Firmware Protection and Attacks Against ATmega Microcontrollers

Dionisio Perez-Mavrogenis

Abstract—In this paper we will propose a possible non-invasive clock-glitch attack against the popular AVR ATmega644 microcontroller with the aim of obtaining access to the firmware or cryptographic material, synthesised from research performed on boards with similar architectural properties. We will also present current attacks on microcontrollers, and the protective mechanisms developed in response, in order to understand how the proposed attack works, concluding with the sad realization that absolute security is an illusion and manufacturers should instead try to make their products secure "enough" based on a given threat model.

Index Terms—firmware protection, attacking microcontrollers, microcontroller protection, atmega

1 Introduction

This paper will present an overview of the current attack methods for tampering with MCUs (MicroController Unit) in order to obtain access to IP (Intellectual Property), where IP refers to the firmware of the MCU and information that could be obtained from that, i.e. implementation secrets or proprietary algorithms that help a manufacturer perform better over their competitors. Firmware tampering (or theft) detection and prevention of a MCU is a popular problem with active research, as successfully addressing it would benefit the government and military, the car industry and various service providers [23], to name a few. Due to the diversity in the functions MCUs perform and their operating environments, MCUs are engineered differently and a distinction between ordinary or commercial and MCUs engineered with protective mechanisms should be made[23] and, due to the sophistication of the protective mechanisms and the attacks, a broad classification of attackers into three groups [2], namely :

- **Class I** Clever and curious people with a very limited budget, with some degree of knowledge and no time restrictions.
- **Class II** Professionals skilled on electronics with access to specialised equipment and resources. Their time allowance and funding might be limited.
- **Class III** Organisations with access to MCU manufacturing equipment. Their funding is usually unconstrained and their time schedule might be tight.

Firmware tampering has a number of consequences, including attackers downloading the code from a MCU and flashing it onto a MCU that they sell, effectively avoiding development and testing costs but still offering the same product as other manufacturers [9], theft of private cryptographic keys or other industrial secrets and perhaps theft of service [23]. A less obvious, but perhaps more important, case is the case of back-dooring[1] a MCU by re-flashing on it a modified version of the firmware with code added by the attacker in order to accomplish their malicious intents, a very realistic scenario for governments that order electronic equipment from other countries, effectively impacting a large portion of the population depending on the targeted organisation.

It is essential for an attacker to be familiar with the technical features and architecture of the board they are trying to attack (Sec. 2) and it is equally important to understand the current attack mechanisms (Sec. 3) and the protective mechanisms developed in response and which particular attack type they protect against (Sec. 4), if one wants to be effective on their attack. Once familiar with the above attackers can decide how they want to attack the board, based on their goals (Sec. 5).

2 The AVR MCU Series

In this paper we will focus on the ATmega644 board, of the Atmel AVR line of MCUs, as it is a widely available and popular low-cost microcontroller [5] with little loss of generality however, as boards in the same line of MCUs are very similar architecturally and hence the information here should be transferable with minimal hassle. The AVR series is an enhanced-RISC architecture 8-bit MCU family that consists of the ATtiny, ATmega and ATxmega sub-categories, 32-bit AVRs and application specific FPGAs[24]. The models have varying degrees of hardware capabilities and large operating voltage windows in order to accommodate demand and integrate well with peripherals. Developing software for an AVR is easy as the AVRs benefit from the `avr-libc` high-performance library, the `avr-gcc` and `avr-gdb` compiler and debugger(both based on very popular and high quality GNU software tools), the `avrdude` programming software(or Atmel's `AVRStudio`) and `Simulavr` simulator software. Additionally, Atmel provides proprietary

1. The act of adding code to a system without the user's knowledge.

APIs for interacting with the AVR and the developers can choose from a wide variety of programmer units available for working with the AVRs[24].

2.1 ATMega Architecture and Features

This paper will focus on the ATmega644, an enhanced-RISC Harvard architecture 8-bit CPU with a two stage pipeline and a total of 131 instructions. Fig. 3 shows the conceptual difference between a Von Neumann and strict Harvard architecture, where the key distinction lies in the separation of application code and program data into different memory sections (Harvard) and tasking the CPU with distinguishing between code and data present in the same memory region (Von Neuman). The 644 implements a modified Harvard architecture for both power and computational efficiency, designed to access multiple memory locations simultaneously thus being able to execute an instruction per cycle, as shown in Fig. 2. Their speed grades are 0-10 MHz for 2.7 V - 5.5 V and 0-20 MHz for 4.5 V - 5.5 V [8].

