

Production Provisioning Guide

Atmel ECC-based CryptoAuthentication Device Family

Introduction

The Atmel® Security Provisioning Kits provide an easy and secure method to create certificate authorities for provisioning Atmel Elliptic Curve Cryptography (ECC) based CryptoAuthentication™ devices. This document describes the software and steps required to integrate ECC-based device provisioning into a production environment.

Features

- Production Flow
- Provisioning Server and Signer Modules
- Provisioning Client
- Provisioning Library

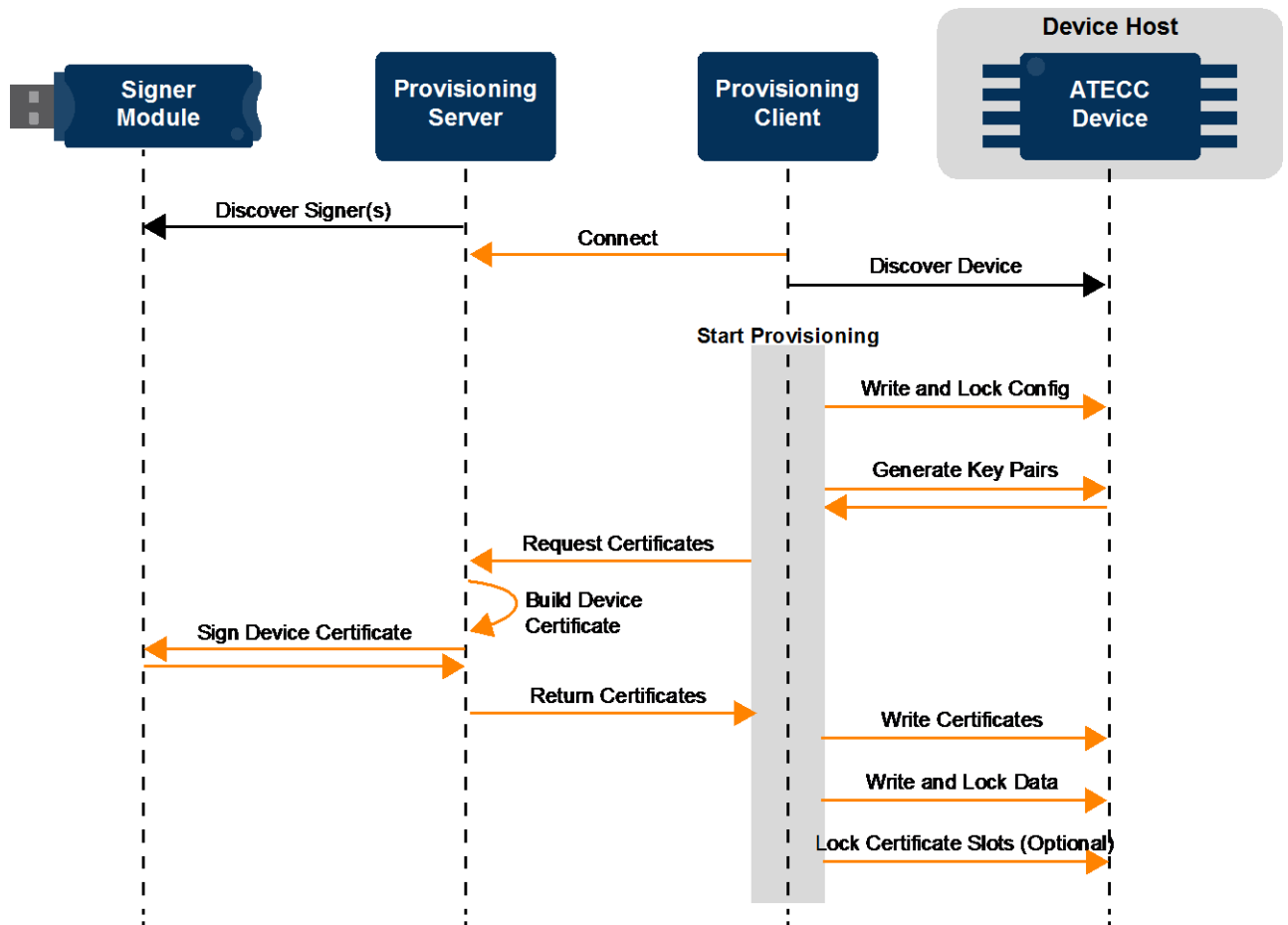
Table of Contents

1	Production Flow	3
2	Provisioning Server and Signer Modules	5
2.1	Running the Server	5
2.2	Signer Modules	5
2.2.1	Signer Module Passwords	5
2.3	Logs	6
3	Provisioning Client.....	7
3.1	Provisioning Library.....	8
3.2	CryptoAuthLib Library.....	8
3.3	Provisioning Client CDC Example	8
3.3.1	Usage Flow.....	9
3.4	Program Options	11
3.5	Config File Format.....	11
4	Provisioning Functionality.....	12
5	Provisioning Library.....	15
5.1	Provisioning Stages.....	15
5.2	Prerequisites	16
5.2.1	CryptoAuthLib HAL Driver	16
5.2.2	Initialize the CryptoAuthLib Basic API	16
5.3	Provisioning Server Communication	17
5.3.1	atcaprov_certreq_start()	17
5.3.2	atcaprov_certreq_send()	17
5.3.3	atcaprov_certreq_recv()	17
5.3.4	atcaprov_certreq_done()	17
5.3.5	atcaprov_request_error()	17
5.4	Writing Non-Certificate Data.....	17
5.5	Progress Notification	18
6	Revision History	18

1 Production Flow

This production flow demonstrates the steps required to provision an Atmel ECC-based CryptoAuthentication device within a user's product. Typically, this process is setup to run during the manufacturing process of the user's product.

