Application Note

Implementing Secure Boot with the Atmel ATSHA204

Detailed Explanation

Introduction

Most systems that use programmable nonvolatile memory for their operating program – whether it's large or small – can take advantage of a secure boot process. Secure boot is a method of ensuring that the operating program for a system is authorized, typically by the OEM that designed and built the system. By ensuring that the operating program is authentic, the OEM can prevent unpredictable system performance, safety or regulatory violations, excess warranty costs, lost revenue, and more.

The basic outline for implementing secure boot is provided in the document, "Secure Boot Simplified." This document provides a more in-depth look at the secure boot process and addresses many of the common implementation concerns.

1. Secure Boot Process

Secure boot procedures using an external symmetric-key security device follow this general flow:

1. At the OEM site, a hash algorithm is used to create a digest of the operating program. This digest is combined with a validating secret to create the signature.
2. Both the program and the signature are stored in the flash, perhaps at a third party site like a subcontractor.
3. On startup, the boot program calculates the digest of the operating program using the same hash algorithm. It also reads the signature from the flash memory.
4. The boot program transmits the digest and the signature to the security device.
5. The device combines the digest with the secret to create its own signature, and compares it with the signature passed to it by the boot program. The security device passes a "yes" (comparison succeeded) or "no" (signatures do *not* match) back to the processor.
6. If the validation fails, the boot program may prompt the user to download a new, authentic image, and return an error indication or take other action.

The security of this procedure rests on two assertions: the boot program cannot be changed by an adversary, and the value of the verification secret cannot be learned by an adversary.

- If the boot program can be changed, then the adversary can just remove the validation step and permit the system to operate with an unauthorized code. When the boot program is implemented via non-erasable Read-Only Memory (ROM), there is high confidence the program is unalterable. Various methods exist for implementing boot code using internal or external nonvolatile memory. Many flash microprocessors include some sort of flash 'lock' on one or more sections of the internal memory which greatly limits the ability to alter the boot program.
- If the adversary can get the secret, then they are able to generate acceptable signatures for fraudulent operating programs. There are many methods of extracting a secret from a device, including analyzing the signals to/from the device, probing the internal signals of the device and many other schemes. Since security is a common need in digital systems today, it is to be expected that a wide variety of successful attack methods have been developed for use against common storage devices.

The Atmel® ATSHA204 cryptographic authentication device includes a broad range of hardware security features designed to prevent such adversaries from obtaining the secrets stored in the device. These include active metal shields over the device circuitry, encrypted internal memories, detection of external voltages that are out of specification, internal clock generation and many other features.

Note: The security of this procedure does *not* depend on the secrecy of the information passing between the ATSHA204 security device and the microprocessor. Watching the bus traffic with a logic analyzer or other tool provides no information that could be used to permit changes to the flash memory on this system or on any other system.

2. Atmel ATSHA204 Details

The ATSHA204 implements the SHA-256 hashing algorithm for the signature generation process. Like all cryptographic hashing algorithms, SHA-256 has the following two properties:

- An adversary that has access to both the input and the output digest cannot make a change to the input (or create an entirely different input) that has the same digest, which would be termed a "collision."
- An adversary that has access only to the digest cannot determine the value of the input to the hash algorithm that created that digest. These algorithms are "one-way."

Of course, no algorithm is perfect, and there is some small probability that these properties could be violated with enough attempts. Because the digest of SHA-256 is 256 bits long, the number of required attempts to find a collision is at least 2^{128} – typically considered to be sufficient for the next 20 or 30 years. Another potential problem is that clever folks will find a way to crack the algorithm with fewer attempts. SHA-256 has been well analyzed and is generally recognized by most governments and cryptographers as quite secure.

3. Common Secure Boot Questions

3.1 Can't the adversary simply modify the system hardware to include a fake security device that always sends back a "yes" answer?

Yes. The adversary can perform this modification to the system, inserting a small microcontroller that would send back the appropriate signaling at the right time to always indicate that the verification succeeded even if the signature was invalid. The ATSHA204 has an effective strategy to prevent this type of attack — the copy mode of the CheckMac command.

Essentially, this mode adds a second layer of authentication on top of the operating code validation. This second layer authenticates that the validation operation is being performed by an authentic ATSHA204 and prevents system operation if a fraudulent chip is present.

The CheckMac command is normally used during secure boot, accepting the digest and signature as input and returning a boolean indicating the success or failure of the comparison. In copy mode, in addition to returning the boolean to the system, the device copies a separate secret to a temporary key storage location only if the comparison succeeds. This separate secret can be combined with a unique challenge from the system to generate a unique response.

