Description

This application note discusses at a high level the security models one might use with the CryptoAuthentication™ family of devices.

The security models support the usage models presented in the Product Uses Guide, but with emphasis on the security aspects. Implementation of the security models can be accomplished by use of the datasheet, API guide and the provided libraries. Documentation can be found at http://www.atmel.com/products/cryptoauthentication/default.asp. This document presents realistic handling of keys, common attacks and message generation modes. All three versions of the chip are discussed, the AT88SA100S, the AT88SA10HS, and the AT88SA102S.

Three topics need to be covered before launching into the discussion of security models. An understanding of the central cryptographic algorithm of the chip, keys and personalization, and the usual attacks is central to understanding the models.

Crypto-Authentication™

High Level Security Models

Application Note

1. SHA-256

The primary function of a CryptoAuthentication chip is to generate a message authentication code (MAC) from both static and variable data. The MAC is generated on the chip, out of sight of snooping programs or attackers. Some components of the message used to generate the MAC are always stored in secure locations within the chip (static) and are unreadable from outside the chip, while other components used in the message come from the host and are variable. The secure nature of the MAC generation allows for varied uses in differing scenarios.

The datasheets for the various models cover MAC generation in some detail, and should be read and understood before the design of the security architecture is attempted.

The CryptoAuthentication device is the first low cost chip to generate MAC's using a SHA-256 engine; the chip is compliant with the FIPS 180-2 Secure Hash Standard. A block of data is fed into the SHA-256 engine and is mathematically reduced to a set number of bits, called a digest, of specific size, in this case 256 bits or 32 bytes. 1.16×10^{77} outcomes are possible (2^{256}).

For a hash algorithm to be useful and secure, a significant portion of the bits in the hash should change if a single bit flips in the input and no discernable pattern should be evident no matter how structured the input changes. Here are two nearly identical inputs to generate a SHA-256 digest, differing by only one bit (ASCII for "d" is 01100100 - for "e" is 01100101).

"The quick brown fox jumps over the lazy dog."
"The quick brown fox jumps over the lazy eog."

SHA-256 results, hexadecimal:
ef537f25c895bfa782526529a9b63d97aa631564d5d789c2b765448c8635fb6c
bd0cf2c3ee82be889a1b5b754d2e4a20e07d22da40c39380da061092ca1638d7

First ten bytes, binary:
1110111101010011011111110010010111001000100101011011111110100111
1011110100001100111100101100001111101110100000101011111010001000

Casual inspection will show that the single bit flipped changed the results significantly. This makes “walking in” a result hard to accomplish. Since the hash compresses data so drastically, it should be intuitive that the result tells almost nothing about the inputs used. The NSA and the universities involved in cryptographic research all agree that a brute force attack on SHA-256 is essentially impossible.

An AT88SA102S chip in normal use generates a MAC based on a selectable number of fields, both static and variable. Three fields are always used: a user defined secret protected inside the chip, a static internal secret key chosen by an index called a KeyID, and a challenge chosen by the host. Two optional fields are available and may be included in the values of the message: a 48 bit unique serial number and a 24 bit status number.

The static unique serial number can be used to insure that no two chips will produce identical results even if the User Defined Secret is set to the same value for all the chips. If the requirement is to keep peripherals from being interchangeable, including the unique serial number guarantees that one and only one peripheral will generate the correct digest under any circumstance. A good example is a door lock that can be programmed to admit only certain devices which will act as electronic keys.

It is also possible to have many devices generate the same digest by excluding the serial number of the device and programming the User Defined Secret to the same value. A given challenge will then yield the same result on many devices, and large numbers of predictable challenge-response pairs could be pre-generated.

Imagine a help desk person instructing a user over the phone to input some string to his driving software and to read back the results. The tech inputs the same word to the help-desk computer. If the results match, the device is in fact authentic. If they do not, the device is a clone or the call is a fraud. This is Model #1 below, the fixed-key model.

2. Keys and Personalization

This document and all the application notes, white papers, and datasheets for the CryptoAuthentication product line tend to use specific words to mean specific things in ways not necessarily found in a household dictionary.

The ATSA102S and ATSA10HS include an array of static **key** values, each 256 bits, programmed into the mask definition of the chip. These values cannot be read from the chip and are provided to the customer from Atmel via a secure data transfer process.