Figure 1-1. Provisioning Within a User's Product Production Flow



Step 1 Provisioning Server

- Discovers available signer modules.
- Then waits for clients to connect and request certificates.

Step 2 Start Provisioning

The provisioning process is ultimately controlled by the Provisioning Client, which is responsible for interacting with the actual ATECC device to be provisioned.

- The client first connects to the Provisioning Server.
- Then the client waits for an ATECC device to become available.
- The Provisioning Client uses the provisioning library to perform the actual provisioning of the ATECC device.

The table below gives a brief summary of each of the components shown above:

Table 1-1. Production Component Descriptions

Component	Description
Signer Module	Configured signer module from the Atmel AT88CKECCSIGNER Module Kit. This USB module must be plugged into the same computer the Provisioning Server is running on. It has already configured using the Atmel Signer Module Utility. The Signer Module is the device Certificate Authority (CA) and securely stores and uses the signer's private key to sign device certificates.
Provisioning Server	The application responsible for managing configured signer modules, receiving certificate requests, and returning signed certificates to be written into an ATECC device.
Provisioning Client	The user's application that controls the provisioning process. The Provisioning Client uses the provisioning library and the Atmel CryptoAuthLib library to interact with the ATECC device being provisioned and the Provisioning Server. The development of this application is the responsibility of the user, as it's dependent on the particulars of the user's production setup. See Section 3.3, "Provisioning Client CDC Example" for application source code to guide the development of a user's own client application.
CryptoAuth Device	The ATECC device that is to be provisioned. Since this uses I ² C or Single-Wire Interfaces, a device host of some sort (e.g. USB-I ² C adapter, user's product microcontroller, etc...) is required to enable communication with the Provisioning Client.

2 Provisioning Server and Signer Modules

The Provisioning Server application is responsible for managing signer modules and using them to respond to certificate requests from one or more Provisioning Clients.

The Provisioning Server application is a TCP/IP server and can be run on the same computer as the Provisioning Client or located on a separate computer as long as the client can connect to the server over a network.

2.1 Running the Server

The Provisioning Server can be started from the start menu:

Start Menu > All Programs > Atmel Secure Products > Provisioning Server > Provisioning Server



With a TCP/IP server, firewall applications may need to be configured to allow connections to the server on its default port of **11235**.

The application is installed in one of the following locations depending on operating system:

- C:\Program Files\Atmel\Atmel Security Products\Provisioning Server\ProvisioningServer.exe
- C:\Program Files (x86)\Atmel\Atmel Security Products\Provisioning Server\ProvisioningServer.exe

2.2 Signer Modules

While running, the Provisioning Server periodically scans for added or removed signer modules attached to the system. For that reason, the Atmel Root Module and Signer Module Utility shouldn't be run on the same computer while the Provisioning Server is running.

The Provisioning Server can handle multiple signer modules. The certificate requests received by the Provisioning Server contain the information required to select the right one to use for that request.

2.2.1 Signer Module Passwords

If a signer module has been configured with a password, the Provisioning Server prompts for that password. An example prompt is shown below.

Example: Provisioning Server Password Prompt

```
Press ESC to quit.
[2015-11-06 15:51:07, info] Provisioning Server started.
[2015-11-06 15:51:07, info] Signer Manager started.
[2015-11-06 15:51:07, info] Server starting...
[2015-11-06 15:51:07, info] Server listening on 0.0.0.0:11235.
[2015-11-06 15:51:15, info] Signer (01231E6CD82CA571EE, Example ATECC508A Signer)
requires password.
Signer (01231E6CD82CA571EE, Example ATECC508A Signer) password:
```

Once the correct password has been entered, that signer module is available for certificate requests.

```
[2015-11-06 15:52:36, info] Signer (01231E6CD82CA571EE, Example ATECC508A Signer)
password accepted, ready.
```



The password is good for as long as the Provisioning Server is running and the Signer Module is plugged in. If the Provisioning Server is restarted or the Signer Module is unplugged and re-plugged in, then the password will have to be re-entered.

2.3 Logs

The Provisioning Server saves a number of logs that may be useful. These logs are stored in one of the following locations depending on operating system:

- C:\Documents and Settings\All Users\Application Data\Atmel\ProvisioningServer
- C:\ProgramData\Atmel\ProvisioningServer

Within these folders are two sub-folders, certs and logs.

- **logs** – The logs folder has program activity and signer logs. A new log file is created each time the Provisioning Server is started. The signing log file is in a tab-separated format and can be easily imported into a spreadsheet application such as Microsoft Excel or a database if need be.
- **certs** – The certs folder contains the full signer and device certificates for each device the server has provisioned.