The 644 is equipped with 2 Kb of EEPROM, 64 Kb of flash memory, 4 Kb of SRAM, and a large number of general purpose (GP) and I/O registers and all memory (including memory mapped I/O images) is linear, i.e. it follows the flat memory model. The flash memory is separated into the bootloader and application code section and the boundary between the two sections, as well as the page size, can be configured by programming the appropriate fuses. Both sections hold code, however code residing in the bootloader section can execute the `SPM`[2] instruction which allows the bootloader code to write to *any* section in the flash memory and hence possibly modify itself, facilitating purposes like firmware upgrades. The bootloader code can be triggered by a direct jump from the application section or by programming the reset vector via the reset fuse to point to the appropriate section of the bootloader code. The EEPROM is memory for data that needs to persist between reboots of the MCU and hence it is (widely) used to hold configuration variables and other non-temporary data the application code (or the bootloader) may need, having an average lifespan of 100,000 write cycles per page. The SRAM is volatile storage and is used as the stack and heap for the firmware and for storing the Register File, i.e. the 32 GP registers, I/O and Extended I/O Memory. The reserved register locations exist in order to support the use of peripheral units as well as hold program status information (e.g. the Stack Pointer can be found in one of the GP registers). Fig. 1 gives an overview of the SRAM hierarchy for the ATmega644.

2.2 ATMega Security Features

The AVR ATmega644 is not meant to be a secure hardware module but offers firmware access control by using six lock bits responsible for controlling access to the board's memory and prevent reading or modifying the memory (e.g. prevent code executing from the bootloader section to read/write the application code section using `SPM` or the memory being read out with a programmer). This access control is not permanent, as that would limit the usefulness of the MCU and therefore one has the option to reset the lock bits (i.e. having no protection scheme enabled) by issuing a Chip Erase command, which has the effect of completely erasing the Flash, EEPROM and then the lock bits [8]. Chip erasing is performed with the sequence of events as presented and this is important, as one does not want to remove the access protection before removing all data and hence the lock bits are set to 1 only after the whole program memory has been erased. Even though the flash memory has an average lifespan of 10,000 write cycles, as well as programming being a relatively lengthy operation, this approach tries to preserve the intellectual property on the board rather than the board itself. Table 1 presents an outline of the available Lock bits provided by the ATmega series. The functionality of the BLB1 group is to control access and modification of the bootloader section, group BLB0 bits control access to the application code

2. `SPM` = Store Program Code, assembly instruction for the AVR.

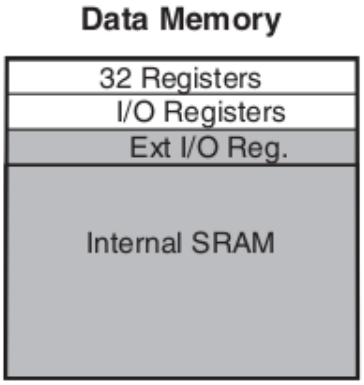


Fig. 1: SRAM layout for ATmega644 (reproduced from: [8]).

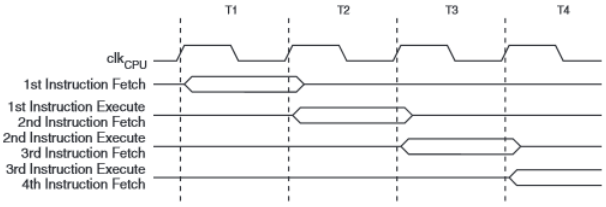
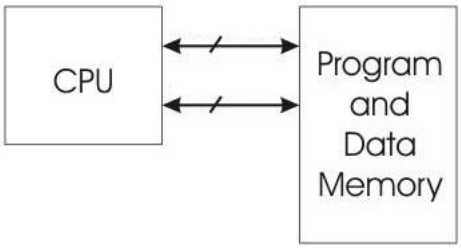


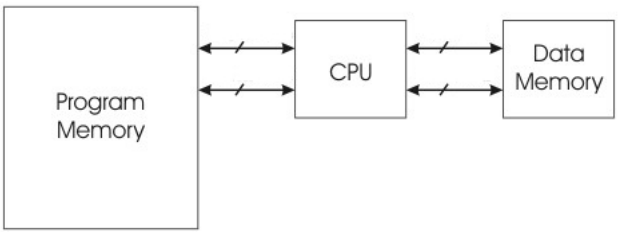
Fig. 2: 2-stage pipeline of the ATmega644 (reproduced from: [8]).

TABLE 1: Security lock bits offered by the ATmega644. BLB stands for Boot Lock Bit and LB for Lock Bit. (reproduced from: [4] [8])

Lock Bit Byte	Bit Number	Default
BLB12	5	1
BLB11	4	1
BLB02	3	1
BLB01	2	1
LB2	2	1
LB1	1	1



(a) Typical Von Neumann architecture.



(b) Strict Harvard architecture.

Fig. 3: A comparison of different machine architectures (reproduced from: [11]).

section and group LB bits are responsible for controlling modifications on the EEPROM and Flash. A detailed explanation of their functionality and how to use them is given in the manuals [8] [4].