The AT88SA102S and the AT88SA10HS store an additional **User Defined Secret** in an array of 64 fuses. **Personalization** is defined as the act of programming these fuses. It should always be accomplished before the chip is used, and generally in a protected environment. The chips implement a method to confidentially load this secret into the fuse array protected through an internal static value called a transport key. If programmed in this manner, a third party sub-contract manufacturing line may securely program chips with values known only to the OEM.

Personalization of an AT88SA102S or AT88SA10HS chip consists of blowing fuses, and is one-way and one-time. Running the personalization command blows a fuse and disables any further writes. Care should be taken to personalize in an environment that prevents brown-outs and interrupted operation. After personalization, the chips should be tested to insure the process has completed correctly – half-blown fuses are possible if the process is interrupted and are generally unrecoverable.

Atmel offers fuse personalization on the manufacturing line of the chip, using a high-security Hardware Security Module (HSM).

The AT88SA100S includes a 256 bit key that is stored in an SRAM – but does not include a User Defined Secret, as the value of the key is generated and loaded into the chip by a third party and not Atmel.

Personalization of the AT88SA100S must be done after the chip is attached to a voltage source to maintain the SRAM key, and that tampering with a battery equipped with an AT88SA100S is almost guaranteed to clear the chip. Personalizing the key in SRAM prevents further writes to SRAM, and the value of SRAM is not available to any read command.

3. Common Attacks

Here is a list of some common attacks. In following section on models, the attacks will be covered for applicability to the specific model and configuration of the systems to counter the attacks will be discussed.

- Man-in-the-middle
- Brute force
- Replay attack
- RAM monitoring
- Reverse engineering the microcontroller program
- Hardware attacks

3.1. Man-in-the-middle attack

This attack is usually done with a bus monitor or logic analyzer. The bytes are captured as they cross the one-wire data pin to the CryptoAuthentication device, and the answer is captured as it comes back. In one direction or the other the connection between the host and CryptoAuthentication device is temporarily interrupted and the attacker will insert bits different than the authentic party intended. The datasheet is used as a decoding tool.

The one way nature of the SHA-256 hash algorithm defends against this kind of attack. Without knowing the additional secret bytes that are added to the input challenge inside CryptoAuthentication prior to the hash, the attacker cannot predict what response will be correct. Changing any bit in the chip input (challenge) or output (response) streams will cause the host system to see the pair as invalid.

Two classes of security model are common – fixed challenge-response pairs and variable challenges. If a fixed challenge-response pair is used, the attacker must know every challenge in advance. If the challenge is a variable that is either random or drawn from a huge set of possibilities, prediction is very difficult and no amount of monitoring is likely to succeed.

3.2. Brute force attack

Like the Man-in-the-middle attack, the bytes are captured as they cross the bus. They are then moved to a computer of some sort for analysis. Typically, the computer will try many possible values of secrets to see which, when combined with the recorded inputs, will create the recorded outputs. Due to the enormous size of the secret keys, this attack is not possible with CryptoAuthentication – even with special FPGA hardware built expressly for this purpose.

3.3. Replay attack

A replay attack is a close cousin to a Man-in-the-Middle attack. The bus is monitored, recorded, and is later played back to fool the microcontroller into believing a client side chip is present when it is not.

The usual method used to thwart the attack is called a diversified challenge. A random number is generated by the microcontroller, and sent to the CryptoAuthentication as the challenge. The MAC generated includes the random number, so each response is unique and recording/replaying the answers is useless.

It should be noted that a replay attack on a CryptoAuthentication chip is a good deal more involved than any standard interface. Logic analyzers are common for RS-232, USB, LPC and so forth, and many high end oscilloscopes will automatically devolve signal captures into bytes of data. The nature of the CryptoAuthentication chip one-wire interface defeats these instruments and forces the work to be done by hand.

The only way this attack succeeds is if the random number is not random. For example, calling the RAND function in C a number of times, but always starting with the same seed will always give the same sequence of pseudorandom numbers. Recording the sequence and replaying it will work. In fact, no purely software random number generator is practical; all random number generators that are even reasonably random have some hardware component.

If a replay attack is a concern for a specific situation, careful thought should go into the random number generator. If, at a minimum, some non-volatile memory is available the problem is solvable by preventing the sequence from starting over. If possible, a FIPS pseudo random number generator should be utilized.

