Expanding on the DS1963S iButton attack

Ronald Huizer (r.huizer@xs4all.nl)
Nova Southeastern University

Abstract

In previous work we have shown a novel attack against the DS1963S iButton, which is reduces the complexity of leaking a single specific secret from 2^{64} to 2^{11} attempts. In this paper we will extend on this work, generalize the attack against all eight secrets that the DS1963S uses, and allow the attack to be supplemented by computing resources to reduce the time necessary to recover the secrets.

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 places, 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 eight 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.

In previous work, all eight DS1963S secrets have been compromised through a hardware based fault attack by Brandt (2011), and we have compromised one out of eight secrets though a software attack, as outlined in Huizer (2012). In this paper we extend on our previous work, and we present a software only attack on all eight secrets.

Architecture

Prior to describing the attack on the DS1963S, 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 layout

The DS1963S has 4096 bits of NVRAM used as data memory, and additional special purpose memory. Memory is byte addressable and divided in pages, each 32 bytes in size.

Data memory uses the first 16 pages. They can be freely used for both read and write access, regardless of the mode of operation. Each page of data memory is associated with a particular secret, and a particular write-cycle counter. Table 1 shows the layout of data memory, the secret and write-cycle counter each page is associated with, and whether a write to said page will increment the associated write-cycle counter.

Secret memory uses page 16 and 17, to store four 64-bit secrets each. They can be written indirectly through the commands *Compute First Secret* and *Compute Next Secret*, or directly by writing data to the scratchpad, setting the HIDE flag through a power on reset, and then copying the scratchpad to secret memory. They cannot be read directly, as the security of the DS1963S is based on them. Each secret has an associated write-cycle counter for that secret. These counters are distinct from the data page write-cycle counters. Table 2 shows the layout of the secrets in memory.

The scratchpad on page 18 serves as temporary latch memory to copy data from the bus master to the DS1963S 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.

Write-cycle counters are stored on page 19 and 20. Each time a write is made anywhere in the memory area associated with that counter, they are incremented. Note that data pages 0 through 7 do not increment any write cycle counter, but do use an associated write-cycle counter in case one is expected. Table 2 shows the layout of the write-cycle counters in memory. According to the specification, the counters do not wrap, but we have not tested this ourselves.

The PRNG counter is stored at the beginning of page 21. It cannot be written to, but it is incremented every time the DS1963S performs a SHA1 operation. It is unclear whether this counter can wrap or not.

Page #	Address Range	Data Memory Secret #	Write-cycle Counter #	Write-cycle Increment
0	0000-001F	0	0	No
1	0020-003F	1	1	No
2	0040-005F	2	2	No
3	0060-007F	3	3	No
4	0080-009F	4	4	No
5	00A0-00BF	5	5	No
6	00C0-00DF	6	6	No
7	00E0-00FF	7	7	No
8	0100-011F	0	0	Yes
9	0120-013F	1	1	Yes
10	0140-015F	2	2	Yes
11	0160-017F	3	3	Yes
12	0180-019F	4	4	Yes
13	01A0-01BF	5	5	Yes
14	01C0-01DF	6	6	Yes
15	01E0-01FF	7	7	Yes

Table 1: Data memory map of the DS1963S, as per *DS1963S SHA iButton* (2002).

Memory addressing

The DS1963S uses two 8-bit *target address registers*: *TA1*, storing the low 8-bits of the address, and *TA2*, storing the high 8-bits of the address. It also uses 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 numbers, but rather views the memory space as a byte granular sequence of addresses.

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.

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-1 operations, and set explicitly when a power-on reset is performed. A power-on reset is normally performed by breaking the connection between