3 Attacks on Hardware

After getting more familiar with the attack target and its architecture, an occasionally complicated process [23] [25], one is made aware of the protective mechanisms of their target and, depending on the sophistication of the protections and the goals of the attacker, one should decide between *passively* or *actively* attacking the chip. When performing a passive attacks one monitors the chip's normal operation and tries to infer the input-output mapping whereas in active attacks the attacker manipulates the chip or its operating environment with the aim of obtaining insight on the chips inner workings by abusing abnormal behaviour. Attacks on MCUs may attempt to recover cryptographic keys or the firmware on the device and do not necessarily target the hardware itself but can exploit flaws in algorithmic design and implementation and protocol failures, inter-component communication patterns [2] [14] or exploiting memory remanence [23] [10].

The following discussion closely follows Skorobogatov [23] and the Hardware Reverse Engineering course material [25].

3.1 Non-Invasive Attacks

Non-invasive attacks are attacks which require no de-packaging or special preparation of the chip and hence leave little tamper evidence behind. These attacks might be very time consuming to find and are not guaranteed to be successful (researchers report an average exploitation rate of $\tilde{5}0\%$ [23] [5]), but are very cheap to replicate once found. A popular technique is power analysis, exploiting the fact that different instructions executing on a CPU require different amounts of power [15] [14] and hence one can infer which instruction is executing on a CPU by analysing a power trace generated by the MCU. These attacks are easy and relatively inexpensive to perform as they only require equipment for sampling voltage differences and constructing a power trace trace [14]. Simple Power Analysis (SPA) involves direct observation of the MCU when it performs cryptographic

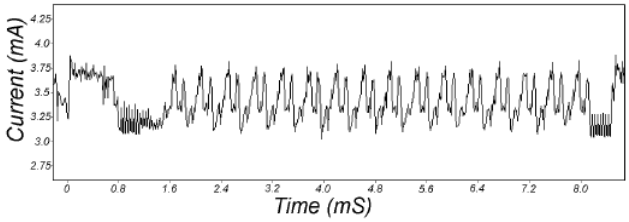


Fig. 4: Power trace during a DES encryption, where the 16 rounds are clearly visible (reproduced from: [14]).

operations and can leak information about both the keys and the cryptographic operations themselves like the nature or structure of the algorithm [14] [3]. Differential Power Analysis (DPA) is a more powerful version, incorporating the notion that power consumption is also data dependent, and extracts sensitive information by exploiting the fact that power consumption is related to the values being manipulated and applies tries to statistically correlate the obtained power traces with possible keys and does not need deep knowledge about the attacked device [17]. DPA involves obtaining large power traces of known cipher-texts (but not necessarily knowing the corresponding plain-texts) and individual bits of the key are recovered by analysing the statistical properties of the power consumption for different key guesses [14] [3].

Another popular attack type is inducing glitches or faults, which can be injected in a MCU by operating it outside its operating temperatures, voltage or clock frequency and exploiting the resulting undefined behaviour of the MCU [23] [13]. Although inducing a fault is easy, inducing an exploitable fault is hard, but can be achieved by systematic search [23] [5] [15]. *Power glitches* and *clock glitches* aim to make the CPU skip or execute incorrect instructions by applying transients. This attack is board-specific and can target in individual components of an MCU and a systematic search can deduce which components are affected by a given glitch sequence. Clock glitches involve increasing the clock signal frequency so that some flip flops sample their input before being updated and hence report an incorrect value [23] and are mainly aimed against software-based protection mechanisms, affecting CPU operation by supplying the CPU with incorrect data (or instructions). Power

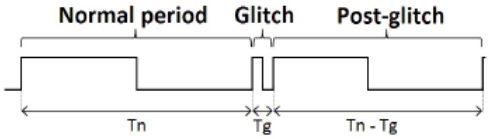


Fig. 5: Illustration of the principle of a clock-glitch attack.(reproduced from: [5]).

glitches work by supplying either too much power or too little, shifting transistors' threshold and causing flip-flops to read their state incorrectly and as a result need to be carefully synchronised with the internal clock and prolonged attacks might damage the board [23]. Glitch attacks are especially dangerous as they may abuse the program counter in order to map out the memory [5] [2] [23].