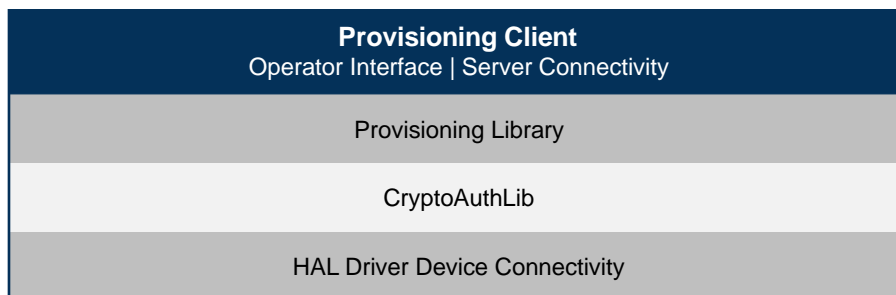
The ProgramData folder is often hidden by default on windows. To make hidden files display as default in Microsoft Windows 10:

1. Click the **Start** button > **Control Panel** > **Appearance and Personalization** > then **Folder Options**.
2. Click the **View** tab and under **Advanced settings** select **Show hidden files, folders, and drives** > then click **OK** to make the ProgramData folder appear by default.

3 Provisioning Client

The Provisioning Client is the application that controls the entire provisioning process. Since each user's production processes can be unique, the final implementation of the Provisioning Client is dependent on the user to adapt to their systems. The CryptoAuthLib library, provisioning library, and an example application are provided to facilitate the user's development of their own application.

Figure 3-1. Provisioning Client Library Layers



The above diagram illustrates the dependencies between the Provisioning Client and the provided libraries. The Provisioning Client directly talks with the provisioning library, which in turn, uses CryptoAuthLib to facilitate communication with the ATECC device through the HAL (hardware abstraction layer) driver.

The Provisioning Client may interact with CryptoAuthLib or the HAL driver directly to enable hardware-specific functionality such as discovering when a device is available to provision.

Outside of the provided libraries, the user may be expected to develop the operator interface, Provisioning Server connectivity, and ATECC device connectivity (as a CryptoAuthLib HAL driver) for the Provisioning Client. More details can be found in Section 3.3, Provisioning Client CDC Example, which discusses a complete example of a Provisioning Client.

Table 3-1. Provisioning Client Responsibilities

Component	Description
Operator Interface	If the Provisioning Client is a standalone application, then a user interface for the operator needs to be developed. The Provisioning Client can also be integrated as part of a larger application responsible for the programming of a user's product as a whole. In that case, the interface and functionality can be rolled into that larger application.
Provisioning Server Connectivity	The Provisioning Client needs to provide the actual server connectivity, which is most often accomplished as a simple TCP/IP client. The actual messages to and from the server are handled by the provisioning library, which will notify the application when it needs to send or receive data.
ATECC Device Connectivity	Device connectivity refers the actual communication between the Provisioning Client and the ATECC device being provisioned. Communication may need to go through one or more hardware interfaces before it actually reaches the device itself. That hardware may be as simple as a USB I ² C adapter for talking with the device directly, or a custom interface designed for the product the ATECC device is on. This connectivity is implemented as Hardware Abstraction Layer (HAL) driver for CryptoAuthLib. See Section 3.2, CryptoAuthLib Library for more detail.

3.1 Provisioning Library

The Provisioning Library provides the process framework for provisioning an ATECC device. It communicates with the ATECC device through CryptoAuthLib and performs the following steps:

1. Write and lock the ATECC device Configuration zone.
2. Generate the ATECC device's key pair.
 - Please note the private key is securely generated within the ATECC device and cannot be read out.
 - The public key is required for generating the device's certificate.
3. Request a device and signer certificate from the Provisioning Server.
 - The provisioning library constructs and interprets the messages to and from the Provisioning Server. However, since the Provisioning Client is responsible for the actual server connectivity, the library informs the Provisioning Client when it needs to send or receive data.
4. Write the device and signer certificate data to the ATECC device.
5. Give the Provisioning Client an opportunity to write any additional data to the ATECC device.
6. Lock the data zone.
7. Optionally, lock individual slots.
8. Validate certificates.

The library is provided as C source code and is easy to modify should the user require features outside of the default provisioning flow.

See Section 5, Provisioning Library for information on how to use the Provisioning Library.

3.2 CryptoAuthLib Library

The Atmel CryptoAuthLib is a software support library for the Atmel ATSHA204A, ATECC108A and ATECC508A CryptoAuthentication devices written in C. It is a portable, extensible, powerful, and easy-to-use library for working with the ATSHA and ATECC CryptoAuthentication family of devices.

CryptoAuthLib can be downloaded from the Atmel website at: www.atmel.com/tools/CryptoAuthLib.aspx

CryptoAuthLib provides a hardware abstraction layer (HAL) for talking to a CryptoAuthentication device regardless of the hardware interface. The actual implementation of the low-level communication with the ATECC device is through a HAL driver. CryptoAuthLib supplies some basic drivers; however, the user may need to develop a custom driver to work with their particular host hardware. The Atmel CryptoAuthLib application note, “[CryptoAuthLib Driver Support for Atmel CryptoAuthentication Devices](#)” details how to develop a custom HAL driver.

