Cipher Instruction Search Attack on the Bus-Encryption Security Microcontroller DS5002FP

Markus G. Kuhn

Abstract—A widely used bus-encryption microprocessor is vulnerable to a new practical attack. This type of processor decrypts on-the-fly while fetching code and data, which are stored in RAM only in encrypted form. The attack allows easy, unauthorized access to the decrypted memory content.

Index Terms—Bus encryption, cryptography, tamper resistance, secure microprocessor, crypto processor, software protection.

1 Introduction

The idea of inserting cryptographic functions into the bus connection between a processor's CPU core and external memory was described first by Best [1], [2], [3], [4]. Bus encryption is used to protect confidential software and data that cannot be stored completely inside a single tamper-proof chip from being read by people with physical access to the circuit board. It avoids the cost and complexity of tamper-proof packaging for complete circuit boards [7]. The main applications are financial transaction terminals and pay-TV access-control decoders, where adversaries may easily gain full physical access to the system but must be prevented from obtaining the secret algorithms and keys stored inside the device. Another application is strong copy protection of software [8]. The Intel 8051 [6] compatible 8-bit microcontroller DS5002FP [5] is currently the most widely used commercial bus-encryption processor.

The attack presented here has allowed the author to access all the secrets stored in several commercial DS5002FP-based security systems within a few minutes, and a very similar attack can be applied to the older DS5000 processor, as well as to Best's original crypto-processor design.

2 Security Mechanisms of the DS5002FP

The DS5002FP implements the three on-chip block-cipher functions EA for 17-bit address-bus encryption, ED for 8-bit data-bus encryption, and ED^{-1} for 8-bit data-bus decryption, as shown in Fig. 1. K is a 64-bit secret key stored in tamper-protected, battery-buffered static RAM inside the CPU chip. The data-bus cipher function ED depends on both the key K and the accessed address a. When the CPU core writes byte d to address a, byte value $d' = ED_{K,a}(d)$ is stored at address $a' = EA_K(a)$ in external RAM. During a read access to address a, the crypto logic fetches from address a' the byte value d' and presents the decrypted byte value $d = ED^{-1}_{K,a}(d')$ to the CPU core. In other words, the byte values in the external RAM are individually encrypted and the addresses of all bytes are permuted.

After a DS5002FP-based device has been assembled, a small lithium battery must be connected to the processor. It will supply the CPU key register and the CMOS SRAM chips with enough voltage to retain data for the next 10 years. The device is then powered up. Using a special CPU pin, the system manufacturer activates a firmware monitor in the processor, which generates a new key value using the on-chip hardware random-number generator and stores it in the key register K. The software is then uploaded in clear into the CPU via the serial port and the firmware monitor stores it encrypted under K in the external SRAM. Once an on-chip security lock bit has been set using a firmware command, all subsequent firmware commands for memory access are disabled. This lock bit cannot be cleared without overwriting K. The idea is that nobody, not even the device manufacturer, can ever access K or use the firmware to decrypt and read out the software later. K cannot be changed without making it necessary to upload the entire software again through the serial port and firmware monitor. New software can be loaded into the device through the firmware after the lock bit has been cleared, but this will overwrite K with a new value and will thereby render the previously stored encrypted software meaningless. However, the uploaded software can modify itself, which allows application developers to provide a secure in-field replacement mechanism for most parts of the software.

All 8051 compatible microcontrollers feature separate program and data address spaces. The DS5002FP stores the first 48 bytes of program memory, including the reset and interrupt vectors, in an on-chip "vector RAM." Whenever the CPU core is not accessing external memory, the crypto logic will generate a pseudorandom dummy access in order to complicate bus-observation analysis. In addition, opcode fetch accesses are sometimes swapped with preceding dummy accesses to further complicate bus observation.

• *The author is with the University of Cambridge, Computer Laboratory, New Museums Site, Pembroke St., Cambridge CB2 3QG, U.K. E-mail: Markus.Kuhn@cl.cam.ac.uk.*

Manuscript received 22 July 1997.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number 105404.

