APPLICATION NOTE 152

SHA iButton Secrets and Challenges

Several applications use the Dallas SHA iButton® tokens for eCash, vending, fare collection, and user authentication. They require the use of secrets to recognize authentic tokens and to secure data and monetary values against tampering. The security of an entire cryptographic system rests with its ability to properly generate and protect its secrets and challenges. This application note describes methods to generate world-class cryptographic secrets and challenges. It also describes the devices that can hold those values where they are safe from attack, and describes schemes where secrets never leave the protected silicon environment at any time and are never subject to compromise.

Introduction

Secure systems (eCash, vending, fare collection, user authentication) using the Dallas SHA iButton tokens require the use of secrets to recognize authentic tokens and to secure data and monetary values against tampering. These same systems also require the frequent generation of random challenges for authentication processes. The security of a cryptosystem depends on the strength and security of the secrets and challenges involved.

This App Note defines the requirements for secret and challenge generation and security, and discusses some of the features of the Dallas SHA iButton tokens that can help meet those requirements.

Some terms used herein are as follows: A **secret** is a numerical value known to the valid participants in a cryptosystem but unknown to anyone else. Knowledge of the secret value is absolute proof of authentication and originality.

A **MAC** (Message Authentication Code) is the result of a SHA hash function when some of the input data is kept secret. It is used as a way to secure the non-secret information against tampering and to authenticate the source of the information. In Dallas SHA iButton and IC devices, the result of the SHA function is a 20-byte long MAC. (A MAC is sometimes referred to as a signature, although this is arguably an inaccurate use of the term.)

A **challenge** is a numerical value, usually generated at random, issued to a token to be used as a basis for the token to prove its authenticity. By performing a cryptographic operation on the challenge using a secret value and returning the result, the token may prove that it knows the secret and, therefore, that it is authentic. Random values are used so that each authentication is unique and an attacker cannot use replay methods against the system.

Entropy is a measure of disorder and randomness. The term "random" does not always suggest true, un-knowable randomness without knowable bias.

Secret Generation

SHA device secrets are 64-bit binary numbers, which provide for 18,450,000,000,000,000,000 possible values. The strength of the secret lies in the time that it would take to perform a brute-force attack (an attack in which every secret is tried until one works). A good SHA secret will make use of the entire field of possible secrets, making a brute-force attack as difficult and time-consuming as possible.

Poor secrets are those that limit the scope of possible secret values. A secret that is an upper-case ASCII character string (ie: "RCIKLEHB") reduces the field to only 208,800,000,000 because there are far fewer letters in the alphabet

than there are binary values that can be made from an 8-bit character. Secrets that form words (ie: "MYSECRET") further reduce the field because there are far fewer combinations of letters that make valid words than there are of randomly selected letters. The very best secrets are randomly-generated binary numbers.

Of course, any secret that is created from public information (your dog's name, your phone number, etc.) is weakened due to the limited number of bits of public information from which to choose. Indeed, the scope of such a secret is equal to the number of items from which the secret was selected times the number of variations of that data that are possible —not a very large field at all.

Rule Number 1: Secrets should always be created using a random binary number source.

Random Numbers
The debate over the quality of random numbers and how they should be generated is perhaps one of the most heated in cryptography. Even the very definition of randomness is the topic of much lively discussion. For the sake of our purposes, we will consider a number cryptographically random if no one could have predicted, to any degree greater than the scope of the field, what number it would be. In other words, it is not the next step in a sequence, it does not have better odds of being any one number or range of numbers versus another, and it has no pattern of any kind (at least no pattern that any possible attacker could know).

Random numbers have qualities that are important. One quality that is critical to us in making secure secrets is called distribution. A random number is said to have good distribution if it has equal odds of being any number in the desired field. In other words, any random number between 1 and 100 (inclusive) would have exactly the same odds of being 1 or 25, or 78, or 100, or any other number in the range. If the random number generator has a tendency to favor odd numbers, or larger numbers, or numbers near the center of the range, then it is not a statistically good generator.

Pseudo-random numbers can be generated algorithmically or mathematically. These numbers are not truly random, but may appear to be random to the observer. In fact, these numbers have a pattern that will repeat at some point based on the size of the pool of stored history that it can draw from. Pseudo-random number generators often have excellent statistical qualities, but are a poor source of secrets because their output may be predictable. Indeed, anyone knowing the mechanism and any one output of the generator may be able to compute the next output exactly, or may be able to reduce the field of possible next outputs considerably.