Page #	Address Range	Description
16	0200-0207	Secret 0
	0208-020F	Secret 1
	0210-0217	Secret 2
	0218-021F	Secret 3
17	0220-0227	Secret 4
	0228-022F	Secret 5
	0230-0237	Secret 6
	0238-023F	Secret 7
18	0240-025F	Scratchpad
19	0260-0263	Write cycle counter 0 (counts for page 8)
	0264-0267	Write cycle counter 1 (counts for page 9)
	0268-026B	Write cycle counter 2 (counts for page 10)
	026C-026F	Write cycle counter 3 (counts for page 11)
	0270-0273	Write cycle counter 4 (counts for page 12)
	0274-0277	Write cycle counter 5 (counts for page 13)
	0278-027B	Write cycle counter 6 (counts for page 14)
	027C-027F	Write cycle counter 7 (counts for page 15)
20	0280-0283	Write cycle counter 8 (counts for secret 0)
	0284-0287	Write cycle counter 9 (counts for secret 1)
	0288-028B	Write cycle counter 10 (counts for secret 2)
	028C-028F	Write cycle counter 11 (counts for secret 3)
	0290-0293	Write cycle counter 12 (counts for secret 4)
	0294-0297	Write cycle counter 13 (counts for secret 5)
	0298-029B	Write cycle counter 14 (counts for secret 6)
	029C-029F	Write cycle counter 15 (counts for secret 7)
21	02A0-02A3	PRNG Counter (counts SHA1 operations)

Table 2: Non-data memory map of the DS1963S, as per *DS1963S SHA iButton* (2002).

the DS1963S 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.

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) is used by the bus master to set the *target address* registers and write data into the scratchpad of the DS1963S. The operation 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. This update of the *target address* can later be used with the *copy scratchpad* operation to save the scratchpad content to a secret.

The flow chart for the *write scratchpad* operation is given in Figure 1. Two key observations have been colored in green: regardless of the flow, the *target address registers* will always be read from the bus master and latched into registers *TA1* and *TA2.* We can also see that only when the *HIDE* flag is set, the *target address* and *ending address* are ensured to be aligned to span one or more full secrets.

Read scratchpad (0xAA) 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.

Copy scratchpad (0x55) 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, data addresses [0, 0x1FF] can be written, and when the HIDE flag is set, the secret addresses [0x200, 0x23F] can be written.

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

Read Authenticated Page (0xA5) is used to read a full or partial page, and a SHA-1 based HMAC over the 32-bytes of data in the page, the write-cycle counter of the page, the secret associated with the page, the page number, and the serial number of the device.

It should be noted that the write-cycle counter of the secret used is not part of the SHA-1 calculation. If it had been, this attack would not have been possible, as the write-cycle for our target secret is incremented every time we write to the secret, thus changing the resulting HMAC even if none of the other parameters change.

The SHA-1 input for *read authenticated page* is outlined in Table 3.

