 68 F5 C0 54 DD 32 40 F9 F6 C2 9B AF 0D 46 36
EC 5F 26 A3 23 30 21 30 1F 06 03 55 1D 23 04 18
30 16 80 14 BB 5C 3D F7 4D 4C 93 D4 2B 50 D1 7F
B3 23 C3 3A B0 2C 27 BA 30 0A 06 08 2A 86 48 CE
3D 04 03 02 03 48 00 30 45 02 20 35 96 2E 3F F4
1A 3A DA E7 6F E1 FE 9D 7A 83 BE 36 FA 06 C5 01
79 55 F2 2C 8C FE 1D 43 38 19 CC 02 21 00 E8 53
87 83 A6 98 21 8E 43 A0 08 73 B3 FD B4 4B 7E 1C
EC FB 61 33 52 59 99 DF B1 E1 79 3E D7 8B
$
```

Step 3 Verify the Certificate Chain: `host-chain-verify`

Type the command: `host-chain-verify`

`host-chain-verify` retrieves the device certificate and the signer certificate from the ATECC508A, reconstructs the certificates, and then performs a chain verify which verifies that the device certificate is valid and has been signed into the chain leading to a RoT.

Example: A typical `host-chain-verify` session will look like this:

```
$ host-chain-verify
HOST: Signer certificate verified against signer certificate authority (CA) public key!
HOST: Device certificate verified against signer public key!
```

Step 4 Generate a challenge from the Host: `host-gen-chal`

Type the command: `host-gen-chal`

`host-gen-chal` generates a random challenge and asks the ATECC508A to sign it using the private key stored in the ATECC508A corresponding to the device certificate.

This is one half of the typical “challenge/response” pattern. After the response has been received (Step 5), an ECDSA verification can be performed which does the math to determine if the signature was valid.

Example: A challenge will look similar to this:

```
$ host-gen-chal
HOST: Generated challenge:
14 84 E8 89 41 D5 9A 1C AD 1F 68 44 3A 09 C6 45
30 BF 27 38 D2 28 56 B7 DD D6 98 CF 92 AB 3D 69
```

Step 5 Generate Response to Challenge (Signature): `client-gen-resp`

Type the command: `client-gen-resp`

`client-gen-resp` generates the signature of the challenge performed in Step 4. It requests the ATECC508A sign the challenge and return the signature it generated. This signature is used in the next verification steps.

Example: The generation of the signature will look similar to this:

```
$ client-gen-resp
CLIENT: Calculated response to host challenge:
BB BD 18 73 C3 88 86 E7 86 4A 53 CF 8F 18 4D EC
1A 39 A2 B9 FC 0B FE 73 CE 51 42 0C FB 81 26 F9
63 C1 A0 AF A8 67 58 FB 3B 9D 19 6B FE 86 98 47
0C 13 C9 95 8D 37 C9 47 57 61 A0 F7 D4 52 42 45
```

Step 6 Verify the Signature: `host-verify-resp`

Type the command: `host-verify-resp`

`host-verify-resp` performs an ECDSA verification to determine if the signature is valid. ECDSA verification requires three pieces of data:

- Public key of the device.
- Challenge given to the device to sign.
- Signature of the challenge.

If the ECDSA verification step verifies the device, the device has proven that it has the private key associated with the public key that is in its device certificate and signed into the certificate chain. That's a long way of saying that it has proven that it owns the public key and if its certificate with the same public key passes the chain verification, then the device is considered fully verified and an authentic OEM device.

Example: Final device verification step:

```
$ host-verify-resp
CLIENT: Calculated response to host challenge:
BB BD 18 73 C3 88 86 E7 86 4A 53 CF 8F 18 4D EC
1A 39 A2 B9 FC 0B FE 73 CE 51 42 0C FB 81 26 F9
63 C1 A0 AF A8 67 58 FB 3B 9D 19 6B FE 86 98 47
0C 13 C9 95 8D 37 C9 47 57 61 A0 F7 D4 52 42 45
HOST: Device public key from certificate:
B9 17 F9 9F BA A0 AF 3C 67 61 B8 DB D8 2F 8E 6B
C1 CB D0 CF 87 82 08 0E 2B D3 EC EF E8 E9 C5 3B
E2 1C 2E 5D CC A1 92 A5 A1 22 68 EA FF 94 68 F5
C0 54 DD 32 40 F9 F6 C2 9B AF 0D 46 36 EC 5F 26
HOST: Device response to challenge verified!
```

6 Revision History

Doc Rev.	Date	Comments
8983A	09/2015	Initial document release.

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