The Dallas DS1963S SHA iButton has mechanisms that can generate statistically sound pseudo-random numbers, and has a very large data pool from which to draw in generating them. The pattern will not repeat in billions upon billions of generated bytes—although it will indeed repeat eventually. Alone, it is a weak source of data for making secrets or challenges. However, when the device is supplied truly random input data, or entropy, it becomes an excellent random secret generating tool.

Entropy (true randomness) can be found in many places. The timing of the keystrokes of a human operator, the timing of the arrival of network packets, the synchronism of the power line frequency to a microcontroller crystal clock may all be sources of real entropy. These sources may have truly random components, but usually also have poor statistical qualities. For example, the timing of human keystrokes contains both a predictable component (the key strokes are more-or-less regular, and fall into a reasonable range of timings) and a random component (the exact split-second timing of each stroke). The solution is to collect real-world entropy and process it through an otherwise pseudo-random system that will then generate output data with good statistical qualities. In effect, the pseudo-random system serves to distribute the real entropy throughout the field. The end-result is an excellent cryptographic random number to use as a secret or challenge.

Rule Number 2: The random numbers used to make secrets or challenges must have good statistical as well as cryptographic qualities.

Protecting Secrets
Once the secret is generated its value is lost if it becomes public. In fact, some would say that the value of the secret is lost if it even has the potential to become public. Keeping secrets *secret* can be the largest single problem, and the

biggest security issue, in any cryptosystem.

Because human beings are always the weak-link when it comes to secrets, the best way to maintain the security of the secret is to generate and maintain the secret inside of cryptographic devices such that it is never known or accessible to any human at all. The Dallas SHA devices allow secrets to be assembled entirely inside each device, and beyond the view of any human or external machine.

The Dallas SHA devices maintain secrets in special memory cells that cannot be read or interrogated from the outside world. The secrets can be used to perform the SHA cryptographic functions, but cannot be observed directly. Because the SHA function cannot be reversed, the secret always remains secure inside the device.

Rule Number 3: Secrets should always be hidden inside secure devices and never exposed.

One Secret, Many Secrets
The ultimate secret is one that is known to no one. But how can such a secret be used if no one knows it? Secret sharing is a technique whereby the true secret is in fact the combination of several partial secrets. The true secret can be recreated only by bringing together all the partial secrets in the proper sequence. The holders of the partial secrets do not know the real secret, and knowing any one partial secret reveals nothing about the real secret. (Picture a vault with several locks on it. All the keys must come together to open the vault, and holding any one key, or even all the keys but one, is of no use to a thief.)

Dallas SHA devices provide an internal mechanism for secret sharing. Each SHA device is able to build a new secret from two or more partial secrets. To do this, it uses the current secret, and a new partial secret provided from the outside, to compute a MAC. A portion of the MAC then replaces the original secret. This allows the real secret to be computed entirely inside each device, one partial secret at a time, and never be exposed to the outside world at all. The real secret is never exposed and cannot be compromised.

The partial secrets still require special handling, of course, because obtaining all of them can still reveal the true secret, but the partial secrets can be distributed throughout the deployment process and odds of all of them being compromised are much lower.

Rule Number 4: Always build the master secrets inside the tokens using multiple partial secrets.

Challenges
SHA devices are authenticated by a method called challenge-and-response. This is a simple process where a token proves that it knows a secret without ever revealing the secret. To make a token prove that it is authentic, the host generates a random number (challenge) and sends it to the token. The token generates a MAC of the challenge using the secret, and returns the MAC to the host. The host performs the same MAC internally (because it also knows the secret) and, if the response from the token is a match, the token has proven itself authentic. Only a token that knows the secret could have generated a matching MAC.

The reason why the challenge must be random is to make it impossible for an attacker to know which challenge value will be presented next. This prevents him from being able to use a response obtained by eavesdropping on other valid token conversations.

The rules that apply for the generation of secrets apply the same to the generation of challenges. The challenge must be truly random (wholly unpredictable) and must make use of the entire field of possible challenge values—statistical as well as cryptographic qualities.

Rule Number 5: Just as with secrets, the random numbers used to make challenges must have good statistical as well as cryptographic qualities.

Using the Dallas DS1963S to Generate Secrets and Challenges
The output of the SHA function is essentially pseudo-random to an observer who does not know the input to the function. The pool from which the SHA function builds its output is quite large. These qualities make the Dallas SHA devices excellent pseudo-random number generators. Minor variations in the input data cause large variations in the resulting output (called diffusion). This makes these devices excellent post-processors of true entropy, making random numbers of exceptional statistical and cryptographic quality.