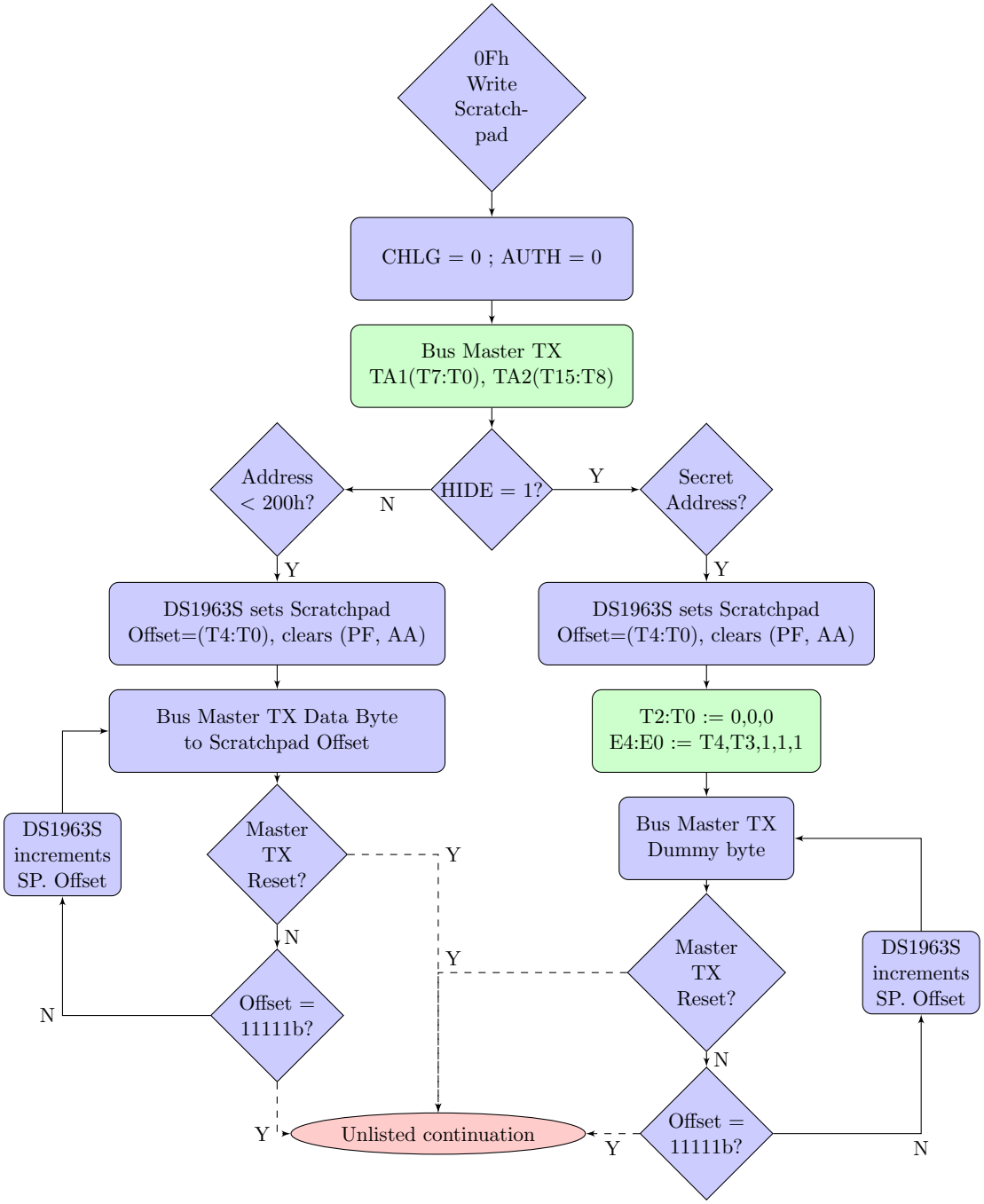

Figure 1. Write scratchpad flow chart, as per *DS1963S SHA iButton* (2002).

I[0] = SS[0]	I[1] = SS[1]	I[2] = SS[2]	I[3] = SS[3]
I[4] = PP[0]	I[5] = PP[1]	I[6] = PP[2]	I[7] = PP[3]
I[8] = PP[4]	I[9] = PP[5]	I[10] = PP[6]	I[11] = PP[7]
I[12] = PP[8]	I[13] = PP[9]	I[14] = PP[10]	I[15] = PP[11]
I[16] = PP[12]	I[17] = PP[13]	I[18] = PP[14]	I[19] = PP[15]
I[20] = PP[16]	I[21] = PP[17]	I[22] = PP[18]	I[23] = PP[19]
I[24] = PP[20]	I[25] = PP[21]	I[26] = PP[22]	I[27] = PP[23]
I[28] = PP[24]	I[29] = PP[25]	I[30] = PP[26]	I[31] = PP[27]
I[32] = PP[28]	I[33] = PP[29]	I[34] = PP[30]	I[35] = PP[31]
I[36] = CC[0]	I[37] = CC[1]	I[38] = CC[2]	I[39] = CC[3]
I[40] = MP	I[41] = FAMC	I[42] = SN[0]	I[43] = SN[1]
I[44] = SN[2]	I[45] = SN[3]	I[46] = SN[4]	I[47] = SN[5]
I[48] = SS[4]	I[49] = SS[5]	I[50] = SS[6]	I[51] = SS[7]
I[52] = SP[20]	I[53] = SP[21]	I[54] = SP[22]	I[55] = 0x80
I[56] = 0	I[57] = 0	I[58] = 0	I[59] = 0
I[60] = 0	I[61] = 0	I[62] = 1	I[62] = 0xB8

I[64]	A 64-byte input array to the SHA-1 engine.
SS[8]	The 8-byte secret used for this data page.
CC[4]	The 4-byte write-cycle counter for this data page, in little-endian form.
PP[32]	The 32-byte data page that is being read.
FAMC	The family code. For the DS1963S this is always 0x18.
MP	MP[7] = Control bit M MP[6] = Control bit X MP[5:4] = 0 MP[3:0] = Number of the data page being read.
SN[6]	The 6-byte serial number for this DS1963S device.
SP[32]	The 32-byte scratchpad.

Table 3: SHA-1 input for *Read Authenticated Page,* as per *DS1963S SHA iButton* (2002).