3.4. System RAM monitoring attack

A program is inserted to watch the computer or microprocessor memory and provide periodic snapshots. Secrets in memory will be extracted from the snapshots. This attack is typically mounted on the host processor which may need to use the secrets to calculate the expected response from the AT88SA102S.

The principle reason for the AT88SA10HS chip is to defeat this attack. Clearly a MAC from the client AT88SA102S device must be checked by the host to evaluate whether the answer is correct, and if the match is against the result of a calculation performed in system memory. All the values must be present at some point in system memory. An AT88SA10HS chip will hold the secrets internally and perform the match on-board. Since the secret values in the message to the hash algorithm are never in RAM and the hash itself is done in hardware, no amount of monitoring will work.

If the use of an AT88SA10HS chip is impractical, then the conditions above should be restricted as far as possible:

- A microcontroller should be configured to restrict as far as possible the ability to read the internal programs or to insert new code. Most microcontrollers have EEPROM bits or fuses that will disable JTAG or test/programming interfaces and at least attempt to protect the internal memory resources.

4 High Level Security Models

- An RTOS should attempt a secure boot and should be configured to authenticate new code before it is allowed to run.
- The memory map for the microcontroller may be specified so that only the expected code will fit into the code segment.
- Code should be intentionally written to expect specific code at specific hard-coded addresses in memory and fail catastrophically if the code is moved.

What are NOT desirable are good transparent coding standards, such as a standard C *struct* with all the numbers neatly arrayed in memory. At the least, the numbers should be stored in a randomized or obscured fashion and used off the stack when needed.

If the key is scrambled in memory, reconstructed only when needed, and passed to the calling function on the stack, then a monitor program must catch the key in a snapshot of memory taken after it is generated but before the program overwrites the stack. If the all the key data is in a constant variable or a long-lived memory block, however, the monitor program has many clock cycles to catch the data in a snapshot.

3.5. Reverse engineering the microcontroller code attack

The host's program is extracted, modified to bypass the security checks, and reinserted, very possibly into many counterfeit devices. The common term for guarding against this attack is "platform integrity" protection.

The first line of defense is a bootloader in ROM that is difficult or impossible to read out and reproduce. Bootloaders are far more difficult to modify than user code and easier to protect. The bootloader should do a go/no-go check on the code in the microcontroller. One of the following models will present a way to do this with a CryptoAuthentication chip.

Secondly, a program should be compiled with all available optimization enabled. Reverse compilers are much, much better today than just a few years ago, but reverse engineering an optimized project of any reasonable size is still not an exercise for the amateur. The exercise can be made much more difficult by sprinkling a series of checks against the CryptoAuthentication chip throughout the code, forcing the attacker to find each and every one.

3.6. Hardware attack

The security chip is literally taken apart physically and the values read out.

Ultimately, no chip is perfectly invulnerable to this attack, but it requires by far the most expert people and the most expensive equipment. The defense is to keep the cost of the attack as high as possible, and the benefit of succeeding as low as possible.

The CryptoAuthentication chips have several features designed to make physical attack unproductive:

- An active shield; if a focused ion beam (FIB) is used to "burn" holes in the shield, the shield will disable the chip.
- Tamper detection circuitry that, if tripped, disable the chip and prevent clock attacks, voltage brown-out attacks, and other pin-based attacks.
- An internal clock that makes differential power analysis attacks and timing attacks more difficult by protecting clock edges from observation.
- The hardware key values are dispersed and obfuscated, not in a ROM.

Taken all together, the defenses make a physical attack very expensive.

As we go through the authentication security models, we'll touch again on these attacks.

4. Fields and programming

The following table, provided for each security model, details the fields available for inclusion in the message to the MAC in a CryptoAuthentication chip. The MAC command is of the form MAC (mode, KeyID, challenge). The mode parameter, which controls the values included in the message, is included as the last line of the table for convenience. In the AT88SA10HS and the AT88SA102S, the status fuses may be set and included as part of the message, or may be used to track other information. In the AT88SA100S the status fuses are never included in the input message to the MAC. An example table follows.