3.3 Provisioning Client CDC Example

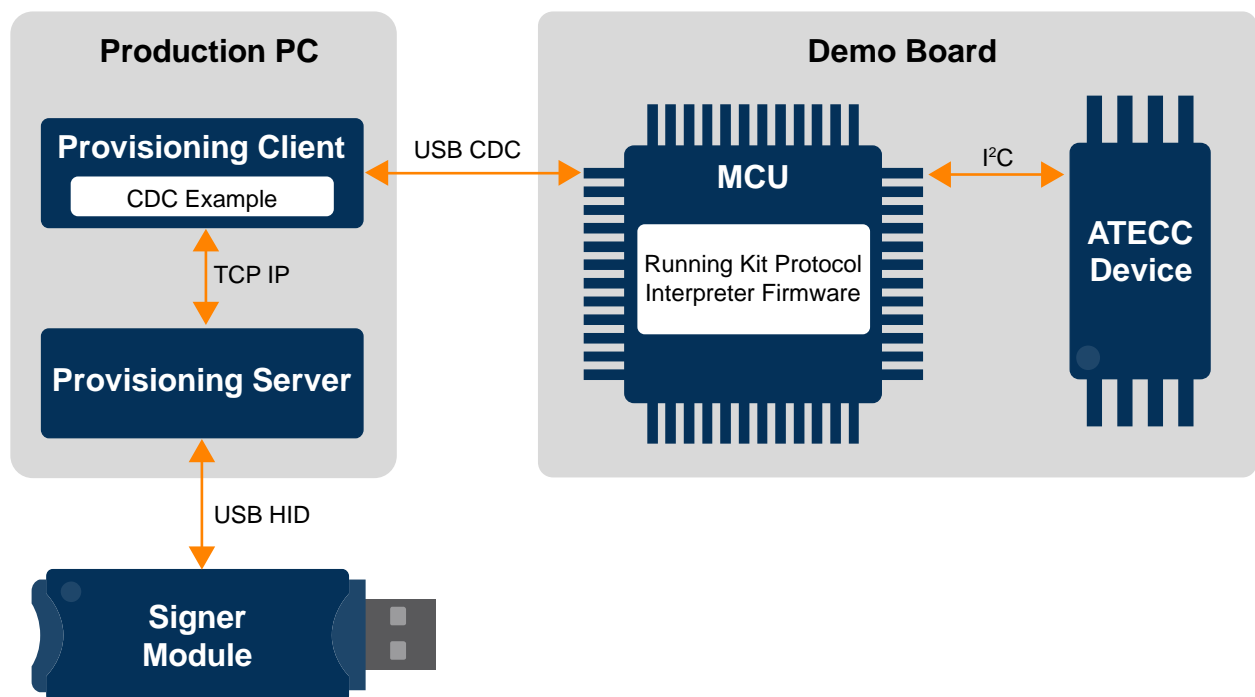
This application is a complete example of a Provisioning Client for the user to work with as is, as a reference, or to modify for their own production setup.

The Provisioning Client CDC Example has the following configuration:

- Provided as a Visual Studio 2013 C++ project with all source code.
- Operator interface is a simple console-mode (text only) style.
- Provisioning Server connectivity is implemented using the Boost asio library for networking.
 - Tested with version 1.59.0
 - Please reference www.boost.org

- Device connectivity is through the USB CDC Kit Protocol CryptoAuthLib HAL (hardware abstraction layer) driver for windows.
 - This driver allows the application to discover boards running the Atmel Kit Protocol interpreter firmware.
 - Kit Protocol is used by Atmel’s security demo kits for communication to a PC.
- Tested with the SAM D21 Xplained Pro Evaluation Kit (www.atmel.com/tools/ATSAMD21-XPRO.aspx) with an Atmel ATECC508A device on its 2-Wire Interface/I²C bus. The Atmel CryptoAuth Xplained Pro Evaluation and Development Kit (www.atmel.com/tools/CryptoAuthXplainedPro.aspx) is an easy way to attach an ATECC508A device.
 - The Kit Protocol Interpreter firmware for the SAM D21 Xplained Pro is included as an Atmel Studio 7.0 project.

Figure 3-2. Provisioning Client CDC Example Setup



3.3.1 Usage Flow

Below describes how the Provisioning Client CDC Example works from startup through actual use.

Step 1 On startup, the client looks for and reads the `atcaprov_cert_info_0.c` file.

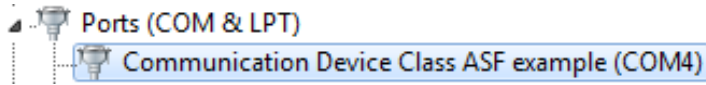
This file is generated by the Signer Module Utility when configuring a Signer Module. It's not compiled in this case, but just read. It contains information that specifies what kind of signer is required when requesting certificates from the Provisioning Server.

Step 2 Connects to the Provisioning Server.

Unless otherwise specified, it assumes the server is running on the same computer as the client application.

Step 3 The client waits for a compatible board to be plugged into the production PC.

The program attempts to communicate with any device that shows up with an Atmel Demo Kit description, such as “Communication Device Class ASF Example”. Below is how the device may appear in Device Manager.



Once found, it attempts to communicate using Kit Protocol and discover any ATECC devices onboard.

Step 4 Once an ATECC device has been found, it notifies the operator and waits for the operator to press *Enter* to start the provisioning process.

Step 5 The Provisioning Client provisions the ATECC device using the provisioning library.

Step 6 The Provisioning Client prompts the operator to remove the board with the provisioned ATECC device and connect the next board to be provisioned.

Example: Provisioning Client CDC Example Output

```
① --Reading provisioning config file atcaprov_cert_info_0.c...ok
② --Connecting to localhost:11235...ok
   Press ESC to quit.
③ --Waiting for board to be connected...ok
   Device SN: 01234038D92CA571EE
④ -----Press ENTER to provision...
   Writing config zone.....done
   Locking config zone.....done
   Reading config zone.....done
   Generating device key.....done
   Requesting certificates...done
   Writing certificates.....done
   Writing data zone.....done
   Locking data zone.....done
   Locking slots.....skipped
   Validating certificates...done
⑤ --Provisioning successful! Please remove board...
⑥ --Board removed. Waiting for board to be connected...ok
   Device SN: 01233418D92CA571EE
   Press ENTER to provision...
   Writing config zone.....done
   ⋮
```

3.4 Program Options

If the ProvisioningClientCDCEExample application is run without any arguments, it makes some default assumptions about the Provisioning Server and configuration file. However, these options can be changed. Running the application with the `--help` argument displays the following help.

