Secure Computer Systems
A novel attack on the DS1963S iButton

Ronald Huizer (r.huizer@xs4all.nl)
NSU ID: N01265319
Nova Southeastern University

Abstract

We will present a novel attack on leaking secret data from the DS1963S iButton, which is reduces the complexity of leaking secret data from 2^{64} to 2^{11} attempts.

Introduction

The DS1963S iButton is lightweight button-shaped SHA-1 processor that is used as an electronic cash system, and as an access token for authentication. The system is used in practice for micro-payments in several countries, such as Istanbul, where it is used as part of the AKBIL system to pay for public transportation and Brazil, where they can be used to pay at parking meters. General information on iButtons can be found in *The book of iButton standards* (1997) and more specific information on the DS1963S can be found in *DS1963S SHA iButton* (2002).

The security model of the DS1963S is based on 8 64-bit secrets which are used as keys in the SHA-1 HMAC calculations the button performs for signing, verification, and authentication. The security of the system is largely dependent on the inability to read the secrets, as being able to do so would open the door for button duplication, and emulation. The button takes care to prevent all read access to the secret memory, and also possible leakage of the secret memory to other memory areas which can be read.

The DS1963S secret memory has been successfully dumped by Brandt (2011) using a fault attack. His method was presented at 27C3 and involves having to disassemble the iButton in order to determine the secrets. Although very interesting, his method is destructive and requires specialized hardware and electrical engineering knowledge.

We will introduce and implement a novel software-only attack on the DS1963S iButton that is able to retrieve secrets memory used in the SHA-1 HMAC functions of the device in 2^{11} attempts in the worst case. It therefore significantly improves on brute-force methods on secrets, which would need 2^{64} attempts in the worst case.

We have earlier hypothesized the existence of this attack during our presentation at Infiltrate 2011 – see Huizer (2011). Although this observation proved to be insufficient by itself, it is used in our current attack, combined with another design issue we have found and which at the time of writing has not been published.

Architecture

Prior to describing the attack on the iButton, we will present a quick overview of the architecture as is relevant to the attack. A more thorough description of the architecture can be found in *DS1963S SHA iButton* (2002).

Memory

The DS1963S has several different memory pages of 256-bit each. The first 8 pages are regular data memory. These pages can be referenced using page #0 through 7, and the same pages can be references as page #8 through 15 in order to automatically increment the associated write cycle counter.

Page 16 and 17 are used to store 8 secrets each of which 64-bit in size. Each regular data page number is associated with both a write cycle number and a secret number. For example, page #5 would use write-cycle counter #5 and secret #5 in operations where either of the two is applicable.

Further pages include counter memory, secret write cycle counters, the scratchpad and a PRNG counter. Except for the scratchpad, which resides on page #18 and will be described later, the other pages are outside the scope of this article.

Memory addressing

The DS1963S uses two 8-bit *target address* (TA1 and TA2) registers, and an *ending address* register (E/S which contains both the ending address and several status bits we will not discuss), which are used differently depending on the operation performed, but generally denote the start and end addresses in memory. Although pages are used to describe the memory model, the addressing scheme itself does not explicitly use page number, but rather view the memory space as a flat byte granular sequence of addresses. For example, as pages are 256-bit in size, page #5 would start at address 0xA0.

The target address registers are often described by target address byte indices, where T0 is the lowest significant byte of TA1, T7 the highest significant byte of TA1, T8 the lowest significant byte of TA2 and T15 the highest significant byte of TA2. The lowest significant byte of E/S is E0, and the 5th lowest significant byte of E/S is E4. The other bytes in E/S are flag data.

Scratchpad

The 256-bit scratchpad serves as temporary latch memory to copy data from the bus master to the iButton and vice versa. When retrieving data from and writing data to memory pages, the scratchpad is used as an in between buffer.

Readability of the scratchpad is governed by the HIDE flag: when set, attempting to read the scratchpad will result in 1-bits, and when unset scratchpad content can be read. As the scratchpad is used by the internal HMAC engine, the HIDE flag is used to ensure no security critical information can be leaked through the scratchpad buffer.

HIDE flag

The DS1963S security model is based on the HIDE flag, which governs whether the scratchpad can be read or not, and whether secret pages can be written to or not. The HIDE flag is set implicitly by various SHA related operations, and set explicitly when a power-on reset is performed. A power-on reset is normally performed by breaking the connection between the iButton and the socket. This behavior can be emulated for 1-wire RS232 connected devices by pulling the RTS and DTR lines low for a certain amount of time[1].

Operations

The DS1963S offers a wide variety of operations, but we will limit our discussion to the operations relevant to our attack.

Write scratchpad (0x0F). The write scratchpad command has two modi operandi, depending on whether the HIDE flag is set or not. When the HIDE flag is unset, the bus master can use this command by sending the opcode, a 2-byte target address, and the data to be written to scratchpad memory. T4:T0 specifies the offset within scratchpad memory the data is written to, and E4:E0 specifies the end offset. The target and ending addresses are limited to the address range [0, 0x1FF].

When the HIDE flag is set, the operation is meant to update the target and ending address registers to prepare them for writing data to secret memory. As such, the target and ending addresses are limited to the secret memory range [0x200, 0x23F]. Note that this operation does not modify the scratchpad or any memory, but just updates TA1, TA2 and E/S.

Read scratchpad (0xAA). The read scratchpad command is used by the bus master to read the TA1, TA2, and E/S bytes, as well as the scratchpad data from the scratchpad. In case the HIDE flag is set, the data read will be logical 1-bits.

[1]The implementation of the attack uses a 1 second interval for this.