Data remanence in SRAM cells is another possible attack vector, as prolonged exposure of SRAM cells to the same values can make the cells 'remember' their state. Hot-carrier effects and electromigration effects can cause physical alteration to a cell's surface, leaving an excess charge which can alter a cell's threshold voltage noticeably and reveal which type of signal was most common for particular cell (0 or 1) [10]. Furthermore, because of the hit-carrier effect high speed electrons can be trapped in the gate oxide to remain there as excess charge. Cooling the memory down can prolong the time for this charge to deplete, thus prolonging the time available for data recoverability [10] [22] [23]. Hot-carrier effects also apply to EEPROM memory, but to a lesser extent, as material-wise one can only tell virgin-cells from used cells and there might be a shift in the threshold voltages of erased cells[10] [23]. Finally, due to its simplicity, another common attack includes timing attacks, which in MCUs is even more noticeable as instructions have fixed execution cycles. Timing attacks exploit the software implementation of cryptographic algorithms. Compiler optimisations and other implementation choices might make the execution time of an algorithm dependent on the input and the secret key rather being fixed for any input, for example when input is compared byte-wise with a key and rejected when the first non-matching byte is found, rather than first consuming the whole input string. Furthermore different instructions have different execution times (`RET` is considerably slower than `INC r1` [8]) and thus one could collect timing information for various input messages and systematically brute-force the key. If timing information is correlated with power analysis then defences such as constant instruction execution time could be defeated. One might use `NOP`s in the case of a wrong key in order for rejection and confirmation responses to have constant execution time but `NOP` consumes substantially less power than `LD` [5] and correlating timing and power consumption information would reveal this.

3.2 Semi-Invasive Attacks and Invasive attacks

Attacks under this category require depackaging of the chip and, while semi-invasive attacks do not require complicated machinery or any sort of contact with the die surface, invasive attacks require physical contact and partial depassivation and are therefore a lot more expensive, time consuming, complicated and leave tamper-evidence [23] [10]. Decapsulation of the chip can happen in a number of ways, depending on the packaging [25] [20]. HNO_3 or H_2SO_4 could be used if the packaging is made out of epoxy resin and, if one wants to remove the residue or dirt left from chemical etching, then bathe the die in an ultrasonic bath with acetone or distilled water [23] [25] [20]. Alternatively, one could send the chip to a hardware analysis lab and get them to open the chip [12]. Once the die surface is exposed it could be attacked with UV light. Older chips and chips that are designed to withstand low-cost non-invasive attacks are susceptible to having parts of the memory altered if it is exposed under UV light [23] [25], altering memory contents or resetting security fuses that prevent read-back of the memory by sufficient exposure under UV light. Mechanisms that rely on fuses or checking a memory address for a particular value are susceptible to this kind of attack.

A possible step after decapsulation is reverse-engineering the MCU in order to understand how it does its task, potentially having to image numerous layers. Low cost devices can usually be imaged from the top side, since they do not contain many layers or overlapping layers. If layers are overlapping mechanical etching could be used to take successive layers off [20] [25] or one could image the chip from the rear [25] [23]. Backside imaging involves shining IR light on the rear side of the chip and imaging it from this angle, since it is a mirror-image of the front side. This is possible because, usually, light shown through the backside does not have to go through multiple layers and hence protective metal meshes or normal chip layers are avoided [25]. On some chips it is possible to extract ROM contents via this technique by directly observing the memory [23]. An alternative to IR light would be the use of lasers and the photoelectric effect for imaging, and Optical Beam Induced Current and Light Induced Voltage Alteration are the two most common techniques for failure analysis that take advantage of the photoelectric effect. These techniques involve shining lasers on the semiconductor surface in order to alter some property; in OBIC a slight current is created and by analysing this one can deduce the device's properties (including defects and anomalies) and produce an image of the the board being scanned, while in LIVA the board is connected to constant power supply and changes in the power supply are monitored as laser is shone on the device, allowing one to deduce the device's characteristics and construct an image [7]. Lasers can also be used to read the state of memory cells in CMOS SRAM [23] or in order to corrupt memory

contents by localized heating, by shooting a laser on a memory cell long enough for its state to change [15] [23]. For speeding up reverse engineering of a chip, one might physically need to modify it. This is certainly the case for invasive attacks [23] [10]. Specialized equipment is hard and expensive to find, but one could go for slightly older second-hand equipment with a basic setup including a microscope (optical is good but SEM is better [23] [25]), a moving base for the chip to go onto, micro-positioners and probing needles with μm-thick precision and the test circuitry and hardware used, all of which could be obtained for under $5000 [16] [10] [23] [25]. Furthermore one could use cutting lasers in order to etch through the passivation layer in order to accurately position their probes [23], as well as destroy circuitry. A more reliable method includes using focused ion beams for accessing deeper chip layers for creating and destroying interconnects, but it is extremely expensive to obtain the machinery and knowledge on how to use it is essential (although renting time on one is an option) [23] [25] [10].

4 Countermeasures to known attacks

As demonstrated in the previous section a manufacturer has to guard against a multitude of attacks. For designing effective defensive mechanisms one has to enumerate the likely attack scenarios and methods, type of attacker they will be facing and decide the type and extend of confidentiality they would like to provide [23] [14].