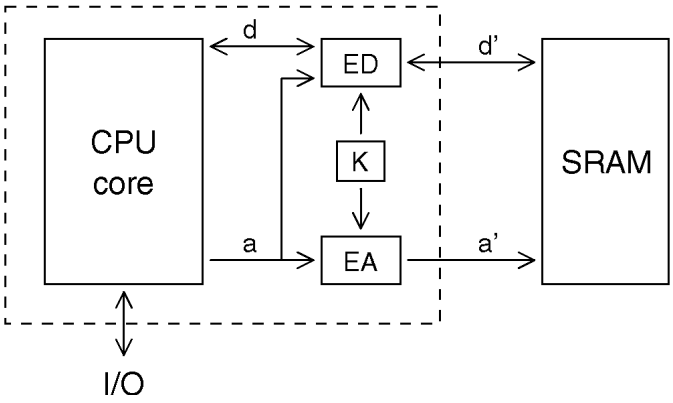


Fig. 1. Bus-encryption processor (dashed box) plus external static RAM.

3 Attack Concept

The idea behind a *cipher instruction search attack* is to present a large number of guessed encrypted machine instructions to the CPU and, then, to identify some of the decrypted machine instructions by observing the CPU reaction. The machine instructions identified this way are then used to form small encrypted programs that the attacker presents to the crypto processor to gain more information until, eventually, an encrypted program can be constructed that provides cleartext access to the entire protected memory.

We use external hardware to reset the DS5002FP processor repeatedly and, after each reset, at a selected moment, we substitute chosen instruction bytes for those that would normally be fetched from external SRAM. We observe the reaction of the processor on the chosen instruction bytes until we have identified instructions that help us in tabulating the data-bus encryption function for a sequence of eight consecutive addresses. We then use this information to introduce correctly encrypted instruction sequences. These will be successfully decrypted by the processor and will send the protected SRAM content in clear to the parallel port. We record the parallel-port output and, then, disassemble the protected software and extract all the sensitive data that was supposed to be inaccessible.

For the attack, we connect most pins of the DS5002FP CPU and two pins of each SRAM chip to a special read-out device. The CPU power supply has to be maintained carefully throughout this procedure to avoid the loss of the key K stored inside the CPU. Our device allows the computer that controls the attack to reset the analyzed CPU at any time and to record in FIFO memory the CPU reactions on the address bus, as well as on one of the four 8-bit parallel ports, as shown in Fig. 2. It also allows us to replace the encrypted bytes that the CPU fetches from SRAM with the content of an instruction FIFO which had been filled with a test sequence during the previous CPU reset by the controlling computer. To allow switching between SRAM and instruction FIFO, we interrupt the chip-enable and read/write connections between CPU and SRAM in the analyzed system and route these signals through the control logic of our device, which can then either pass the signals on to the SRAM or can block them and instead activate the FIFO output driver that is also connected to the data bus. The device can access all processor pins via a suitable SMD test clip. A detailed description of this read-out device can be found in [9].

The directly-addressed data-transfer instruction to a parallel-port register turns out to be an especially suitable instruction to search for. For instance, the command

```
75 a0 42   MOV a0h, #42h
```

writes the hexadecimal value **42h** into the latch register of parallel port P2, which is located at address hexadecimal **a0h**. Any of the other output ports could just as well be used.

4 Initial Tabulation

In order to search for the cipher bytes representing this **MOV** instruction, we fill the instruction FIFO with the five bytes

$$X, Y, Z, \texttt{00h}, \texttt{00h}^*,$$

where X and Y are search loop variables for which all 2^{16} combinations are tested systematically. The CPU is stopped when the instruction FIFO is empty and the two **00h** bytes simply ensure that the CPU runs for two more access cycles. The * marking indicates that the 8-bit value P visible at the observed parallel port will be recorded while this instruction FIFO byte is being fetched. For all pairs (X, Y), we test whether $E_0 : P \mapsto Z$ is a bijective function, i.e., whether 2^8 different Z values result in 2^8 different parallel-port outputs. In the common case of no port reaction, nonbijectivity