Flaw description

In Huizer (2012) we have presented the key observations that allow the security model of the DS1963S to be broken. We will describe them again here, for the sake of completeness.

The first security relevant issue on the DS1963S is the granularity of writes to secret pages. As we have discussed, it is possible to write raw secret data to secret pages by writing data to the scratchpad using the *write scratchpad* operation, setting the *HIDE* flag, and using the *write scratchpad* operation to set *TA1*, *TA2*, and *ES* to secret memory, and finally writing the scratchpad into secret memory using the *copy scratchpad* operation. Such partial writes are disallowed by the *write scratchpad* operation, as we can see in the flow chart in Figure 1. The operation will set the *target address* to a multiple of 8 by setting the lowest three bits to zero, and ensure the *ending address* spans the whole secret by setting the lowest

three bits to one.

It should be noted however, that *TA1* and *TA2* are latched in regardless of the flow chart of the *write scratchpad* operation, and only zeroed out in the specific case where the *HIDE* flag is set, and the *target address* refers to secret memory. Moreover, *TA1* and *TA2* persist past a power-on-reset, which means that it is possible to set them to any value we want when the *HIDE* flag is cleared, including misaligned secret memory, and then set the *HIDE* flag through a power-on-reset.

Previously we have used the *sign data page* SHA-1 operation to calculate a HMAC over known data as a baseline, and then overwrite the secret one byte at a time, iterating through all possible values of this byte, and signing a data page with them. If the resulting HMAC is equal to the baseline HMAC we have found the right byte. This reduces the complexity of determining a secret from 2^{64} to 2^{11}.

However, the *sign data page* operation can only be used on data pages 0 and 8, which are associated with secret 0. We have updated our approach to use the more convoluted *read authenticated data* operation, which can be used on any of the data pages, and therefore any of the associated 8 secrets. We have also optimized the approach by offloading part of the cracking of the SHA-1 secrets to the CPU, reducing the length of the attack by several hours.

Methodology

Below we present a breakdown of the methodology used to compromise the security model of the DS1963S. Algorithm 1 shows the method used to partially overwrite a secret with user specified data. It relies on the design flaw we've discussed in the previous section.

Algorithm 2 describes the full attack against an arbitrary secret, using the partial overwrite as a basis. It should be noted that the attack irreversibly changes the write-cycle counters of the secrets, and therefore may not be fully transparent based on the context of the software that uses the DS1963S. However, as the original write-cycle counters can be read and stored prior to launching the attack, it is possible to create an exact data dump of the state of a DS1963S button, and configure a custom[1] button with said data.

Algorithm 1 Partial secret overwrite.

1. Use the *erase scratchpad* operation to set the scratchpad to 1-bits, and clear the *HIDE* flag.
2. Set the data to be written using the *write scratchpad* operation. *TA1* and *TA2* are chosen to refer to a byte within a secret.
3. Power-on reset the DS1963S to set the *HIDE* flag.
4. Use the *copy data* operation to selectively overwrite a byte of secret memory.

[1] This will require custom hardware, as although on a fresh DS1963S button we can initialize the write-cycle counters of the secrets however we want, but we cannot change the serial number of the button, as it is unique and laser etched for every dongle.

Algorithm 2 Full attack against an arbitrary secret.

1. Use the *erase scratchpad* operation to fill the scratchpad with 0xFF-bytes and clear the *HIDE* flag.
2. Use the *read authenticated page* operation to calculate the HMAC over the serial, the page number, the data page and its write-cycle counter, and the secret.
3. Read the signature from the scratchpad using the *read scratchpad* operation, and store this as the target signature.
4. Initialize the guessed secret with0-bytes, and the secret index n to 0.
5. Overwrite the secret byte at index n with the guessed secret byte at index n using Algorithm 1.
6. Use the *erase scratchpad* operation to ensure the scratchpad is filled with a baseline of 0xFF bytes.
7. Calculate the signature using *read authenticated 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.

Efficiency improvements

The attack we have presented so far will sequentially iterate through all the individual bytes in all the secrets, and for each iteration have to wait for the Power-on-Reset to stabilize. This is currently hard-coded to be 1 second, but a more dynamic approach should be possible, as according to Brandt (2012) this time-frame is largely dependent on the ambient temperature. To improve on the attack, we observe that we do not have to leak out all key bytes for a given secret. Instead, we can choose to leak 4 bytes through our side-channel attack, and brute-force the remainder in software. This brute-force attack can be performed concurrently to leaking the key bytes for another secret, allowing for parallellization of the attack. It can also be implemented on custom hardware such as FPGAs, for even better results.

