APPLICATION NOTE

Atmel AVR232: Authentication Using SHA-256

8-bit Atmel Microcontrollers

Description

Many embedded applications consist of multiple units and often it can be necessary for one unit to be sure of the identity of another connected unit. For example if a unit only can be guaranteed to work correctly together with a certain model of another unit, it needs to make sure what model it is. Just asking is not enough as the other unit can lie; some form of secure authentication is needed.

This document describes authentication methods between digital devices. They can be seen as a sort of lock and key; the lock wants to know that the correct key is present before opening the door. But a physical key is easy to copy, while a secure digital authentication is virtually impossible to copy.

The information in this document is intended as an introduction and overview to cryptographic authentication. The reader is encouraged to find more thorough information about specific algorithms and attacks from other sources, for example the “Handbook of Applied Cryptography” (see [1], which is freely available on the internet, and the Cryptography portal (see [2]) on Wikipedia.

Note that Atmel® cannot take responsibility for your company’s security guidelines. Consult with a specialized security firm to get your implementation validated.

1. Theory

Authentication can either be mutual, where both parts in the communication wants to be certain of the identity of the other, or one-way, where only one part wants to authenticate the other. In many embedded applications, a one-way authentication is probably enough. The authentication has to be secure even when an attacker can listen to all communication. The authentication does in no way protect the privacy of data sent on the channel; it only proves the authenticity of one or both parts at the time of authentication.

It is possible to prove the authenticity of every message, but it is more complex and creates a big overhead. In some applications it might be necessary though and then encryption (a cipher) or keyed Message Authentication Codes (MACs) can be used. This is not discussed in this document.

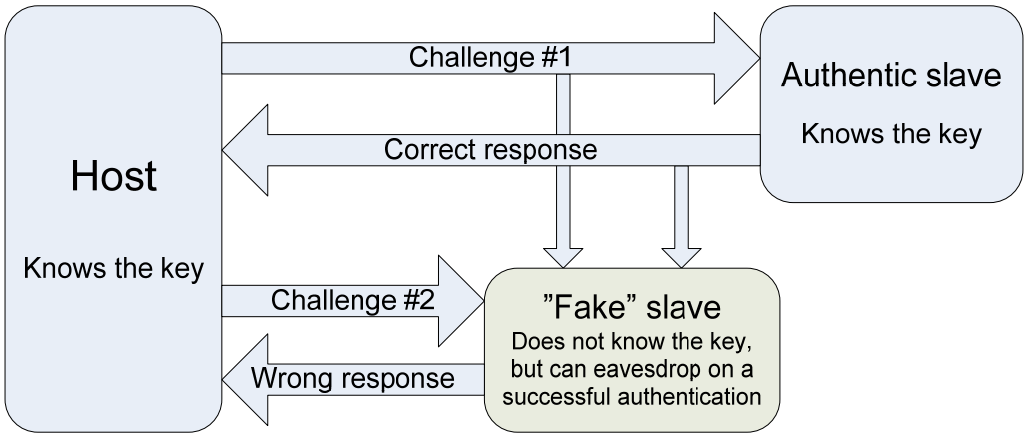
To be able to authenticate each other (or authenticate one part), the parts have to share a secret knowledge (that is, a secret key) and they have to be able to prove that they posses the knowledge to each other without revealing it to eavesdroppers.

1.1 Challenge-Response

A simple method for two devices to prove that they know the secret key is to exchange challenges (“puzzles”) that are only (and always) solvable with the use of the correct key. The results are then exchanged and checked if they are correct. This is called Challenge-Response authentication. For a one-way authentication, only the authenticating part has to send a challenge to the part that has to authenticate itself.

It is very important that the challenge is different every authentication, otherwise an attacker can hard-code the response, and then use a simple replay attack. The same applies to a small number of different challenges.

Figure 1-1. Example of a one-way challenge-response with an eavesdropper, highlighting the need to send different challenges on every authentication.



The transformation can either be a keyed hash function or an encryption function. Only symmetric ciphers are discussed in this document, asymmetric ciphers generally requires more computing power and is uncommon in embedded application.

A symmetric encryption algorithm always has a key that determines how transformation is made and the output is the same size as the input.

Hash algorithms transform an input of variable length into a fixed size output, called a one-way function. Note that it is important to use a keyed hash algorithm, as un-keyed hash algorithms can be computed by anyone. For a cryptographic hash, it shouldn’t be possible to find two inputs that generate the same output faster than trial-and-error (that is, trying different inputs until two with the same output is found, known as a brute-force attack).

There is no difference in security between using an encryption or keyed-hash, other than the individual differences in the algorithms considered.

1.2 Different transformations

There exist many different transformations/algorithms that are suitable for a Challenge-Response implementation on AVR®. The decision is a trade-off between code-size/memory-usage and security.