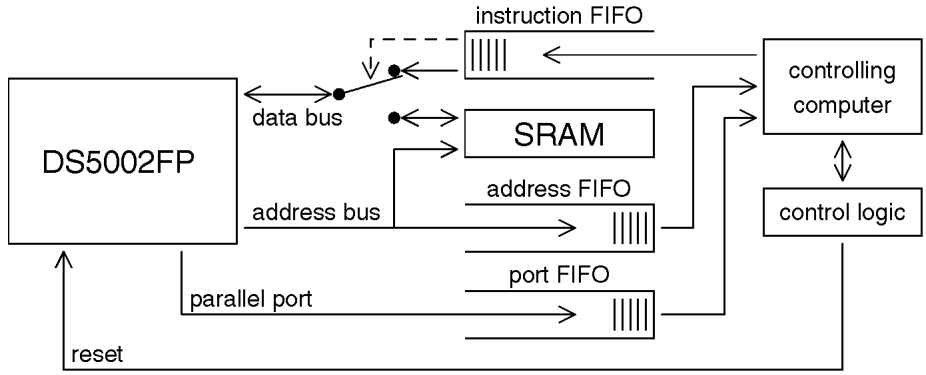


Fig. 2. Attacked CPU and SRAM connected to read-out device.

can already be verified for an (X, Y) pair after testing only two Z values, therefore, not many more than 2^{17} CPU resets are required for this test. As we can perform over 300 CPU resets per second, these tests take only a few minutes.

After each of the 2^{17} CPU resets, the switch from SRAM to FIFO has to occur when the same instruction is about to be fetched from SRAM. This will ensure that the CPU believes it has fetched the first FIFO value X from the same address a_0 each time and it will, therefore, always apply the same decryption function ED^{-1}_{K,a_0} to X. As the CPU fetches the first instructions after each reset from the on-chip vector RAM, we cannot provide the instruction FIFO content to the CPU directly after the reset. We have to guess when the CPU will start fetching instructions from external memory and have to switch from SRAM to FIFO at or after this point. The switch over to the instruction FIFO can be triggered either by a bus-access counter, by predictable port reactions, or by a characteristic sequence observed on the address bus.

Once we have identified a pair (X, Y) with $X = ED_{K,a_0}(75h)$ and $Y = ED_{K,a_0+1}(a0h)$, we have already tabulated the data-bus encryption function for address $a_0 + 2$ as $ED_{K,a_0+2} = E_0$, because the **MOV** instruction ensures that $P = ED^{-1}_{K,a_0+2}(Z)$.

However, **MOV** is not the only 8051 machine instruction capable of generating a bijective mapping from a fetched byte to the port value two access cycles later. The instruction **XRL** and, when the previous port value was **00h** or **ffh**, even **ORL** or **ANL**, will pass the bijectivity test, too. These instructions combine the previous port value and an argument using the bit-wise Boolean operations *xor*, *or*, or *and*, respectively. With the previous port value being either **00h** or **ffh**, we get three different candidate cipher op-codes X and at least two of them result in identical E_0 mappings. These two identical mappings are either those of **MOV** and **ORL**, or **MOV** and **ANL**, and, therefore, the E_0 mapping that occurred more than once corresponds to ED_{K,a_0+2}. For all other previous port values, only **MOV** and **XRL** result in bijective mappings and, in this case, both alternatives for E_0 have to be stored and tried in the next test series described below.

We must test all 2^8 values for X, but, once the first (X, Y) pair has resulted in a bijective $P \mapsto Z$ mapping, we can keep the value of Y constant as it decrypts already to the correct port address; this speeds up the search for the remaining X values by a factor of 256.