Of course, this boot secret must be compiled into the boot program, leaving it exposed to potential attack. If it is a ROM, then a visual analysis of the mask layers in the microprocessor may reveal the secret. This attack is beyond the capability of a typical engineer, but is possible in a lab with the appropriate fine line semiconductor reverse engineering equipment and expertise.

Since the boot secret is in ROM, it cannot be changed in the field, so its compromise can be published. Nonetheless, it is *not* the important validation secret, and its publication only permits a certain kind of hardware modification to take place.

The ATSHA204 must be preconfigured in the following manner:

1. Store the validating secret in Slot 0 to 7. Set the CheckOnly bit in the configuration section for this slot.
2. Store a boot secret in an odd slot directly after the temp Signature slot. For example, the boot secret is stored in Slot 1 if the Signature will be stored in Slot 0.
3. Enable single use for the slot holding the boot secret and set the corresponding UseFlag to 0, preventing its use for any other purpose.
4. Set ReadKey for the slot holding the boot secret to zero.

The boot program would use the following procedure to take advantage of this capability:

1. The boot program sends the code digest to the NONCE command of the ATSHA204 with mode=3 (pass through).
2. The boot program sends a DeriveKey command to the even Signature slot directly in front of the Boot Secret slot.
    a. The boot program will pass in the Signature slot and the Validating Secret slot.
    b. This will cause the Signature slot to store the Signature of the Hashed code.
3. The boot program sends a random NONCE command to the ATSHA204.
4. The boot program will calculate the random number based on the seed returned from the ATSHA204.
5. The boot program will calculate a MAC command based on the calculated random number and the Signature.
6. The boot program then runs the CheckMac command with mode=1 and ClientResp=MAC response on the Signature slot. If the ATSHA204 returns an affirmative response, then the boot secret is loaded into TempKey. Continue with the following steps to validate this response without putting the device in IDLE or SLEEP.
    a. Generate some type of Challenge that is different for every boot. This could be the current time, a random number, the least-significant bits from an A/D converter, or any other varying value, or a combination of the above. It should be combined in some manner with a system-specific number, such as a serial number.
    b. Execute the MAC command, specifying that the inputs to the digest should include TempKey (the temporary storage location to which the boot secret has been transferred) and the Challenge (generated in the preceding step), which should be passed as the input challenge. The resulting digest calculated by the ATSHA204 is then passed back to the system.
    c. The boot program, which has the boot secret compiled into the code, computes the expected response in hardware or software using the SHA-256 algorithm. If the result of this computation matches the one from the ATSHA204, then normal operation proceeds. Otherwise, the boot code can assume the hardware has been compromised and can take the appropriate actions.

Some Atmel microprocessors can be supplied with a standard secure boot capability. Contact the local Atmel sales representative for more details.

3.2 What happens if the attacker uses a different version of the microprocessor, one that has a different boot program and does not include secure boot?

This is another type of hardware modification that is generally very difficult or impossible to perform on a finished system. Nonetheless, it is the mechanism that a clone maker might employ to build fraudulent copies of an OEM system.

If the processor manufacturer offers a non-secure version of a device that otherwise includes a secure boot ROM, then the two versions would normally operate differently. A simple clone using the non-secure processor that does not include the properly personalized security device would not operate properly. Most IC suppliers, including Atmel, carefully control the shipment of chips containing custom, secure boot code. If a fraudulent entity were to contract with one of these suppliers for a special version of a device to build clones of a genuine OEM system, then the supply of the device could easily be halted.

See Section 3.10, which describes a method of using the security device for the purposes of anti-piracy of the firmware. This capability can be used at the same time with the same device used for secure boot, increasing the overall protection level.

3.3 What about other possible hardware modifications a hacker might make to the system to circumvent the secure boot procedure?

It is often the case that an OEM has fewer concerns about its customers physically modifying their systems. The number of such attempts is usually quite low, especially if the systems are complicated, expensive, or physically difficult to modify.

Nonetheless, it is vital that even those with the engineering skills to modify the system not be able to retrieve information that could be published on the Internet, which would then permit others to modify their systems without having to perform difficult or risky procedures. By using a hardware security device such as the ATSHA204, OEMs can be confident that software downloads alone will not be sufficient.

3.4 What happens when a new operating program is downloaded to the system? Can every system in the field have the same downloaded program? Can the operating program be customized for a particular system?

New operating programs can be downloaded to a system with secure boot in exactly the same way as for a system without secure boot, with the sole exception that a new signature should accompany the downloaded program. This program and signature combination is stored in flash in the usual place.