A common first line of defence that is often encountered in attempts to secure systems, with questionable effectiveness, is security by obscurity and in the case of MCUs can be achieved in a number of different ways. The manufacturers can simply avoid printing their logos or model numbers on parts they produce or print on their products identifiers of more secure or expensive products or even try to make their MCUs look like ASICs in order to scare away potential attackers by making their product appear more secure [23] [25], as this will make information gathering for the chip trickier. A further step in making the MCU harder to analyse is hardware obfuscation by making the interconnects into a maze [25] or by making a group of gates commonly placed in well-known structures for performing a given task (for example DES circuitry) intentionally more complicated and physically placed in a different locations or additional circuitry is added in order to thwart analysis [2]. Vendors also try to make information on their products hard to find by selling only to selected partners or under non-disclosure agreements [23].

Security through obscurity has received lots of criticism, and rightfully so, as it will do little to stop experienced or determined attackers. Despite its financial appeal a more effective approach is security in depth, a model in which information is protected by a number of (potentially) different defence mechanisms and in this scenario could be employed at different levels of the chip, from its external potting (or encasing) to its layered architecture. Given the attack categories presented in Sec. 3, it would be useful to broadly categorize defences in a similar fashion.

Non-invasive attacks are the least sophisticated type of attack but protection against them is a relatively laborious process [2]. A first step to prevent unauthorized access to the firmware is using lock bits and fuses [8] [4] and some manufacturers (or developers) go as far as physically destroying reading pins or cutting out testing circuitry [23], while still being able to safely deliver firmware update [9]. Another protective step includes avoiding side-channel leakage by masking all relationships between data input and power consumption, thermal and electromagnetic radiation and timing relationships [14] [23]. Given the software release model and the power of MCUs it's not unreasonable to strive for efficiency, both in terms of consumption and components, and bytecode optimizations performed by compilers and the hardware architecture of the instruction pipeline of an MCU might introduce leakage [14] [23] by how branches are performed, instructions pre-fetched or not take constant time for operations (or execution of different instructions). Even if constant time for cryptographic operations is taken for correct and erroneous keys, or random delays are introduced such that timing relationships are masked, storing intermediate results still poses a problem due to the fact that a 0 and a 1 consume different power when handled (due to the physical nature of CMOS transistors) and hence the power consumed in a given operation is proportional to the amount of 1 bits, a concept known as "Hamming weight" [15] [14] [17]. More approaches to thwarting power analysis include using a lower operating voltage so that power fluctuations are less evident or is introducing noise and delays by means of a random number generator with variable power consumption [14] [25] [13] [17]. Electromagnetic or thermal emission detection can be avoided by packaging that is appropriately shielded [6] [14].

Protection against invasive and semi-invasive attacks is a lot harder because these attacks involve a range of chip modifications (from decapsulation to modification of the die), requiring varied anti-tampering mechanisms. A common technique employed by a number of MCUs is to keep cryptographic or otherwise secret information in an internal battery-backed SRAM [25] [23] module connected to a tamper detection circuit, in order to be able to zero-out sensitive information on tamper detection; we will proceed to give a description of tamper detection mechanisms in the order in which decapsulation occurs. Thwarting decapsulation can get very creative as decapsulation techniques depend on the material from which the chip packaging is made of (varied from pure plastic to ceramic or metal) and its physical construction, with techniques ranging from using a sharp object to pull the top off [23] to using acids and other chemicals [25] [23] to etch the top layer away and expose the die surface and manufacturers trying to make the decapsulation process

as hard as possible. Common techniques involve sealing the die in conductive packaging or making the packaging from conductive and very hard epoxy resin such that packaging removal will result in power supply loss and a response from intrusion detection circuitry from the chip; common responses include erasing sensitive information from the internal SRAM [25], resetting of the device [23] or, for military grade equipment, have reactive chemicals or little charges embedded in the packaging that would respond to stimuli such as other chemicals or small currents (as a result of optical imaging) in a very violent manner and destroy the chip completely. Additional measures include the scattering or photoresistors inside the epoxy which would detect light if the epoxy was removed and hence the tampering attempt [23] [25].

The next level of protection came when radiation attacks became known [23], where attackers used to override fuses by exposing them to UV light. The defensive response to that was the addition of opaque metal rectangles on top of critical components, or the entire die, in order to shield them from radiation and scattering additional security fuses, which were usually hidden near critical memory areas like the Reset or Interrupt Vector Addresses in order to make harder to locate them and damage other components as well when tampered [23] [25]. A further step to protect the top of the device was to place conductive wire mesh layer(s) over the chip area before it is embedded in epoxy , like in the IBM μABYSS [6] and the Atmel ATSHA204 [25], in an attempt to detect tampering and erase the keys from SRAM if the power is disrupted. These mechanisms attempt to thwart micro-probing and modification attacks (both memory contents and circuitry modification) but also prevent visual inspection and analysis of the die [25], making the localisation of critical components harder and are commonly found in smart-cards [23] and SIM cards as well [25]. Another approach to preventing visual inspection and IR imaging (without de-layering the die) is the doping of semiconductor materials in order to reduce light penetration [23], as well as chemical or mechanical planarization of the layers of the MCU [23] [21]. Physical modification of the chip could be prevented by having both hardware health-check routines and by software self tests comparing the known cipher-output of a magic value stored in the device with the actual output of that device [3].