All algorithms described here are block based, meaning they operate on one block of data at a time, and if the input is bigger than the block-size, the main function of the algorithm has to be run multiple times. When doing multiple runs, the result from one block should affect the transformation of the next block. This is incorporated in cryptographic hash algorithms, but if an encryption algorithm is used, a secure mode of operation for the cipher must be used, such as Cipher Block Chaining (CBC) where each cipher output block is XORed with the next input block. Note that this is required only if the input is bigger than the block size of the chosen encryption, and is therefore not described in this document.

Generally the resources needed and execution times for each algorithm corresponds to the key and block size, but many algorithms are specialized to run fast in either hardware or software. If the authentication isn't run very often, it can be more important to minimize code-usage and memory than execution speed.

1.2.1 The DES family

DES (see [6]) is an encryption that uses a 56-bit key, which is small enough to brute-force within reasonably time (see [4]). Triple-DES (3DES or TDES) runs three passes of the DES algorithm, each using a different key and therefore increases the key size to 168 bits. But due to an attack known as meet-in-the-middle, the effective key size is only 112 bits. Both DES and 3DES operates on blocks of 64 bits.

DES, and especially 3DES, is quite slow in software, but very fast in hardware. AES is a better choice for a software implementation as it both runs faster and is more secure.

1.2.2 AES (a subset of Rijndael)

AES (see [7]) is an encryption that either uses a key size of 128, 192 or 256 bits. All key sizes use a block size of 128 bits. It is considered secure and is recommended by most institutions/cryptographers (as of August 2012).

1.2.3 HMAC (Hash-based Message Authentication Code)

HMAC (see [3]) is a common algorithm for implementing message authentication using a hash function. It can use any key-size, but key-sizes larger than the output size of the underlying hash function is meaningless, as the key is compressed down to the output size. And an attacker would only need the compressed key.

Any block-based hash algorithm can be used with HMAC, for example MD5 (see [8]) and all SHA algorithms (see [9]).

2. Key management

For a well-designed authentication, it is probably much easier and cheaper to get hold of the key using other, more drastic methods than breaking the authentication.

If the key is vital for the profit of the company, it should be considered top secret. There is no need for any employee to know the key; it can be randomly generated and securely distributed to the machine(s) that will program all devices that uses the key. The machine(s) does not need to be connected to any network and should be locked away in a high-security facility.

3. Non-invasive attacks

A non-invasive attack is an attack that only uses information available from the outside, for example the challenge and response, power consumption and execution time.

3.1 Brute-force

Trying all possible keys until the correct one is found is called a brute-force attack. If there are 100 different keys, it would take on average 50 tries to find the correct key. However, the attacker can be lucky and find the key on the first attempt, which is always a possibility.

For a 128-bit key it would take on average 2127 (~1.7*1038) transformations to find the correct key. The fastest computer in the world at the beginning of 2008, the IBM® Sequoia machine, is capable of 16.32*1015 floating point instructions per second. A machine capable of doing the same amount of transformations per second (which is highly unlikely, as most encryptions and cryptographic hashes are quite complex) would still need 330 trillion years on average to find the correct 128-bit key.

Other attacks try to decrease the number of possible keys to a manageable number by finding constraints on the key, for example dependencies between different parts of the key. Different algorithms are more or less sensitive to this kind of weaknesses; much research is put into finding those weaknesses and developing new algorithms that doesn't have them. Keeping up to date with the latest research for the algorithms you use is important.

3.2 Side-channel attacks

Side-channels attacks try to obtain information about what data the device operates on during the transformation by looking at "leaked" information, for example power consumption and execution time. It requires very good knowledge of the algorithm used, but for the sake of increasing security it should be assumed the attacker has all information except the key. For example an attacker can look at the power consumption when the key is read from memory and that way determine how many bits that are high and low in that part of the key. The fewer number of bits that are read from memory on each read, the more information about the key can be revealed.

Example: If you have a 128-bit key on an 8-bit architecture, it will take 16 reads to get the whole key from the memory. If all bytes contains exactly 4 high bits, there are only 70 different possibilities for each byte and a total of 7016 (~3.3*1029 or ~298) different keys. If any byte is less or more bits than four, there is even less possible keys.

Figure 3-1. Number of different bytes with exactly 4 bits high.