Table 1. Message fields

Value	Setting	Used in MAC	Notes
Challenge	Host selected	Always	Arbitrary and situational
Key	Static, protected	Always	One of the internal key values
User Secret	Static, protected	Usually	Burned at OEM or Subcontractor
Serial Number	Static	Maybe	Depends on design
Mode	Mac command variable	0x?0	Governs which fields are used

5. Authentication Security Models

5.1. Model #1: Low cost authentication

This is the simplest and least complex scheme. A challenge and the expected response are stored in some host as a pair. When the host wants to authenticate, the stored challenges is presented and the result checked against the expected response. Typically each host would include a different challenge-response pair. In some applications, the host has a long list of challenge-response pairs and may never use the same pair twice.
.

The following table lists the variables that can go into a MAC and the use for this model. Refer to the datasheet for sizes and types for a given chip.

Table 2. Low cost authentication

Value	Setting	Used in MAC	Notes
Challenge	Arbitrary	Always	Matches an expected response
Key	Always the same	Always	
Secret	Always the same	Always	
Serial Number	Fixed	No	
Mode	0x20		

6 High Level Security Models

The scheme can be used to inhibit software cloning. Fifty or a hundred checks can be sprinkled through the code on a microcontroller, each using some transitory value as a challenge with a hard-coded response. If, as an example, the value of a computed variable with function level scope is sent to the CryptoAuthentication library and the function fails with a catastrophic error if the response is incorrect, then the hacker must dig out each and every check in assembly code.

Since the challenge/response pair is fixed, a replay attack is likely for this model if the challenges are in any way predictable and can be known by the attacker. In many situations, however, this may not be a barrier to the business usefulness of the strategy.

In a medical equipment situation, for instance, there may be a consumable device that is authenticated by a physically large machine – and each machine has a different challenge response pair. Since the owner of the machine would be unlikely to send the machine to an eBay supplier of clone devices, there's no way for the cloner to know what response the clone would need to generate.

One clear advantage of this model is that no secret needs to be stored on the host at all and no calculation by the host is required. Pairs are stored at will, very possibly in many places in the code or in protected secondary storage.

The model is also useful when different host-client sets can have different challenge/response pair sets. Here a replay attack satisfies only one host, but a production run of tens of thousands of devices with different challenge-response pairs will force the exercise to be performed for every host. Very high-speed production of challenge/response pairs on a production line is not complex.

5.2. Model #2: SA-10HS for Platform Authentication

A variation of this scheme with an AT88SA10HS chip is frequently useful. The function of an AT88SA10HS chip is to check a MAC result returned by either the host microcontroller or a client AT88SC102S chip. If the code in a microcontroller is hashed by the boot loader (or other code verification element) and the resulting digest sent to the AT88SA10HS along with the expected response, that hash can be checked just as easily as the hash generated for a MAC on an AT88SA102S.

The distributor would know the correct response value for a specific program or code segment, computed using the secret values set in the AT88SA10HS on the target device. The code and a signature would ship as a unit, and at installation the code would be hashed into a digest used as the challenge to an AT88SA10HS HOST command. The signature shipped with the code is the expected digest for that specific AT88SA10HS configuration. If the AT88SA10HS fails to authenticate the code, it has been tampered with.

This model is similar to the first model with one important distinction. Systems in the field cannot be used to generate the expected signature for a modified code element since the response never leaves the AT88SA10HS chip. For a modestly priced device, it could be expected that an attacker might take one apart to permit generation of the signature (response) value for non-authenticated code.

5.3. Model #3: Standard Authentication

In this case the challenge to the CryptoAuthentication chip is a random number, defeating the replay attack:

Table 3. Standard authentication

Value	Setting	Used in MAC	Notes
Challenge	Random	Always	Matches a calculated response
Key	Always the same	Always	
Secret	Always the same	Always	
Serial Number	Fixed	No	
Mode	0x20		

As the value of the Challenge isn't the same each time, the Response is therefore different each time, and recording and replaying the sequence is useless. This model is both far more secure and somewhat more complex than the first two models since now the secret key value must be known – and protected – somewhere in the host system in order to calculate the expected response.

This is the primary use of an AT88SA10HS chip; the values are securely stored inside the AT88SA10HS and the digest is verified inside the chip. If the AT88SA10HS is not used, the secrets needed for the matching calculation must be stored inside the host and a RAM monitoring attack or reverse engineering the microprocessor code becomes viable. If the AT88SA10HS chip is used, the model is light-weight from a complexity point of view.