`--help` Argument

```
>ProvisioningClientCDCEExample --help
Options:
  --help                Show help.
  --certinfo filename (=atcaprov_cert_info_0.c)
                        Path and file name for the provisioning
                        configuration file. This is the
                        atcaprov_cert_info_X.c file output by the Signing
                        Module Utility when configuring a signer module.
                        It contains information needed when requesting
                        certificates from the Provisioning Server.
  --config filename     Path and file name for the ATECC508A device
                        configuration zone data. If omitted, a default
                        configuration will be used.
  --host name/ip (=localhost) Provisioning server host name or IP address. If
                        omitted, localhost is used, which assumes the
                        Provisioning Server is running on the same
                        computer.
  --port num (=11235)   Provisioning server port number. If omitted,
                        11235 is used.
```

3.5 Config File Format

If one uses the `--config` option to specify a new device configuration file, the file should be a text file with ascii hex representing the 128 bytes of the Configuration zone.

Example: Default ATECC508A Configuration

```
00 00 00 00 00 00 00 00 00 00 00 00 00 01 00
C0 00 AA 00 8F 20 C4 44 87 20 C4 44 8F 0F 8F 8F
9F 8F 82 20 C4 44 C4 44 0F 0F 0F 0F 0F 0F 0F 0F
0F 0F 0F 0F FF FF FF FF 00 00 00 00 FF FF FF FF
00 00 00 00 FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF 00 00 55 55 FF FF 00 00 00 00 00 00
33 00 5C 00 13 00 5C 00 3C 00 1C 00 1C 00 33 00
1C 00 1C 00 3C 00 3C 00 3C 00 3C 00 1C 00 3C 00
```

Note: The first 16 bytes are ignored, as that part of the Configuration zone is not writable.

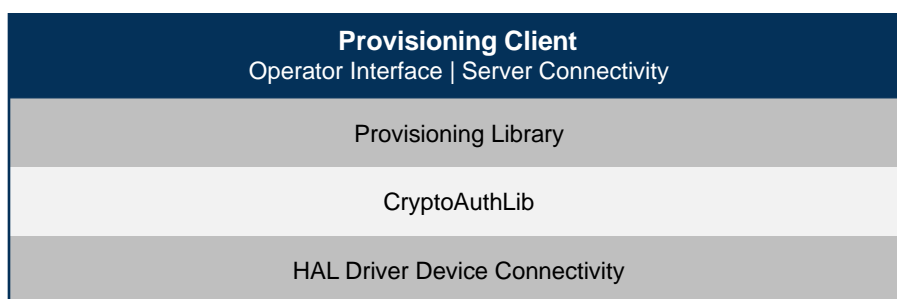
4 Provisioning Functionality

The provisioning of an ECC-based CryptoAuthentication device (e.g. ATECC508A) using a Signer Module from the AT88ECCSIGNER kit can take many forms. For example, some users may want to provision the devices before installing them onto a board or product, while others may want to provision a part already installed. The wide variety hardware platforms and production setups necessitates a custom provisioning system in many cases.

To facilitate each user's unique production setup, the provisioning software is provided as a simple C library in source form that can be adapted to a user's needs. This is intended to guide the user through the process of integrating the provisioning library into their own customized production application.

A complete application that includes the provisioning library is called the Provisioning Client.

Figure 4-1. Provisioning Client Library Layers

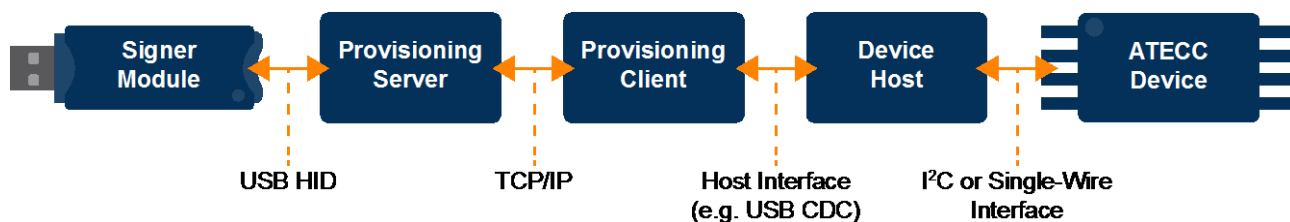


The Provisioning Library requires the Provisioning Client to implement two primary functions:

1. Connectivity to the CryptoAuthentication device to be provisioned (through a Device Host).
2. Connectivity to the Provisioning Server.