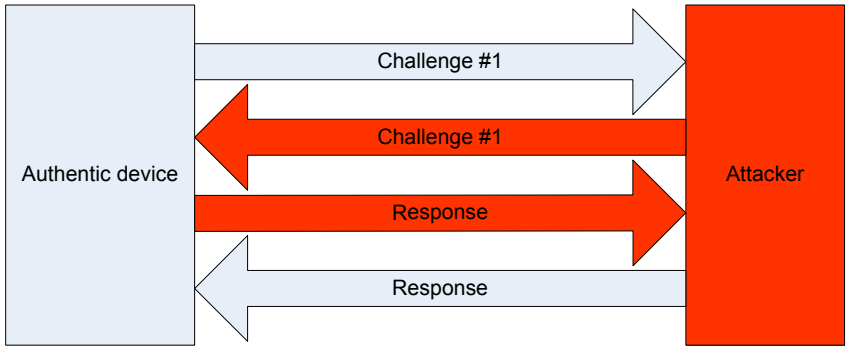
$$\frac{8!}{4!*(8-4)!} = 70$$

Timing analysis of the execution can also reveal information of the key. It is generally possible to make sure the algorithm runs in an equal number of cycles for all different key and data inputs and doing so is fundamental to the security of the authentication.

3.3 Reflection attacks

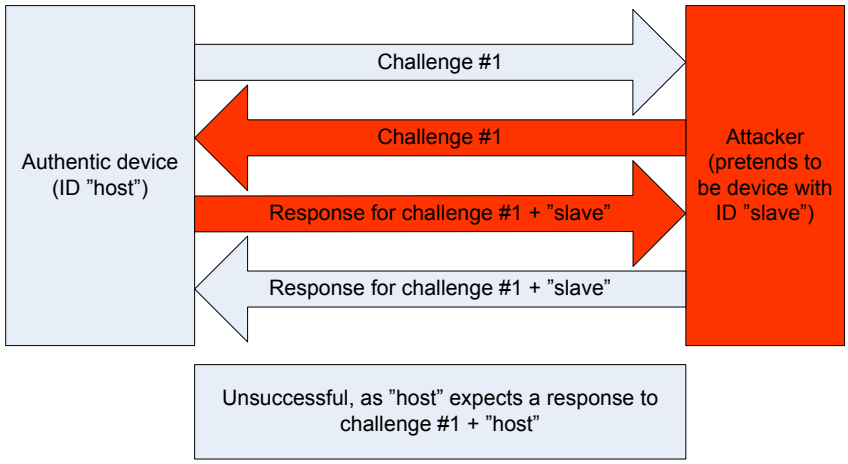
With a reflection attack, the attacker tries to fool an authentic device by making the authentic device do the authentication for it, see Figure 3-2.

Figure 3-2. A successful reflection attack.



This attack can be prevented by either defining an order for the authentication or appending ID information about the receiver to the challenge before calculating the response, see Figure 3-3. A possible variant of this attack is when the attacker sends the received challenge to another authentic device instead of the one that originally sent the challenge. This can also be prevented with the use of ID information.

Figure 3-3. An unsuccessful reflection attack thanks to appended ID information.



3.4 Collisions / "Birthday attack"

This attack is specific to hash algorithms.

Since hashes transform an input of variable size to a fixed size output, there has to be different inputs that generate the same output, and those are called collisions. Finding a collision for a specific hash output requires on average 2127 (~1.7*1038) tries for a 128-bit output, but because of the "birthday paradox", finding two arbitrary inputs that generate the same 128-bit output only requires on average ~264 (~2.3*1019) tries.

Those collisions can reveal information about the key. Attacks that exploit collisions have been found for weak hash functions (like MD4, MD5 and SHA-0, see [3]). But because the attacker has to collect responses from a device that knows the secret key, the efficiency is limited by the communication speed to that device and the number of devices the attacker can get hold of. Practically, it means this sort of attack is as infeasible as a brute-force attack, unless new, drastically more effective methods are found that both needs a lot fewer collisions are capable of finding them faster.

By restricting the allowed input size (the challenge size) key recovery gets even harder, as it might not be possible to find enough collisions without varying the input size.

4. Invasive attacks

An invasive attack is when the attacker physically alters the device to directly read out information. Most embedded devices with built-in non-volatile (for example, flash) memory have fuses/lock-bits to prevent reading and altering of the memory, an attacker can get free access to all memory if he manages to change the fuse.

It can also be possibly to read data from the memory by physically probing it. The only defense against that (in software) is to not store the key in non-volatile memory. By storing the key in volatile (for example, SRAM) memory, the attacker has to keep the power to the device which makes it much harder to read the data. This will of course have the consequence that a device that has been shut down no longer can authenticate itself (or authenticate others for that matter).