Having tabulated ED_{K,a_0+2} as described above, we now tabulate the data-bus encryption function for the addresses $a_0 + 3$ to $a_0 + 9$. In addition, we look for two cipher op-codes N_0 and N_1 to be fetched from a_0 and $a_0 + 1$ that are both single-byte instructions with only one dummy memory access and no serious side effect on any following instructions. Examples for such op-codes are **NOP** (no operation), **INC A**, or **SETB C**. We need these as padding instructions to move the **MOV** instruction ahead in the address space such that Z will be fetched from the next higher address, whose encryption function can then also be tabulated. Machine instructions of the 8051 architecture require an even number of access cycles, therefore, single-byte commands will always be followed by a dummy memory access.

For the next test series, we fill the instruction FIFO each time with the bytes

$$X, \lfloor Z/8 \rfloor, Y, E_0(\texttt{a0h}), Z, \texttt{00h}, \texttt{00h}^*.$$

Again, X and Y are the search loop variables. Although we are now looking for one additional **NOP**-like cipher op-code represented by $X = N_0$, the search complexity is still only around 2^{17}. We can already use the known data encryption table E_0, which we obtained during the previous test sequence, to determine the correct encrypted parallel-port address value E_0(**a0h**) that will be fetched from $a_0 + 2$. There exist many suitable values for X and the first one found is sufficient. So, this test hardly ever requires more than around 2,500 resets and can be performed within a few seconds. We are looking for (X, Y) pairs that fulfill the following two conditions:

1) the mapping E_1: $P \mapsto Z$ must be bijective and
2) the encrypted address from which E_0(**a0h**) has been fetched must be identical to the encrypted address $EA_K(a_0 + 2)$ from which Z had been fetched in the previous test series when the table E_0 was generated.

The address check ensures that the cipher op-code X has been an op-code for a one-byte instruction, that therefore the E_1 table actually represents ED_{K,a_0+3}, and that the second FIFO value $\lfloor Z/8 \rfloor$ satisfies only the dummy fetch of the **NOP**-like instruction. Using a value like $\lfloor Z/8 \rfloor$ that changes during the tests but that will not have 2^8 different values ensures that the **NOP**-like instruction represented by X is not one that exchanges the dummy access with the following op-code fetch.

If this test series fails and there are two alternative tables stored as E_0 candidates, then the other table will be tried. The first X value that passes the test will be stored as N_0 for the following tests. The **MOV** ambiguity that will result again in two or three different Y values is handled as with the first test series that tabulated E_0.

In order to tabulate $E_2 = ED_{K,a_0+4}$, we fill the instruction FIFO with

$$N_0, \texttt{00h}, X, \lfloor Z/8 \rfloor, E_0(\texttt{75h}), E_1(\texttt{a0h}), Z, \texttt{00h}, \texttt{00h}^*$$

and the first successful X value will be stored as N_1. This third test sequence usually requires less than 270 iterations, as only the opcode for the second **NOP**-like instruction is searched for and all other bytes can already be determined using the previously obtained tables E_0 and E_1.

Starting with the tabulation of $E_3 = ED_{K,a_0+5}$, no op-codes have to be searched for any more, as all bytes can now be determined using already known tables of the data-bus encryption function. The instruction FIFO will be filled with

$$N_0, \texttt{00h}, N_1, \texttt{00h}, E_0(\texttt{00h}), E_1(\texttt{75h}),$$
$$E_1(\texttt{75h}), E_2(\texttt{a0h}), Z, \texttt{00h}, \texttt{00h}^*,$$

where E_0(00h) is the encrypted op-code of the NOP instruction. The dummy access following this NOP instruction is already served with the op-code of the next instruction, because the processor could swap the dummy access and the op-code fetch. This test requires only 256 CPU resets to tabulate E_3. E_4 to E_7 can be tabulated the same way by inserting additional NOP instructions before MOV, each of which will increase by one the address from which Z will be fetched.