It is important to note that the download process itself does not need to be secure. If the user downloads a fraudulent program, the system will catch this at the next reboot when it attempts to run the code. Of course, it would be helpful if the download manager used the ATSHA204 in the system to validate the download and issue a warning to the user when a new download has a problem that will prevent its use.

If the OEM desires that a single download image be able to be applied to every system in the field, then the ATSHA204 devices in every system must contain the same validating secret and the same signature can be used for every system.

In some cases it may be possible for each system to have a unique signature, which can offer additional security. This is easy to implement on the ATSHA204 because every ATSHA204 contains a 72-bit serial number that is guaranteed to be unique. When the system is manufactured, instead of programming the same validation secret into every device, a root validation secret is combined with the serial number using an algorithm of the OEM’s choosing, possibly the SHA-256. The result of this cryptographic operation, usually referred to as a diversified secret, is then stored securely in the ATSHA204. This value is used to generate the signature for the operating program that is stored in the flash memory of that particular system.

Later, when the software is to be updated, the following process takes place:

1. The server reads the serial number from the system via the network. If desired, it can be compared against a “black list” of bad serial numbers.
2. The server validates the serial number by sending a random number (challenge) to the system, which passes it to its onboard ATSHA204. The system runs a MAC command on the ATSHA204 with the challenge as input, using the boot validation key as the secret for the computation. The output of the MAC command (response) is sent back to the server.
3. The server calculates the expected diversified secret, the one that should be in the system, based on the root validation secret stored in the server and the transmitted serial number from the system.
4. Using the diversified secret, the server validates the serial number response and returns an error if the validation fails.
5. If the serial number validation succeeds, then the server signs the new code image with the diversified secret and then sends the code image and signature to the system.

3.5 How can the Advanced Encryption Standard (AES) algorithm accelerator found in many standard microprocessors be used to generate the flash memory program digest quickly?

The method used to generate the digest of the operating program – both at the factory where the signature is generated and at run time where the signature is checked – is independent of the security device algorithm. Atmel offers a number of microprocessors that include AES hardware accelerators, which can be used quite effectively for digest generation.

Both the **C**ounter with **C**ipher Block Chaining-**M**essage Authentication Code (CCM) and **G**alois/**C**ounter **M**odes (GCM) of the AES standard are designed to implement an authentication mechanism, and are widely accepted in the cryptographic community. In addition to the AES computation over the operating program space, these modes require only the computation of a few extra blocks of AES. There are a number of other protocols available for implementing a hash function using any block encryption algorithm.

Most hardware AES accelerators can accommodate one or the other of these modes. In particular, the CCM mode was designed to be efficient for hardware implementation and is often included in the hardware capability.

3.6 If the operating program is very large, it may take a long time to create the digest of the entire flash memory on boot. Is there a method for speeding up boot process to an acceptable time?

In fact, this is a real problem with today’s very large flash memories. Some Atmel microprocessors incorporate hardware hash accelerators, which have very high performance and can quickly calculate the digest over a large program stored in memory. If the microprocessor does not have an appropriate hardware accelerator, then the issue can be addressed in the following way:

- The actual code validated on boot is only the application loader itself. Within the application loader code (which is stored in flash, but is completely trusted because it is signed) would reside the procedures for using the ATSHA204 to validate the subsequently loaded code. This code may be loaded only one piece at a time. Validation of the entire operating program can thus be spread out in time.

  Note: As the validating key is stored in the ATSHA204 and not the application loader, the fact that the loader code itself is visible in the flash memory doesn’t affect the security of the system.

  Depending on the system architecture, this validation might also be implemented as part of the normal memory management procedure used to bring blocks of executable code into an internal cache.

  Example: This might work in a hypothetical medical display unit as follows: The ROM boot program validates the application loader, which then validates the display application code prior to loading it into memory. The display application, which is now trusted, loads the requested image and signature into memory. It then verifies the signature of the image before displaying it.

- Another strategy for speeding up validation is to make the download manager a trusted module, perhaps one that can be executed only directly after a secure boot has taken place. In such a configuration, a bit is set somewhere that indicates to the application loader that instead of executing the program already stored in flash, it should execute the download feature, which is trusted because it is part of the code validated by the boot program at startup.

  The download happens in the usual way, but in this case, the signature is validated prior to updating the flash memory with the program. Since downloads happen infrequently, it is usually acceptable to take the time to generate a digest over the entire update image. If the runtime signature validation is not performed, then it would be possible to use some hardware mechanism to load the flash chip with an unauthorized program, thus bypassing the download verification step.