Here is an example of a scheme for generating random challenges using the DS1963S SHA iButton in a vending application. Of course, the same techniques apply in secret generation schemes:

> A microcontroller-based vending machine requires random challenges in order to authenticate arriving iButton tokens. The microcontroller has a DS1963S iButton onboard to serve as a SHA coprocessor. The microcontroller is handling the arrivals of iButton vending tokens, and at the same time handling serial communications (polling) from the vending machine electronics. The firmware runs a 16-bit timer at all times from the fastest clock available. Whenever a message arrives from the serial vending machine bus, the value of the timer is captured and added to a 16-bit location in memory. Likewise, whenever an iButton token arrives, the time value is captured and added to another 16-bit location. Because the arrival of iButton tokens is asynchronous to the system, as are the vending machine serial communications, these values contain components of real entropy. When a random challenge value is required, the system writes the stored 16-bit timer sums to the DS1963S coprocessor and performs a ‘generate challenge—command. The microcontroller then reads the resulting MAC from the DS1963S and uses some portion of it as the random challenge value. It also adds some of the bytes of this result to the two memory locations, further extending the effect (or diffusion) of the entropy into future random values.
>
> The DS1963S ‘generate challenge—command uses a selected secret. This could be one of the system working secrets, or it could be a random value injected when the device is configured. Each time the ‘generate challenge—command is used, the DS1963S increments an internal counter which is also a part of the input to the SHA mechanism when this function is used. This means that the resulting MAC will be different every time, and will vary in pseudo-random fashion, even if the input to the device is held constant.

The example above shows how a small amount of real-world entropy can be processed by the DS1963S SHA iButton to create a random challenge value with strong statistical and cryptographic qualities.

Other Sources of Entropy
Microcontroller-based systems need to seek out sources of true randomness wherever they can find it. It is important that these sources be mechanisms that an attacker could not attempt to control or bias in any way. The arrival of an iButton device could be controlled using electronic switches, for example. While this remains a viable source of entropy, it should not be the only one.

Sources of entropy (true randomness) might include the following:

- Asynchronous external events (serial communications, power line zero crossing, etc.)
- Human-activated external events (keystrokes, button presses, token arrival, etc.)
- Timing of external devices (the width of a pulse from an iButton token, the speed of a bar code target, time for an LCD display to issue READY status after a character write, bounce of a mechanical switch or relay contact, etc.)
- Special random value generation hardware (noise sources, cross-coupled oscillators, etc.)

Rule Number 6: Provide as much real world entropy as possible, and always use at least two independent sources.

Unique Token Secrets

When a critical system secret is to be stored in tokens that will be distributed in large numbers into the hands of un-trusted individuals, there is always a concern that the secret will somehow become public, which would be catastrophic in a large implementation. Despite the very best methods to protect the secret from physical and electronic attacks, it is always desirable to minimize the potential impact of a compromised token. This can be done by making the secret in each token unique to that token. Of course, the host must be able to compute this unique secret given the identity of the token, or it would not be able to authenticate it.

A method for accomplishing unique token secrets is to use the SHA function to generate a MAC of the token ID and perhaps some other token or owner identity data (like a PIN or password), using the system master secret. The resulting MAC (or a portion thereof) is used as the secret for that token. An attacker obtaining this unique token secret will have no knowledge of the master secret (because SHA cannot be reversed to reveal its input data).

In a system that uses unique token secrets, the host will have to obtain the data required to regenerate the unique token secret before it can use that secret to authenticate the token. This will require that the host read the token identity (serial number) and request any additional required information (such as PIN or password) from the user. This data is then used, along with the master secret known in the co-processor, to regenerate the unique secret for the token.

The DS1963S SHA iButton, when used as a coprocessor in the host system, can perform this function entirely inside the device and thereby never expose the master secret, the unique token secret, or the results of the MAC to the outside world.

Rule Number 7: Use the SHA tools available to make token secrets unique and thereby protect the critical system master secret.

Conclusion

It is easy to see that the security of an entire cryptosystem rests with its ability to properly generate and protect its secrets and challenges. This Application Note has described methods to generate world-class cryptographic secrets and challenges, and devices that can hold those values where they are safe from attack. Indeed, we have described schemes in which secrets never leave the protected silicon environment at any time and are never subject to compromise.

When cryptosystems fail, the cause is almost always due to failure in the implementation and not of the cryptography itself. Poor secret generation, badly generated random numbers, weak challenges, and failure of secret security are usually the culprits. Attention to the details of secret generation and security are critical to a sound cryptosystem implementation.

More Information

DS1963S: QuickView -- Full (PDF) Data Sheet -- Free Samples