5. Authentication example specification using HMAC-SHA256

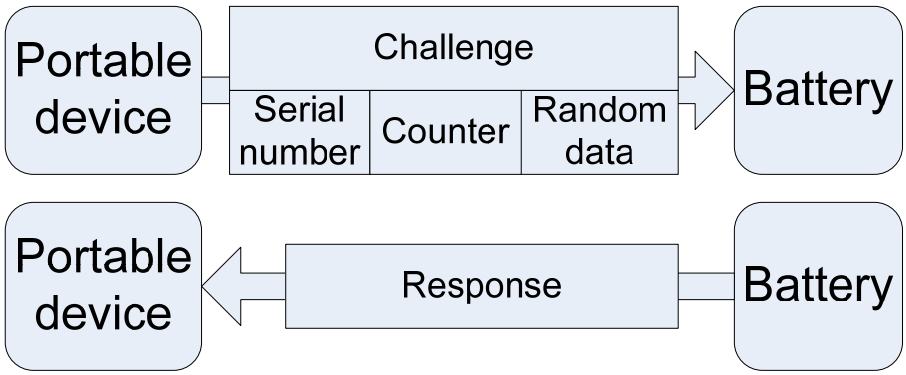
The most important thing to consider for a security specification is how much protection is really needed. What is the weakest link? Is the key stored and used securely in the production facility? Having a more secure firmware solution than what is needed wastes a lot of valuable code space and other resources, so identifying the threats and having an appropriate security is important.

For this example specification we want a portable device (the host) to be able to authenticate its battery. This can be used for service records, battery life-time guarantees and to guarantee the user that this specific battery is physically secure (it will not blow up…).

- Only the battery needs to be authenticated. It does not matter if the battery is used in another portable device
- A one-way authentication is chosen
  - Communication between the host and the battery does not need to be secret
- No encryption is needed
  - Every message passed between the battery and the host does not need to be authenticated. An attacker cannot get valuable information by inserting fraud messages on the communication channel
- The authentication only need to be done at startup (and then possibly at set intervals)
  - This portable device has a long expected lifetime; the authentication (and key) should be secure for many decades in the future
- HMAC-SHA256 is chosen as it can use large enough keys (256 bits) and has a sufficient output size (also 256 bits). Even with the improved computer capacity in the future and possibility of new attack methods, it is considered enough to make all attacks infeasible
- The challenge is set to a fixed size of 32 bytes (256 bits). This means a minimal amount of collisions (if any) can be found. One of the bytes is constant (used as status/error byte), further decreasing the possibility of finding collisions. And having “only” 2248 different challenges is well above what would be considered secure
- The remaining 31 bytes of the challenge should consist of a serial number (unique to each host), a counter and random data. The counter gets increased every time the host sends a challenge
  - The battery should never be shut down
- This enables the key to be stored in volatile memory only, which makes it harder to get the key with invasive attacks
- There are two possibilities to get the key to volatile memory:
  - The key can be programmed to non-volatile memory together with the rest of the program/data. At the first startup, the key is moved to volatile memory and erased from the non-volatile. The key should not be stored on the same non-volatile memory as the program code, as having code capable of overwriting the program code memory is generally avoided in all secure applications because run-away code could render the device unusable. If this method is chosen, non-started batteries should be stored very securely and the startup has to occur in a controlled environment

- The battery can be started at the manufacturing plant and the key transferred to it via the communication channel. It doesn't really matter if an attacker is capable of changing the key as it would only make the battery unusable in authentic hosts, but of course it has to be made sure that it cannot be read using the same mechanism
- Note that if the key storage on the host is less secure than it would be in non-volatile memory on the battery, it is just meaningless to bother about only storing it in volatile memory

Figure 5-1. Data sent during a one-way challenge-response.



6. References

1. Handbook of Applied Cryptography.
   http://www.cacr.math.uwaterloo.ca/hac/
2. Cryptography Portal.
   http://en.wikipedia.org/wiki/Portal:Cryptography
3. Full Key-Recovery Attacks on HMAC/NMAC-MD4 and NMAC-MD5. Pierre-Alain Fouque, Gaëtan Leurent, Phong Q. Nguyen.
   http://www.eleves.ens.fr/home/leurent/files/HMAC_CR07.pdf
4. COPACOBANA. A Codebreaker for DES and other Ciphers.
   http://www.copacobana.org/
5. HMAC. Multiple papers.
   http://www.cs.ucsd.edu/~mihir/papers/hmac.html
6. DES and 3DES. Federal Information Processing Standards (FIPS, US government).
   http://csrc.nist.gov/publications/fips/fips46-3/fips46-3.pdf
7. AES. Federal Information Processing Standards (FIPS, US government).
   http://www.csrc.nist.gov/publications/fips/fips197/fips-197.pdf
8. MD5. RFC1231.
   http://tools.ietf.org/html/rfc1321
9. SHA. Federal Information Processing Standards (FIPS, US government).
   http://csrc.nist.gov/publications/fips/fips180-2/fips180-2withchangenotice.pdf

7. Revision History

Doc. Rev.	Date	Comments
8184A	09/2012	Initial document release

Atmel | 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