5 Accessing Protected Memory

Using the values N_0, N_1, and the tables E_0 to E_7, we can now fill the instruction FIFO with tiny encrypted programs that will send everything accessible by the protected software to the output port. For example, to access the program address space, we use the instruction sequence

```
00          NOP
00          NOP
90 xx yy    MOV DPTR, #xxyyh
e4          CLR A
93          MOVC A, @A+DPTR
f5 a0       MOV a0h, A
00          NOP
```

to send the byte stored at the address xxyyh to the port. This corresponds to the following instruction FIFO content:

N_0	NOP
00h	dummy access
N_1	NOP
00h	dummy access
E_0(90h)	MOV DPTR,...
E_1(X)	high address
E_2(Y)	low address
E_3(e4h)	dummy access
E_3(e4h)	CLR A
E_4(93h)	dummy access
E_4(93h)	MOVC A, @A+DPTR
E_5(f5h)	dummy access
E_5(f5h)	dummy access
–	read access to SRAM
E_5(f5h)	MOV..., A
E_6(a0h)	port address
E_7(00h)*	NOP

The first and third byte do not necessarily represent NOP instructions; other single-byte instructions can serve a similar purpose. The instruction FIFO hardware is actually nine bits wide and the ninth bit controls a temporary switch back to SRAM, so that a machine instruction that has been fetched from the FIFO can get and decrypt data from the SRAM. This mechanism is used here in FIFO byte 14 for the fourth memory access of the MOVC instruction. As mentioned above, the DS5002FP exchanges an op-code fetch with one of the previous dummy memory accesses after certain instructions, but the above instruction FIFO content sees to it that the next cipher op-code is available when required.

Very similar instruction sequences can be used to dump the data address space and the special function registers. The on-chip vector RAM area can be read as part of the program address space. These 48 bytes are write protected and, therefore, are not overwritten during the cipher instruction search.

6 Further Ideas

The described memory-access technique lets us read several hundred bytes per second. For faster access, several bytes can be sent to the port by one single instruction FIFO content. As the SMD test clip used on the bus in the analyzed system does not provide very reliable contact, special contact test sequences should be executed periodically during read-out. This provides quick detection of data errors caused by contact problems.

Another cipher instruction search allows us to identify encrypted jump commands. We look for a first byte that provides an injective mapping from the second byte to one of the addresses from which the following bytes are fetched. The search complexity is only around 2^9, therefore, we can test within seconds whether the byte fetched from a_0 is interpreted by the CPU as an op-code or whether the switch from SRAM to FIFO has to be done somewhere else.

7 Conclusions

Although the DS5002FP has been described as the most secure processor currently available for commercial users, and although it has even been protected by special top-layer die coatings against microprobing attacks, the technique presented here defeats the chip's whole security concept using only a personal computer and a device built in a student laboratory with standard components for around US$300. After only a few hours preparation, the author was able to extract the protected software from a DS5002FP (Revision A) based demonstration system that Peter Drescher from the German Information Security Agency (BSI) built as a challenge in July 1996.

A variety of countermeasures can make cipher instruction search attacks infeasible in future bus-encryption processors. If the data-bus block-encryption function operates on whole cache lines of at least eight bytes instead of on single bytes, tabulation will become impractical. Processors without cache can implement restrictions on the maximum number or frequency of both resets and illegal op-codes that the CPU will accept without delaying further resets or destroying the secret key. Instructions that are particularly useful for cipher instruction search attacks might be represented by long op-codes to complicate the search.

Bus encryption continues to be an interesting concept, but a secure implementation is harder than might at first appear. The cipher instruction search attack presented here did not depend on properties of the protected software to get unauthorized access. Unless the software developer is extremely careful, attackers can learn much from observed encrypted bus activity or from the reactions of the protected software on external memory modifications, as the following examples illustrate. The critical conditional jump of a password-check routine is easily identified by comparing bus traces of a successful and a rejected login attempt. After a short cipher instruction search, the attacker can replace the conditional jump instruction with either NOP-like instructions or with the unconditional variant of the jump instruction, in order to get unauthorized access without having to know the password. Encryption and string-compare routines are easily recognized by their cyclic loop

traces. Observing and interfering with the bus while these algorithms execute can help in reconstructing secret keys and passwords. A data-transmission loop is as easily recognized and, once it has been transformed by a single instruction-byte change into an endless loop, it will dump a significant part of the protected memory content to the communication port [10]. Security reviewers should use simulation tools that show traces of the bus activity to get an attacker's view of potentially vulnerable instruction sequences. Goldreich and Ostrovsky [11] discuss systematic techniques to keep attackers who observe or interfere with encrypted bus activity from gaining any knowledge, but they require many additional redundant access cycles and, therefore, decrease the performance. Designers should probably combine bus-encryption processors in high-security applications with several independent protection mechanisms, such as secure packaging and a design that can easily recover from the compromise of single devices to provide reliable overall tamper resistance.