The cost of the model is the requirement to have two chips installed, and the need to submit a request to both chips for a single authentication. If the AT88SA10HS is not used, the SHA-256 algorithm will be required in the host program, as well.

5.4. Model #4: Diversified Authentication

In this model, the serial number adds another level of utility to the model above. Electronic serial numbers are often stored in standard nonvolatile memories, which are easily read by a hacker. In this security model the CryptoAuthentication device authenticates the value of the serial number, preventing a copied serial number from being accepted.

Table 4. Diversified authentication

Value	Setting	Used in MAC	Notes
Challenge	Random	Always	Matches an expected response
Key	Always the same	Always	
Secret	Always the same	Always	
Serial Number	Fixed	Yes	
Mode	0x60		

All the key and secret values are the same in all devices, but the serial number is queried by the host before the response is generated. The client chip serial number is used in both the host and client digest (Response) calculation. This model provides a mechanism for securely distinguishing one client from another, in the above models any client can be used with each host.

The host may very well have a list of permissible serial numbers, effectively building an electronic lock. Only those serial numbers are tested for a match, the use of the specific serial number is likely recorded, and the clients become inexpensive electronic access control devices.

A typical access control system can use this model. The host is a central server with many readers and a database of serial numbers, permissions, accesses, and so on. Security is better than the average card-based security system and the cost is lower.

Like model #3, the system benefits greatly from an AT88SA10HS if the host is a portable device and not a hardened server. Fully personalized client devices can be stored and distributed without significant risk since the host controls which serial numbers have any access rights.

5.5. Model #5: Diversified Key for Batteries

This is the primary use of an AT88SA100S, the protection of batteries. The AT88SA100S stores the authentication key in an SRAM, so some power source (or power storage element like a super-capacitor) must be available on the client.

Table 5. Diversified key for batteries

Value	Setting	Used in MAC	Notes
Challenge	Random	Always	Matches an expected response
Key	Never the same	Always	Derived from serial number
Secret	-	-	Not included in MAC on AT88SA100S
Serial Number	Fixed	No	Depends on design
Mode	0x60		

The key in this case should have been cryptographically derived from the serial number of the client chip during personalization. The host queries for that client's serial number, performs the same cryptographic operation, does the same MAC, and verifies a match. Every key on every chip is different, but reading the serial number has no benefit unless the cryptographic operation is also compromised.

This model has the security benefit that cracking one unit tells the attacker nothing about the other units in the production run – each client has a completely different key from every other client.

The phrase "cryptographic operation" is used because several methods are valid. The AT88SA10HS can be used, and the cryptographic operation in this case will be SHA-256. Or some other host cryptographic computation can be performed using an algorithm of the host's choosing.

This model is particularly attractive for hand-held devices and batteries. The AT88SA100S clears on a power down, so tampering with a battery ruins the battery, but the hand-held device need not store a large number of keys. Compromising the key in one battery does not tell anything about the next battery as each has a unique serial number and therefore a unique key.

Revision History

Doc. Rev.	Date	Comments
8666A	3/2009	Initial document release.

Headquarters

Atmel Corporation
2325 Orchard Parkway
San Jose, CA 95131
USA
Tel: 1(408) 441-0311
Fax: 1(408) 487-2600

International

Atmel Asia
Unit 1-5 & 16, 19/F
BEA Tower, Millennium City 5
418 Kwun Tong Road
Kwun Tong, Kowloon
Hong Kong
Tel: (852) 2245-6100
Fax: (852) 2722-1369

Atmel Europe
Le Krebs
8, Rue Jean-Pierre Timbaud
BP 309
78054 Saint-Quentin-en-
Yvelines Cedex
France
Tel: (33) 1-30-60-70-00
Fax: (33) 1-30-60-71-11

Atmel Japan
9F, Tonetsu Shinkawa Bldg.
1-24-8 Shinkawa
Chuo-ku, Tokyo 104-0033
Japan
Tel: (81) 3-3523-3551
Fax: (81) 3-3523-7581

Product Contact

Web Site
www.atmel.com

Technical Support
cryptoauthentication@atmel.com

Sales Contact
www.atmel.com/contacts

Literature Requests
www.atmel.com/literature

8666A–SMIC–03/09