3.7 What happens if the validation secret leaks out somehow? Can it be updated along with the signatures?

The ATSHA204 provides two methods for updating keys within devices in the field: encrypted writes and derived keys. They provide mechanisms to address any secret compromise, as well as a procedure for adhering to any security policy that requires periodic replacement of commonly used secrets.

For encrypted writes, a secondary parent key can be used to encrypt a new value to be written into the slot containing the validation secret. A secure server individually creates a protected package for each device comprised of the new key encrypted using the parent and a message authentication code (MAC), which prevents the key from unauthorized updates. The device always generates a random nonce to ensure that each key update procedure is completely unique.

For derived keys, the existing validation key can be “rolled,” or combined with a random or fixed value to create a new key. For very high security systems, this provides a vehicle for using each key a very limited number of times. Alternatively, the device can be instructed to combine the parent secret with the input value to create a new key. Depending on the configuration, this procedure can either be broadcast to all systems or targeted to just to a single device.

It is best if the original validation secret remain confidential. To achieve this, Atmel recommends the same ATSHA204 device be used to generate the signatures at the OEM development site. This ATSHA204 should be configured slightly differently from the ATSHA204 devices used in the actual system:

- The ATSHA204 should be configured in “password checking mode,” the same mode described in Section 3.1.
- Key Slot 0 should contain the SHA-256 digest of an authorizing passphrase, which should be known by the appropriate responsible parties, and be changed when necessary.
- Key Slot 1 should contain the validation secret, which need not be known by anyone in the company, thus preventing any disgruntled employee from publishing it.
- Appropriate policies, including physical security of the systems, redundancy, storage, and documentation should be developed and enforced

Atmel supplies a simple demonstration kit that includes a USB interface and PC configuration software that can easily be used for this purpose.

3.8 Asymmetric algorithms like RSA and Elliptic Curve Cryptography (ECC) don't require the storage of a secret in the system, just a public key. Why not simply use these algorithms and save the cost of the security device?

RSA, ECC and similar algorithms are excellent tools for signature validation. They can be used effectively in a secure boot scheme. However, systems incorporating a hardware security device like the ATSHA204 offer a number of benefits over a software-only validation scheme:

- Most asymmetric algorithms are relatively slow usually in the range of 1000 to 10,000x slower than a symmetric algorithm like SHA-256. Using external hardware like the ATSHA204 to implement the signature checking significantly speeds up the boot process.
- Most asymmetric algorithms also require significant code space in the boot program, which might be limited by the particular microprocessor or memory device chosen. In one academic study using the Atmel Mega128, code for ECDSA took about 20K bytes.
- The public key cannot be easily changed for different versions of the system or instances of the system in which different features are enabled, making ROM storage of the public key less effective. Security policies at an OEM might also prohibit the use of a single asymmetric key pair for more than one system type. Storing the keys in an external hardware device solves this problem.
- There is no special device in the system to prevent a clone maker from copying the system by procuring the same components. None of the features described in Section 3.10 can be used.
- The private key must be carefully protected at the OEM, the subcontractor and/or the download server. This may require more complicated or expensive hardware as compared to the simple solution using the Atmel demonstration kit described in Section 3.7.

3.9 How can the secret be programmed into the security device without divulging it to third-party subcontractors?

The ATSHA204 implements a series of features designed to permit the device to be personalized without the risk of disclosure to an attacker with visibility to the pins during programming. All personalization steps can be fully encrypted to prevent a snooper with access to the contractor network (or even the pins of the device being personalized) from determining the secret. Generally, this requires the secrets to be stored in an OEM-controlled, secure system that manages the personalization sequence.

For an additional charge, Atmel offers a secure personalization service in which the devices are shipped to the subcontractor already preprogrammed with the OEM secrets and locked against any modification. Atmel uses industry standard hardware security modules (HSM) to store the OEM secrets, and perform the appropriate programming sequence. These same HSMs typically are used by banks and governments to safely store secrets.

3.10 Are there other system security needs that can be addressed with the Atmel ATSHA204?

There are many uses of the ATSHA204 in a typical system. Those that pertain to the concept of secure boot are included here.

Multiple Keys, Separate Usage:

As a general rule, it is wise to use a separate key for each security feature so that an unanticipated compromise of one key does not lead to the overall loss of system security. The ATSHA204 makes this easy by including 16 separate keys that can all be independently configured for different use models.

Note: This same large key storage space allows multiple different security models or usage methods to be accomplished at the same time, with the same device. This can increase the value of the ATSHA204 to both the OEM and end customer.