Both the DS5000 and DS5002FP are used in a very large number of credit-card terminals and other security sensitive applications. Therefore, the author considered it good practice to inform the manufacturer of this processor more than a year before submitting this paper. The manufacturer has, in the mean time, informed customers, developed countermeasures usable for currently fielded processors, and has added further countermeasures in new mask revisions.

ACKNOWLEDGMENTS

The author wants to thank J. Stelzner, G. Roncolato, and P. Drescher for their help and suggestions during this project, as well as R. Anderson, C. Flack, and the reviewers for their comments on the paper. The presented technique was implemented while the author was with the Computer Science Department at the University of Erlangen-Nürnberg, Germany.

REFERENCES

[1] R.M. Best, "Preventing Software Piracy with Crypto-Microprocessors," *Proc. IEEE Spring COMPCON '80*, pp. 466-469, San Francisco, 25-28 Feb. 1980.

[2] R.M. Best, *Microprocessor for Executing Enciphered Programs*, U.S. patent 4,168,396, 18 Sept. 1979.

[3] R.M. Best, *Crypto Microprocessor for Executing Enciphered Programs*, U.S. patent 4,278,837, 14 July 1981.

[4] R.M. Best, *Crypto Microprocessor that Executes Enciphered Programs*, U.S. patent 4,465,901, 14 Aug. 1984.

[5] *Secure Microcontroller Data Book*, Dallas Semiconductor, Dallas, Tex., 1997.

[6] *MCS 51 Family of Microcontrollers Architectural Overview*, Intel, order no. 270251-004, Sept. 1993.

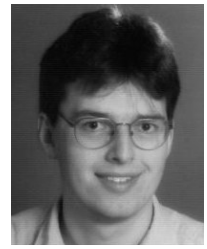
[7] *FIPS PUB 140-1, Security Requirements for Cryptographic Modules*, Federal Information Processing Standards Publication, Nat'l Inst. of Standards and Technology, U.S. Dept. of Commerce, 11 Jan. 1994.

[8] S.T. Kent, "Protecting Externally Supplied Software in Small Computers," PhD thesis, MIT Laboratory for Computer Science, MIT/LCS/TR-255, Cambridge, Mass., Mar. 1981.

[9] M.G. Kuhn, "Sicherheitsanalyse eines Mikroprozessors mit Busverschlüsselung (Security Analysis of a Microprocessor with Bus Encryption)," diploma thesis, Univ. of Erlangen-Nürnberg, Lehrstuhl für Rechnerstrukturen, IMMD III, Erlangen, Germany, July 1996 (in German).

[10] R.J. Anderson and M.G. Kuhn, "Tamper Resistance—A Cautionary Note," *Proc. Second USENIX Workshop Electronic Commerce*, pp. 1-11, Oakland, Calif., 18-21 Nov. 1996.

[11] O. Goldreich and R. Ostrovsky, "Software Protection and Simulation on Oblivious RAMs," *J. ACM*, vol. 43, no. 3, pp. 431-473, May 1996.



Markus G. Kuhn received his Diplom at the University of Erlangen-Nürnberg (Germany) in 1996 and his MSc at Purdue University, West Lafayette, Indiana, in 1997, both in computer science. He is currently with the Computer Laboratory at the University of Cambridge, United Kingdom, supported by a European Commission training grant. His main research interests include distributed systems and computer security, especially the security of tamper-resistant hardware, intellectual property protection mechanisms, and global-scale distributed databases.