In order to do this, we have implemented part of the DS1963S SHA-1 signing scheme in software, such that we can emulate the *Read Authenticated Page* command. As this command is dependent on the serial number, the data page associated with the secret we attack, the scratchpad, and the write-cycle counter of that data page, we retrieve this information from the hardware DS1963S, and parametrize our software implementation using it. For each secret, we leak 4 bytes of key material through the side-channel attack, and brute-force the remaining 4 bytes on a dedicated thread. This decreased the duration of the attack by several hours. This approach is summarized in Algorithm 3.

Algorithm 3 Optimized attack against all secrets.

1. Read data pages 0 through 7, their respective write-cycle counters, and the DS1963S serial number.
2. Initialize eight software emulated DS1963S devices, paremetrized by the data from the previous step.
3. Determine the first 4 bytes of each secret using Algorithm 2.
4. Brute-force the remaining 4 bytes using the emulated DS1963S device for that secret on a separate thread. This step can be performed concurrently with step 3.

Demonstration

Figure 2 shows our utility while working on recovering the DS1968S secrets, which have been initialized with random data. For every secret, the key bytes recovered so far are shown in the middle column, and the target HMAC is shown in the right column. The progress bars update in two phases, the first phase, shown in cyan, shows the progress of our side-channel attack recovering the first four bytes of a secret. The second phase, shown in green, shows the progress of the brute-force attack in the remaining four key bytes. Figure 3 shows the completed attack after a run time of two hours.

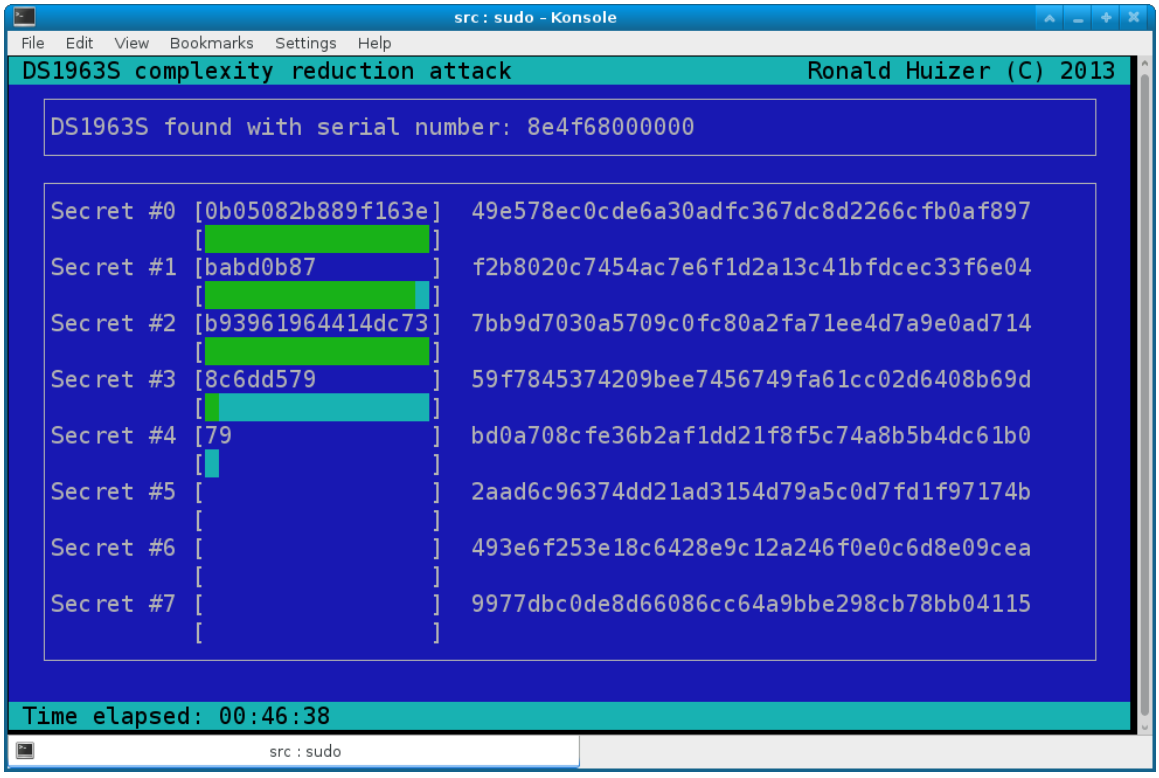

Figure 2. The side-channel attack in progress.