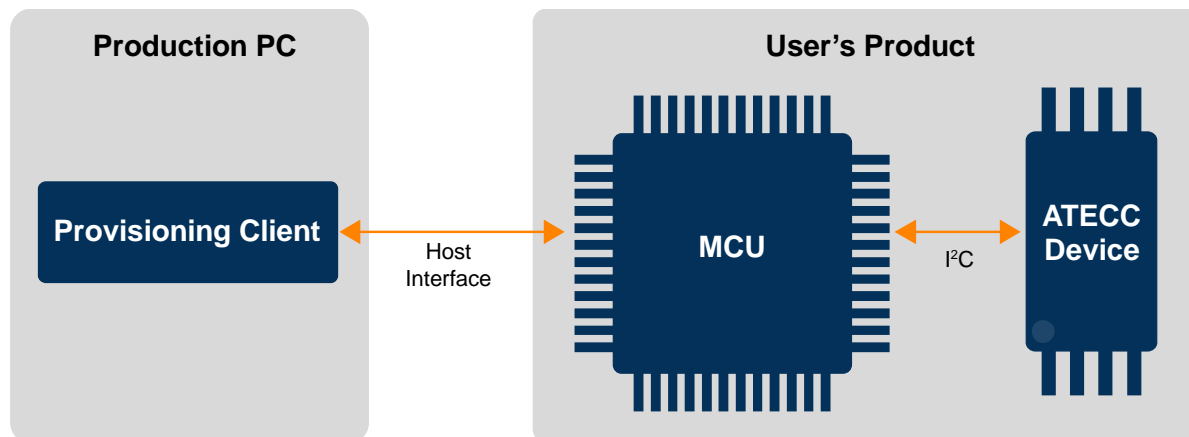
The diagram below shows the different components that communicate during the provisioning process. The Provisioning Client communicates with the Provisioning Server and the Device Host, which ultimately provides the communication with the ATECC device.

Figure 4-2. Provisioning Communication Flow Diagram



The Device Host is the component that's most likely to be different for each user's production environment. Below are a number of example configurations.

Example 1: ATECC in User's Product

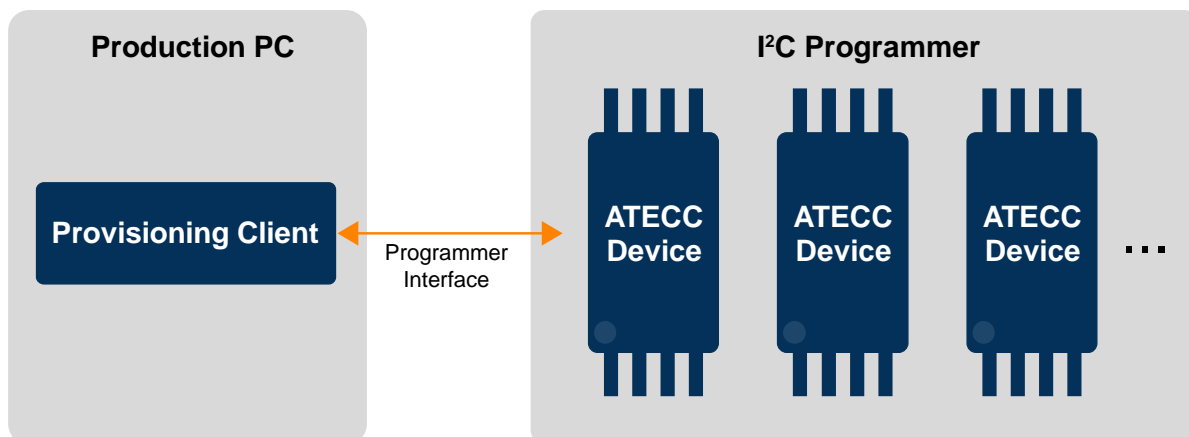


In this configuration, the ATECC device is already installed in a user's product. The Provisioning Client needs to communicate through the MCU (Device Host) in order to provision the ATECC device. The user needs to make sure they have the mechanisms in place in the MCU firmware to forward communication from the Provisioning Client to the ATECC device and vice-versa.

The Provisioning Client also needs to implement the host interface as a CryptoAuthLib HAL driver. See Section 5.2.1, CryptoAuthLib HAL Driver for more details.

This is the architecture used in the Provisioning Client CDC Example.

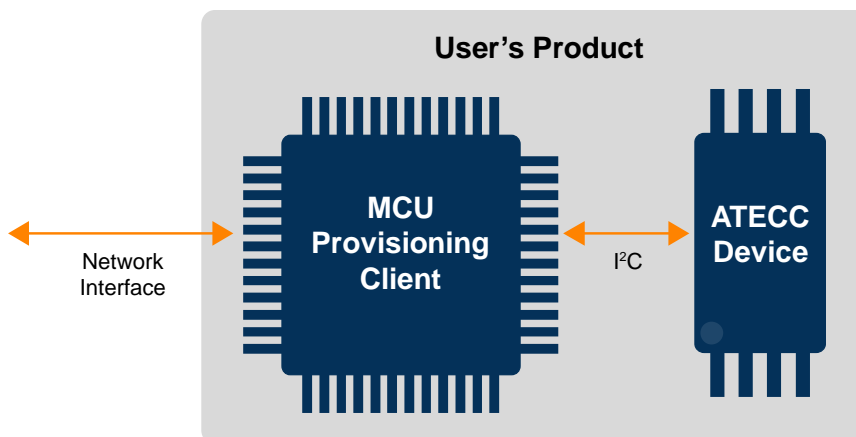
Example 2: ATECC in Programmer



This configuration has the ATECC devices provisioned in a programmer before being installed on a board or product. The programmer could be as simple as a USB to I²C adapter that communicates to a single part or a larger gang programmer handling many parts at once. The user needs to add support for communicating with the programmer to the Provisioning Client.

**TIP**

It is worth noting for programmers handling multiple parts at the same time, the provisioning process, by its nature, is individualized for each device. This means that the data and commands for each device are different.

Example 3: Provisioning Client in User's Product

This configuration has the ATECC device already installed in a user's product and the Provisioning Client running on the product's MCU directly. This simplifies the device connectivity, as the MCU often has direct access to the I²C bus it's on. However, the Provisioning Server connectivity gets a little harder. If the product has a network interface (e.g. WiFi) it would be possible to bring it up for the purpose of enabling communication with the Provisioning Server.