Software Connection to the Atmel ATSHA204:

The operating program should ‘connect’ to the ATSHA204 as part of its normal operation. In this manner, several of the hardware attacks described above can be prevented. There are many ways to implement this connection:

- Include a simple, fixed challenge response at various places within the code at compile time. If the ATSHA204 device is missing, the response will not be available and the program can stop working or issue a warning message. Each time a new version of the software is released, the location of the checks and their values can be changed.
- Use a known state within the program at various points to create a diverse challenge. The checking of the response can also be distributed, as a single response can be read many times from the ATSHA204. In this manner, it becomes difficult to model the expected output of the ATSHA204 with a simple microcontroller.
- Compile one of the ATSHA204 secrets into the program and then provide a changing or random challenge so that a replay of the challenge responses from one system transaction will not work on this or any other system next time.
- If the system connects to a network, then the software and remote network can cooperate to perform a random authentication, which can only be properly completed by an authentic ATSHA204.

This method is also used to prevent piracy of the operating program. Hackers can reverse-assemble or reverse-compile the software to remove the checking methods. However, as the number and diversity of the checking methods increases, this takes more and more time. Also, each program revision should use a new set and location of checks, forcing the hackers to reverse-engineer each program revision independently.

Encryption Key Generation:

OEMs often encrypt the program stored in an external flash device, but there remains the problem of where to store the key for decryption. Usually it is just compiled into the program in some diverse manner, perhaps combined with serial numbers, split into many pieces or otherwise hidden.

The ATSHA204 provides a method to significantly improve the key generation method. Information stored in the program is combined with keys stored in the ATSHA204 to directly or indirectly create a decryption key that is used to decrypt important blocks of code using software or hardware AES in the main processor. It thus becomes difficult for the hacker to determine all the keys and create a fully functional clone, especially if some of these blocks are only infrequently executed.

This capability is also used to generate encryption/decryption keys over a network for locally stored data or messages. For each file or message, a random number is generated by the random number generator in the ATSHA204. That random number is stored with the file or transmitted along with the message. To encrypt or decrypt the data, the random number is combined with a key stored in the ATSHA204 prior to being used by the encryption/decryption hardware or software in the processor.

Of course, it is quite easy to use this same mechanism to generate a decryption key for a downloaded image – just include the random number at the beginning of the download package, and the system can generate a session key using the ATSHA204 to decrypt the downloaded package on the fly. If the decryption key in the ATSHA204 is diversified, then the download package can be directed to a single system. Or there can be two decryption keys in the ATSHA204 – one for broadcast downloads and one for system-specific downloads.

Network Connected Validation:

For any system that connects to the OEM website for any reason (warranty maintenance, firmware updates, etc), an additional level of protection is possible.

1. Upon connection, the website sends a random challenge to the system.
2. The system passes this challenge to its ATSHA204, which should then execute the MAC command. Usually, the architecture should be a KeyID that is not used for any other purpose. This key should have been initialized at the factory with a secret value.
3. The system sends the output challenge of the MAC command back to the website.
4. The website calculates the same response value using a copy of the secret securely stored on the server. If the values do not match, the website can be programmed to indicate a problem to the user.

This same procedure can be offered on the OEM website as a service for users to remotely determine if their system is genuine.

Random Number Generation:

The high quality random number generator in the ATSHA204 can be used for any purpose, including bi-directional network authentication, session key generation, random anti-piracy challenges, etc.

4. Revision History

Doc. Rev.	Date	Comments
8753B	09/2012	Update procedure to reconfigure ATSHA204 and boot program. Update Atmel logo and disclaimer page.
8753A	03/2012	Initial document release.

Enabling Unlimited Possibilities®

Atmel Corporation
1600 Technology Drive
San Jose, CA 95110
USA
Tel: (+1)(408) 441-0311
Fax: (+1)(408) 487-2600
www.atmel.com

Atmel Asia Limited
Unit 01-5 & 16, 19F
BEA Tower, Millennium City 5
418 Kwun Tong Road
Kwun Tong, Kowloon
HONG KONG
Tel: (+852) 2245-6100
Fax: (+852) 2722-1369

Atmel Munich GmbH
Business Campus
Parkring 4
D-85748 Garching b. Munich
GERMANY
Tel: (+49) 89-31970-0
Fax: (+49) 89-3194621

Atmel Japan G.K.
16F Shin-Osaki Kangyo Bldg.
1-6-4 Osaki, Shinagawa-ku
Tokyo 141-0032
JAPAN
Tel: (+81)(3) 6417-0300
Fax: (+81)(3) 6417-0370