Additional tamper-resistance detection and prevention methods include the addition of environmental sensors for detecting temperature, radiation, operating voltage and clock frequency [23]. Expensive or military grade equipment comes with even more tamper detection features than the above, like tilting sensors (a popular example is the IBM 4758) [6], or are designed in a way that removal of a layer should guarantee the destruction of other layers [2]. Although these measures sound complicated, in practice they are not as effective as they pretend to be and only protect against one specific attack (e.g. voltage regulators will not respond to clock glitches) as well as introducing some instability issues [2]. For example optical sensors would fail to detect an attack utilizing a laser as a light source in a dark room [25] or one could paint over the sensors with black paint [23]. Tampering and micro-probing is in reality being made harder by the shrinking sizes of the various components due to technological progress, requiring more expensive equipment and specialization, as well as the use of different fabrication techniques, like the application of an ASIC-like glue logic [23] [25].

5 Attacking The ATmega

In this section we suggest an attack that should work against the ATmega644. We believe that someone with a modest level of competence in electronics could successfully bypass the security fuse and lock-bit protections on the ATmega644 board, as researchers have succeeded in bypassing the protection imposed by the AVR family in numerous ways, using non-invasive attacks in order to achieve this since there is no need for more sophisticated attacks. Clock glitch attacks against AVR chips were successfully applied by Balasch et al. [5] against an ATmega163 and by O'Flynn et al. [18] against an ATmega328P and Kizhvatov [13] managed to retrieve AES and DES cryptographic keys from an ATmega16 and ATXmega128A1. Furthermore, Skorobogatov [23] also points out that AVR MCUs are susceptible to glitch attacks due to the implementation of their security fuses. The suggested attack method will closely follow Balasch et al. [5], further supported by results from other research as well. We will be a Class-I attacker performing a non-invasive clock-glitch attack. The equipment we have access to can be found in any decent university's electronics laboratory and our information regarding the chip will come from datasheets. We believe that the attack engineered by Balasch et al. [5] on the ATmega163 can be ported over to the ATmega644 due to their similarities. Both devices belong to the same family and by comparing their datasheets we can see that the 644 possesses all the hardware features that make the attack possible on the 163, namely it operates on an external clock signal and has a two stage pipeline[5]. Furthermore, the two devices have an almost identical, small, instruction set with instructions requiring a maximum of 5 cycles to execute.

The attack should be tailored to a particular goal and hence we should distinguish between recovering cryptographic material from manipulating program or data flow. Notable subtleties include that the glitch attack should be synchronized with the devices operation, achievable by power trace analysis like in Fig. 4 where one can see when DES encryption begins, and that the two frequencies should preferably be phase-aligned [5][23][13]. For power analysis attacks it is important that the captured traces are carefully aligned so that two identical operations are perfectly time-aligned.

Performing a clock-glitch attack for corrupting the program or data flow by the ATmega644 would require

locating when this happens and then exploiting it. For our attack scenario we assume the firmware we would like to dump has at some point an `OUT p` instruction, i.e. it outputs a memory location to port `p`, and runs on some loop (both reasonable assumptions). Since it is less complicated to exploit program flow rather than data flow [5], we would attempt to skip the branch instruction on the check of each loop. The AVR assembly for your typical `while(condition) {code}` loop is shown in Table 2, along with the cycles that each instruction takes [8]. Even if we don't know the exact length of the loop, one could monitor how long it takes between successive outputs to port `$18` (i.e. our trigger event) and from that time estimate the cycle count, perhaps aided by correlating with the power trace, in order to synchronise target and attack host. Since it is easier to inject faults on multi-cycle instructions that perform the pre-fetching stage on their last execution cycle like `BREQ` does [5], one could target `BREQ` in order to alter the resulting branch. Clock glitching on single-cycle instructions has the effect of replacing the following instruction with a `NOP` [5] and hence one could target the `OUT` instruction to skip over the `INC` instruction and continue outputting the memory. The experimental set-up for this according to Balasch et al. [5] and Pareja [19] involves using an FPGA connected to the target MCU for generating the clock signal (and injecting the glitch) and a computer for controlling the FPGA. The requirements they note is that in order to have a sufficient glitch resolution one should use an FPGA with at least double the operating frequency of the target MCU.