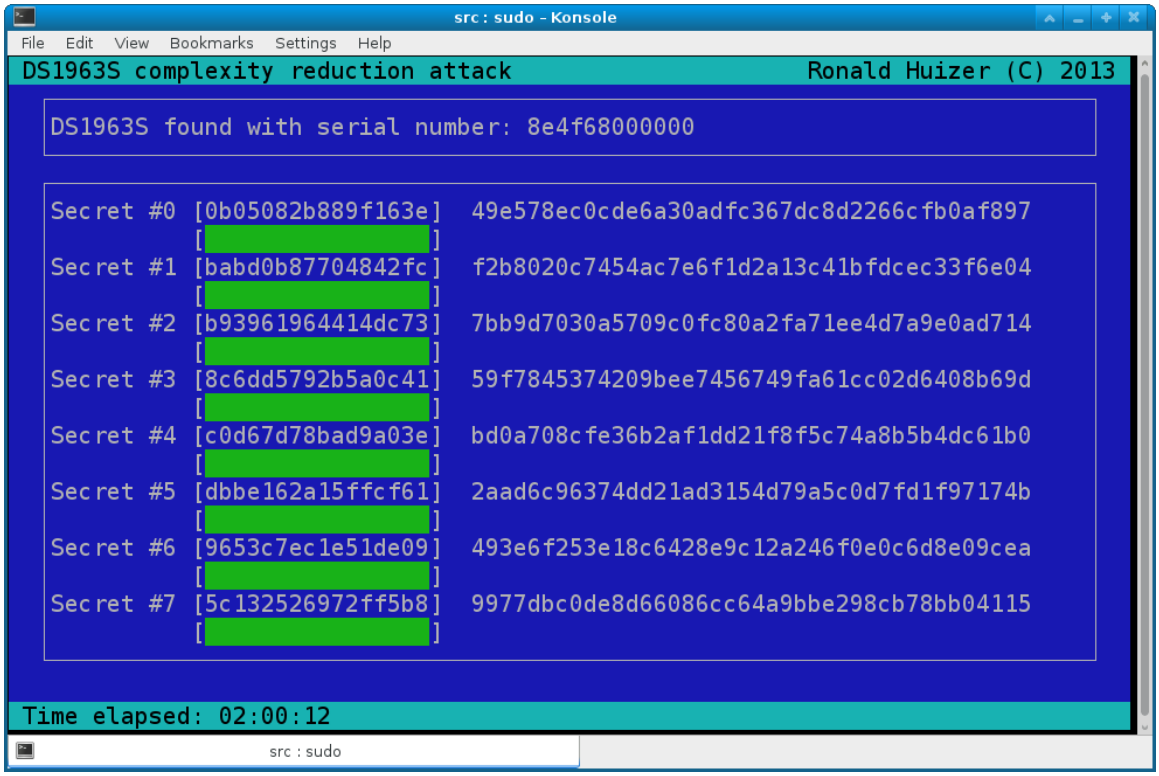

Figure 3. The side-channel attack completed.

Conclusion

We have demonstrated an adaption of the side-channel attack against the iButton DS1963S presented in Huizer (2012), which is applicable to all secrets the DS1963S stores. Furthermore, we have implemented the SHA1 functionality of the DS1963S in software allowing us to increase attack efficiency by parallellizing the DS1963S flaw and software based cracking of remaining key bytes.

This work completely breaks the security model of the DS1963S, and allows perfect copies to be made of these buttons. It should be noted that the process changes the write-cycle counters of the secret irreversibly, which can be a drawback in certain contexts. However, this work makes it cheap to retrieve all data of a given button, and this information can then be used to make a perfect copy of the button using custom hardware.

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`

Brandt, C. (2012, August). Personal correspondence..

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

Huizer, R. (2012, August). A novel attack on the ds1963s ibutton..