Copy scratchpad (0x55). The copy scratchpad command is used to copy data from the scratchpad to a memory page. The command expects TA1, TA2, and E/S to be sent by the bus master, and these should be equal to the values of the corresponding DS1963S registers. When the HIDE flag is unset, addresses [0, 0x1FF] can be written, and when the HIDE flag is set addresses [0x200, 0x23F] can be written.

Erase scratchpad (0xC3). The erase scratchpad command will clear the HIDE flag and fill the scratchpad with 1-bits.

Compute SHA (0x33) – Sign data page (0xC3). The compute SHA function allows several different SHA co-processor functions to be called. Our attack only uses one: the sign data page function. This function will calculate a HMAC over page #0 or #8 using secret #0, and 15-bytes of scratchpad data.

Complexity reduction

The first security relevant issue on the DS1963S is the granularity of writes to secret pages. As we have seen in the architectural overview, it is possible to write raw secret data to secret pages by writing data to the scratchpad using the *write scratchpad* command, setting the HIDE flag, and using the *write scratchpad* command to set TA1, TA2, and ES to secret memory, and finally writing the scratchpad into secret memory using the *copy scratchpad* command.

If we assume we can fully control TA1, TA2, and ES at this point, we could attempt to partially overwrite secret data, rather than overwriting a full secret at a time. This would result in a complexity reduction attack.

For instance, consider we are using the *sign data page* SHA function, which calculates the HMAC over the secret, data of a selected memory page, and 15 bytes of scratchpad data. If we set the data of the selected memory page and the 15 bytes of scratchpad data to 0-bytes, the only variable is the secret page. First of all, we would calculate the signature over this zero byte data, and store the resulting signature. Next, we would write the first byte of secret memory using our idea to modify only the first byte of the secret, and leave the rest alone. Then we recalculate the signature and see if it matches the original we found. If it does not, we increment the byte we have written, otherwise we have found the right secret data byte. We repeat this process on a byte by byte basis until the whole secret has been recovered. This would reduce the complexity of calculating the secret from 2^{64} to 2^{11}.

Protocol flaw

The DS1963S avoids the complexity reduction attack by allowing the *write scratchpad* command to only update TA1, TA2 and ES in such a way that full secrets are written at a time when the HIDE flag is set. In the state machine diagrams in *DS1963S SHA iButton* (2002) this is outlined by the T2:T0 = 0, 0, 0 and E4:E0 = T4, T3, 1, 1, 1 entry. This means security against this attack will come down to the ability or inability to update TA1, TA2, and ES in such a way that they refer to secret memory while not referencing a full secret.

However, the state diagram contains an interesting state, as the bus master transmits TA1, and TA2 prior to checking whether the HIDE flag is set, and verifying whether memory is data memory or secret memory. This means that when the HIDE flag is unset, TA1 and TA2 can be updated to point into secret memory (regardless of the HIDE flag), and the security modifications T2:T0 = 0, 0, 0 and E4:E0 = T4, T3, 1, 1, 1 will never be executed. As a reset to set the HIDE flag *preserves* the TA1, TA2 and ES registers, we have a way to set TA1 and TA2 to refer to secret memory a byte at a time.

Algorithm 1 Partial secret overwrite.

1. Set TA1, and TA2 using the *write scratchpad* command to refer to a byte within a secret.
2. Power-on reset the iButton to set the HIDE flag.
3. Use the *copy data* command to selectively overwrite a byte of secret memory.

The attack

The full attack is outlined below, and combines the protocol flaw with the complexity reduction attack.

Algorithm 2 Full attack against secret #0.

1. Use the *write scratchpad* command to fill the scratchpad with 0-bytes.
2. Use the *copy data* command to copy the scratchpad zero bytes into memory page #0.
3. Use the *sign data page* SHA function to calculate a HMAC over page #0, 15 bytes of scratchpad, and secret #0.
4. Read the signature from the scratchpad using the *read scratchpad* command, and store this as the target signature.
5. Set the guessed secret to 0 bytes, and the secret index n to 0.
6. Overwrite the secret #0 byte at index n with the next value using Algorithm 1.
7. Calculate the signature using *sign data page*. If the signature matches the target signature, we have the correct byte and increment n. Otherwise we increment the guessed secret byte at index n.
8. If all secret bytes have been found, we're done and have the secret, otherwise we go to step 6.

Demonstration

This paper is bundled with the source code, and two Linux *typescript* files of an example run against a physical dongle which can be replayed using the command: *scriptreplay -t example.timing example.script*

Conclusion

We have demonstrated a practical complexity reduction attack against the iButton DS1963S which reduces the complexity of leaking secret data from 2^{64} to 2^{11}. As this attack uses the *sign data page* SHA function, it is limited to data page 0 only. Further research is needed to see if this attack can be extended to other secrets by using different SHA functions.

References

The book of ibutton standards. (1997, December 8). Dallas Semiconductor. Retrieved from `http://pdfserv.maxim-ic.com/en/an/AN937.pdf`

Brandt, C. (2011, April 23). Hacking ibuttons. Presented at the 27th Chaos Communication Congress. Retrieved from `http://cribert.freeforge.net/27c3/ibsec_eh.pdf`

Ds1963s sha ibutton. (2002, February 2). Dallas Semiconductor. Retrieved from `http://pdf1.alldatasheet.com/datasheet-pdf/view/94286/DALLAS/DS1963S.html`

Huizer, R. (2011, April 16). Don't give credit: Hacking arcade machines. Presented at Infiltrate 2011. Retrieved from `http://immunityinc.com/infiltrate/archives/Arcade_Attacks.pdf`