For obtaining cryptographic material a similar approach could be taken, where one could effectively `NOP`-out further rounds of the cryptographic algorithm and brute-force the results of the first few [5][23] or corrupt multiplication results when modulo products are being computed and make factorization easier[2]. This could be done in either a while-loop implementation of the cryptographic rounds or an unrolled version of the code [5]. Alternatively one could mount a DPA attack on the device if they are after the cryptographic keys rather than skipping authentication. For a DPA key extraction one would need resistors and an oscilloscope in order to measure the power consumption of the device [23] [14] [17] and would then need to record a large number of power traces, keeping in mind that it is important for traces that correspond to the same action to be perfectly aligned time-wise [17]. After this, a number of guesses for particular bit-values of the key is performed, run through a model emulating the attacked device and the results are correlated with the actual traces and can reveal show if that guess was correct [17]. Alternatively, one could use available tools and software for performing DPA [18].

It should be noted that Balasch et al. [5] observed a stuck-to-zero trend while corrupting multi-cycle instructions and even though the exact glitch period required to induce a fault might vary between boards, the faults induced are deterministic, i.e. for a given chip and a given glitch period the same fault will always occur. Balasch et al. Furthermore, the result of the glitch is highly dependent on the currently executing instruction, the pre-fetched instruction and the glitch period, but an invalid opcode will be treated as a `NOP` by the CPU [5].

TABLE 2: AVR assembler of a typical while loop of the form `while(i < n) { code ;}`.

```
            ...other code ...
            LDI R16, 4              ; initialize n, 1 cycle
            LDI R17, 0              ; initialize i, 1 cycle
condition :                         ; label
            SUB R16, R17            ; n-i, result affects ZF bit
                                    of SREG.
                                    ;
            BREQ leave              ; if ZF=0 branch to loop,
                                    ; 1 cycle if false, else 2
loop :                              ; label
            ...code loading from
            memory to variable ...  ; k cycles
            OUT $18, R17            ; output to port B
            INC R17                 ; increment i, 1 cycle
            JMP condition           ; 3 cycles
leave :                             ; label
            ...other code ...
```

6 Discussion

In this report we reviewed the current attacks against MCUs and the defences developed in response. We also reviewed the relevant architectural information of the ATmega644 and then proposed a possible non-invasive clock-glitch attack on the ATmega644 board by putting together information from research done on systems with almost identical specifications. Although the exact details for performing a successful glitch, like the glitch period, would need to be found by experimentation on the target board [5], once found they would work consistently [5] and from there the glitching results would need interpretation by the attacker in order to achieve their goal. Although conceptually simple in terms of implementation and theoretic foundation [23] [5], interpretation of the results is itself a challenging task [5] because of the architecture of the MCU. Since the modified Harvard architecture of the AVRs accesses both memory and instructions at the same time, the glitch will have an effect on multiple pipeline stages and its effects are influenced by the particular instructions being handled at the time and hence creating an exploitable glitch is more involved than commonly perceived [5].

As in traditional computer systems *Security is Hard* for MCUs as well. It is difficult for an individual to completely asses the security of their board with companies making exaggerated claims about the security of their products [23], keeping information hidden and not always being rigorous with their analysis [23]. Individuals can build their home probing station for probing attacks for a few thousand dollars [23] [16] and attempt to break their system themselves or can hire firms that

do this professionally (Riscure [15], IC-Crack [1]) for vulnerability testing, but its security status will be assessed based on what current knowledge. Even if the progress of technology and the shrinking size of ICs make the process of micro-probing harder, but not impossible [23] [10], it would be reasonable to expect attackers to go after semi or non-invasive attacks, side-channels, implementation and protocol failures or bugs in very intelligent ways. The technological progress and the arms race between attackers and defenders means that each other will make their opponent more sophisticated. The majority of researchers in the MCU and smart-card field seem to agree that no system is unbreakable and one can only harden their system enough to make the effort of breaking it unbearable to those they wish to protect against [2] [23], i.e. make their product uninteresting to attack in terms of money and time. A complete security assessment in order to harden the device would mean that all possible threat scenarios should be considered and the security limitations of the device made perfectly clear [14]. It is a well-established notion in the security industry that security does not *add* anything to a product but is rather a huge expense to avoid a *potentially* bad situation, with the effect that companies tend to not place as much focus on securing their products. Securing a product means longer release cycles, lengthier and more expensive testing, manufacturing and research phases [14] and usually one has to choose between usability and security.

REFERENCES

[1] Atmel MCU Reverse Engineering. http://break-ic.com/microcontroller_read/Atmel_mcu_reverse_engineer.htm.

[2] Ross Anderson and Markus Kuhn. Tamper resistance: A cautionary note. In *Proceedings of the 2nd USENIX Workshop on Electronic Commerce - Volume 2*, pages 1–11, Oakland, California, November 1996. USENIX.