5 Provisioning Library

The Provisioning Library provides a framework to guide the provisioning process, as well the logic to communicate with the provisioning server to request certificates and validation data.

The ProvisioningClientCDCEExample application provides a sample application illustrating the use of the Provisioning Library.

5.1 Provisioning Stages

Once a device is ready to be provisioned, the provisioning process is kicked off using the `atcaprov_provision()` function. That process progresses through 10 stages as listed below:

1. **Write Configuration Zone** – This sets up the ATECC device's configuration, which determines how its various slots can be used. In the ProvisioningClientCDCEExample, if a new configuration isn't specified using the `--config` argument, then the default configuration in `atcaprov/atecc508a-config.h` is used.
2. **Lock Configuration Zone** – The Configuration zone is now locked to prevent any further changes to the device's configuration.
3. **Read Configuration Zone** – The device's Configuration zone is now read in full, including its serial number and other read-only data. This is required later when requesting certificates for the device.
4. **Generate Device Key Pair** – The device's identity is represented by a key pair of private and public elliptic curve keys. This stage commands the device to generate a new private key using its internal secure Random Number Generator (RNG). That private key is never accessible outside the device. The public key is returned to the provisioning process and is required later when requesting certificates for the device.
5. **Request Certificates** – Using the device's configuration and public key from the previous steps, the provisioning process now sends a request for device certificates to the Provisioning Server which waits for a reply (see Section 5.3, Provisioning Server Communication).
6. **Write Certificates** – With certificates from the Provisioning Server, the provisioning process saves them in a compressed format to the device. Reference the application note, "Compressed Certificate Definition," for more details located at:
www.atmel.com/Images/Atmel-8974-CryptoAuth-ATECC-Compressed-Certificate-Definition-ApplicationNote.pdf
7. **Write Non-Certificate Data** – At this point, the user (Provisioning Client) is given an opportunity to write any additional non-certificate data to the device such as symmetric keys, secret data, and public data. Since the Data zone is unlocked, slot permissions set by SlotConfig in the Configuration zone are ignored, and all slots can be written to (but not read).
8. **Lock Data Zone** – The Data zone is now locked and the slot permissions set by SlotConfig take effect.
9. **Lock Individual Slots** – If specified in the arguments to `atcaprov_provision()`, individual slots can be slot locked. This renders them read-only regardless of SlotConfig permissions.
10. **Validate Certificates** – If requested in the arguments to `atcaprov_provision()`, the provisioning process can finish up by validating the certificate chain and device keys. This process rebuilds the full certificates from their compressed form on the device, perform a full chain verify on the certificates, and validate the device's private key against the public key in its certificate. The process can be seen in the `atcaprov_validate()` function in the `atcaprov_validation.c` file.

Note: The `ATCAPROV_CERT_VALIDATION` symbol must be defined to enable certificate validation.

5.2 Prerequisites

Before the provisioning process can start for an ATECC device, the CryptoAuthLib library needs to be configured to communicate with that device. There are two stages to that process:

1. Configure CryptoAuthLib with the needed HAL driver. This is a compile-time configuration.
2. Initialize the CryptoAuthLib basic API with the device to provision. This is a run-time configuration.

5.2.1 CryptoAuthLib HAL Driver

The Provisioning Library uses the CryptoAuthLib library to communicate with the ATECC device. This library provides a clean method for communicating with ATECC devices at a logical command level, which is accomplished through a Hardware Abstraction Layer (HAL). The nitty-gritty details of actually communicating with the ATECC device are implemented as HAL drivers.

While CryptoAuthLib provides a number of HAL drivers, certain configurations may require developing a new driver for the ATECC host. Please reference the application note, “CryptoAuthLib Driver Support,” at: www.atmel.com/Images/Atmel-8984-CryptoAuth-CryptoAuthLib-ApplicationNote.pdf

In the Provisioning Client CDC Example application:

The device host is a USB CDC (com port) board running the kit protocol interpreter firmware. Kit protocol is the communication method used by CryptoAuth kits. The Provisioning Client CDC Example application compiles the following files to support the CDC kit protocol HAL driver:

From the `cryptoauthlib/lib/hal` folder:

<code>atca_hal.c</code>	Generic HAL routines.
<code>atca_hal.h</code>	Generic HAL routine declarations, structures, and constants.
<code>hal_win_kit_cdc.c</code>	HAL driver for Windows USB CDC devices using Kit Protocol.
<code>hal_win_kit_cdc.h</code>	HAL driver for Windows USB CDC devices using Kit Protocol.
<code>hal_win_timer.c</code>	Windows timing routines.
<code>kit_phy.h</code>	Common kit protocol routines to be implemented by the HAL driver.
<code>kit_protocol.c</code>	Common kit protocol support routines.
<code>kit_protocol.h</code>	Common kit protocol routine declarations, structures, and constants.

On the device host-side, the `cryptoauth-kit-d21-host` project provides an example implementation of the kit protocol interpreter for the SAMD21 Xplained Pro board www.atmel.com/tools/ATSAMD21-XPRO.aspx.

5.2.2 Initialize the CryptoAuthLib Basic API

During run-time, the Provisioning Library relies on the CryptoAuthLib basic API. This API requires `atcab_init()` to be called with an appropriate device configuration. Some HAL drivers may roll discovery of the device host and devices into the `atcab_init()` function, while others may use the `hal_*_discover_buses()` and `hal_*_discover_devices()` functions.