[3] Ross Anderson and Markus Kuhn. Low cost attacks on tamper resistant devices. In Bruce Christianson, Bruno Crispo, Mark Lomas, and Michael Roe, editors, *Security Protocols*, volume 1361 of *Lecture Notes in Computer Science*, pages 125–136. Springer Berlin Heidelberg, 1998.

[4] AVRFreaks. Desingn Note #20 : Understanding AVR Fuses and Lock bits. Application Note, AVRFreaks, 5 2002. http://www.avrfreaks.net/modules/FreaksFiles/files/382/DN_020.pdf.

[5] Josep Balasch, Benedikt Gierlichs, and Ingrid Verbauwhede. An in-depth and black-box characterization of the effects of clock glitches on 8-bit mcus. In *Proceedings of the 2011 Workshop on Fault Diagnosis and Tolerance in Cryptography*, FDTC '11, pages 105–114, Washington, DC, USA, 2011. IEEE Computer Society.

[6] R. Clayton. Extracting a 3DES key from an IBM 4758. http://www.cl.cam.ac.uk/~rnc1/descrack/ibm4758.html, November 2001. Accessed: 2014-05-04.

[7] Jr. Cole, E.I. Beam-based defect localization techniques. In *Microelectronics Failure Analysis*, Materials Park, Ohio, October 20011. ASM International.

[8] Atmel Corporation. ATmega 644/V Datasheet. Datasheet 2593OAVR02/12, Atmel Corporation, 2012. http://www.atmel.com/Images/doc2593.pdf.

[9] Atmel Corporation. Atmel AVR231: AES Bootloader. Application Note 2589E-AVR-03/12, Atmel Corporation, 2012. www.atmel.com/Images/doc2589.pdf.

[10] Peter Gutmann. Data remanence in semiconductor devices. In *Proceedings of the 10th Conference on USENIX Security Symposium - Volume 10*, SSYM'01, pages 4–4, Berkeley, CA, USA, 2001. USENIX Association.

[11] Philipp Hof. A Primer To Microcontrollers. http://www.elec.canterbury.ac.nz/PublicArea/Staff/hof/p10-embed/p10-tutorial/index.html, March 2014. Accessed: 2014-05-04.

[12] Andrew Huang. Hacking the PIC 18F1320. http://www.bunniestudios.com/blog/?page_id=40, March 2014. Accessed: 2014-05-04.

[13] Ilya Kizhvatov. Side channel analysis of avr xmega crypto engine. In *Proceedings of the 4th Workshop on Embedded Systems Security*, WESS '09, pages 8:1–8:7, New York, NY, USA, 2009. ACM.

[14] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential Power Analysis. In Michael J. Wiener, editor, *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer, 1999.

[15] S. Lawler. Power Analysis with Riscure. http://dontstuffbeansupyournose.com/2014/02/11/power-analysis-with-riscure/, February 2014. Accessed: 2014-05-04.

[16] Philipp Maier and Karsten Nohl. Low-Cost Chip Microprobing. Accessed: 2014-05-04, 2014.

[17] Stefan Mangard, Elisabeth Oswald, and Thomas Popp. *Power Analysis Attacks: Revealing the Secrets of Smart Cards (Advances in Information Security)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2007.

[18] Colin O'Flynn and Zhizhang (David) Chen. Chipwhisperer: An open-source platform for hardware embedded security research. Technical report, 2014. http://eprint.iacr.org/2014/204.pdf.

[19] Ramiro Pareja Veredas. Fault injection attacks on microcontrollers: clock glitching tutorial. http://www.t4f.org/articles/fault-injection-attacks-clock-glitching-tutorial/.

[20] Ramiro Pareja Veredas. Ultra-low Cost IC Decapsulation. http://www.t4f.org/articles/ultra-low-cost-ic-decapsulation/.

[21] K. C. Saraswat. EE311 Notes : Deposition & Planarization. Accessed: 2014-05-04, 2014.

[22] Sergei Skorobogatov. Low temperature data remanence in static RAM. Technical Report UCAM-CL-TR-536, University of Cambridge, Computer Laboratory, June 2002.

[23] Sergei P. Skorobogatov. Semi-invasive attacks – A new approach to hardware security analysis. Technical Report UCAM-CL-TR-630, University of Cambridge, Computer Laboratory, April 2005.

[24] Alan Trevennor. *Practical AVR microcontrollers games, gadgets, and home automation with the microcontroller used in Arduino*. Apress, Berkeley, CA New York, 2012.

[25] Bulent Yener and Andrew Zonenberg. CSCI 4,74 / 6974 : Hardware Reverse Engineering. Rensselaer Polytechnic Institute, lecture slides at : http://security.cs.rpi.edu/courses/hwre-spring2014/, 2014.