In the Provisioning Client CDC Example application:

`atcab_init()` is called with a default CDC kit configuration to discover any available CDC kit hosts and devices.

```
ATCA_STATUS status = atcab_init(&cfg_ecc508_kitcdc_default);
```

If that function returns `ATCA_SUCCESS`, the Provisioning Client application knows there is an ATECC device available to provision.

5.3 Provisioning Server Communication

The Provisioning Server is a TCP/IP server that the Provisioning Library needs to communicate with in order to request certificates from a Signer Module for the ATECC device being provisioned. While the Provisioning Library handles the messages, the user's Provisioning Client needs to provide the actual connectivity.

To that end, the user's Production Client is needed to implement five functions. These functions are declared and documented in the `atcaprov_user_funcs.h` file within the provisioning library, `atcaprov`. They are called by the provisioning process started by the `atcaprov_provision()` function.

5.3.1 `atcaprov_certreq_start()`

Called to indicate that the provisioning process wants to communicate with the Provisioning Server. This can be used to open a connection to the Provisioning Server. If a connection has already been established, then this function doesn't need to do anything and can just return 0 (success).

5.3.2 `atcaprov_certreq_send()`

Called to send data to the Provisioning Server. The user doesn't need to interpret the data in any way, but just send the raw bytes on to the Provisioning Server. This function shouldn't return until all the data has been sent.

5.3.3 `atcaprov_certreq_rcv()`

Called to receive data from the Provisioning Server. When called, the provisioning process indicates how many bytes it's expecting. If the function returns and indicates fewer bytes were received than expected, then the function is called again by the provisioning process to receive the full reply.

5.3.4 `atcaprov_certreq_done()`

Called to indicate that the provisioning process is done communicating with the Provision Server for the current ATECC device it's provisioning. This can be used to close the connection to the Provisioning Server. If a persistent connection is being maintained (that is, not opened and closed for each device), then this function doesn't need to do anything and just return 0 (success).

5.3.5 `atcaprov_request_error()`

Called only to report an error from the Provisioning Server. When called, an error code (from `atcaprov_return_codes.h`) and an error message are supplied.

5.4 Writing Non-Certificate Data

After the provisioning process writes the certificate data to the device, the user is given an opportunity to write additional data, secrets, or keys to the device's Data zone before it's locked. This is implemented through the `atcaprov_write_data_zone()` user-implemented function defined in `atcaprov_user_funcs.h`.

The `atcab_write_zone()` function from the CryptoAuthLib basic API provides an easy way to write any data that's needed. Note that one can only write data in 32-byte blocks when the Data zone is unlocked.

If no additional data needs to be written, then the function can just return 0 (success).



TIP Care should be taken not to overwrite the existing certificate data.

5.5 Progress Notification

The provisioning process, through `atcaprov_provision()`, provides a callback mechanism to inform the Provisioning Client about the status of the provisioning progress. This is through the `atcaprov_notify()` user-implemented function defined in `atcaprov_user_funcs.h`.

This function is called by the provisioning process to indicate the various stages and their status to provide feedback to the Provisioning Client.

There are 10 provisioning stages as listed below:

1. **ATCAPROV_STAGE_CONFIG_WRITE** – Writes the ATECC device's Configuration zone.
2. **ATCAPROV_STAGE_CONFIG_LOCK** – Locks the ATECC device's Configuration zone; this renders it read-only.
3. **ATCAPROV_STAGE_CONFIG_READ** – Reads the ATECC device's Configuration zone. This includes the device's unique serial number and is required for the certificate request.
4. **ATCAPROV_STAGE_GEN_KEY** – Generates the ATECC device's key pair. The private key is internally generated by the device's RNG and never exposed outside of the device. The public key is returned and is required for the certificate request.
5. **ATCAPROV_STAGE_CERTS_REQUEST** – Requests certificates from the Provisioning Server.
6. **ATCAPROV_STAGE_CERTS_WRITE** – Writes certificate data to the ATECC device. The configuration of the Signer Module used by the Provisioning Server determines where the certificate data is stored.
7. **ATCAPROV_STAGE_DATA_WRITE** – Allows the Provisioning Client to write additional data to the Data zone. The client is notified through the `atcaprov_write_data_zone()` callback that the user implements.
8. **ATCAPROV_STAGE_DATA_LOCK** – Locks the Data zone. This enables the enforcement of the read/write permissions for each slot as determined by the slot's SlotConfig in the Configuration zone.
9. **ATCAPROV_STAGE_SLOTS_LOCK** – Slot lock any slots requested by the user. This turns them read-only regardless of SlotConfig.
10. **ATCAPROV_STAGE_VALIDATE_CERTS** – Rebuilds and validates the certificates and private key of the device. This is only available if the `ATCAPROV_CERT_VALIDATION` symbol has been defined.

Please see Section 15, Provisioning Stages for detailed information about each stage of provisioning.

For each of the above stages, the `atcaprov_notify()` function is called multiple times to update that stage's status. The following sequences can be expected:

- At the start of a stage, the `ATCAPROV_STATUS_START` or `ATCAPROV_STATUS_SKIPPED` status is given depending on whether it is attempted or skipped respectively.
- If the stage hasn't been skipped, then it completes with either the `ATCAPROV_STATUS_DONE` or `ATCAPROV_STATUS_ERROR` status. If there was an error, the provisioning process returns at that point after calling the notify function.

6 Revision History

Doc Rev.	Date	Comments
8970A	12/2015	Initial document release.

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.