Firmware Protection and Attacks Against ATmega Microcontrollers

Dionisio Perez-Mavrogenis

Abstract—This paper will review some of the attacks used against microcontrollers in order to obtain access to cryptographic material (keys or algorithms) or the firmware of the device and the defence techniques developed in responce, in an attempt to show the co-evolutionary nature of the attack-defence ecosystem. We outline a possible attack against the Atmel ATmega644 synthesized by techniques applied to similar devices and conclude by discussing why completely securing a microcontroller is impossible and that manufacturers should instead make their product uninteresting to attack, where interest is measured both monetarily and by the effort required.

Index Terms—Microcontrollers, firmware dumping, firmware protection, hardware reverse engineering, microprobing, hardware attacks, microcontroller protection

review wherever I say working attack against the atmega to possible attack. reduce structure of things, merge sentences together. also make sure that spaces between Fig. or Table and number are unbreakable using the tilde. also consistently name abbreviations, could have abbreviation table. mention book pages from books from which info was taken. rename decaping to Decapsulation

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). 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 achieve higher performance 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 a lot of parties (including the government and the military).

A distinction between ordinary and secure MCUs should be made[19] and, due to the sophistication of the protective mechanisms and the attacks, a broad classification of attackers as[2]:

- **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 is usually tight.

Firmware tampering has a number of consequences, the most obvious being an attacker 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]. 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 coded added by the attacker in order to accomplish their malicious intents. Furthermore MCU firmware might contain government or industrial secrets whose integrity and value is more valuable than the integrity of the MCU.

Current attack technologies and methods will be reviewed (Sec. 3) and the defensive techniques (Sec. 4 developed to counter these. These are described to give a (brief) overview of the current state of affairs and, once the ATmega MCU features have been reviewed (Sec. 2), a working attack against the ATmega will be presented (Sec. 5). This paper will conclude (Sec. 6) by summarising the material presented and explaining why securing a device is hard.

2 The AVR MCU Series

maybe ditch the 1284. if so, rewrite to make it refer to one board only

The Atmel 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[20]. The models have varying degrees of hardware capabilities and large operating voltage windows in order to accommodate demand and integrate well with peripherals[2][3]. Developing software for an AVR is easy as the AVRs benefit from

1. The act of adding code to a system without the user's knowledge or approval, usually to accomplish nefarious tasks.
2. info:https://www.newbiehack.com
3. info:http://www.atmel.com/v2PFResults.aspx

the `avr-libc` high-performance C run-time 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 APIs for interacting with the AVR and the developers can choose from a wide variety of programmer units available for working with the AVRs[20]. We chose to focus on the ATmega644 as it is a representative of a widely used low-cost microcontroller [5][4].

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 Neuman 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 that lives in the same memory region (Von Neuman). The 644/1284 implement a modified Harvard architecture for both power and computational efficiency, being designed to access multiple memory locations simultaneously, enabling them to execute an instruction per cycle, as shown in Fig. 2. The operating voltages can vary between 1.8V and 5.5V (maximum operating frequency 20 MHz)[8].

The 644 is equipped with an2 Kb of EEPROM, 64Kb of flash memory,4Kb of SRAM, a large number of general purpose registers and a large number of I/O registers (in order to be able to perform I/O) and all memory (including I/O memory mapped images) is linear, i.e. it follows the flat memory model. The flash memory is separated into two regions, the bootloader section and application code section. The boundary between the two sections can be configured by programming the appropriate fuses, and the page size can also be configured that way as well. Both sections hold code, however code residing in the bootloader section can execute a special instruction (`SPM`[5]) which allows the bootloader code to write to *any* section in the flash memory and hence possibly modify itself(designed for purposes such as 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 preferences the application code (or the bootloader) may need, having an average lifespan is 100,000 write cycles per page. The SRAM is volatile storage and is used as the stack and heap for the software

4. maybe remove last sentence/trim the whole thing down ?
5. `SPM` = Store Program Code, assembly instruction for the AVR.

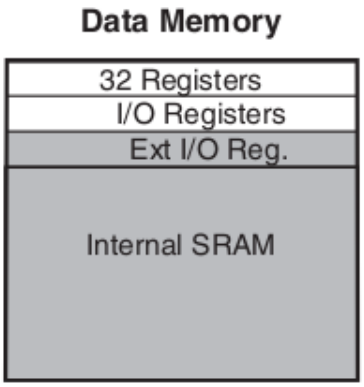


Fig. 1: SRAM layout for ATmega644 (source: [8]).

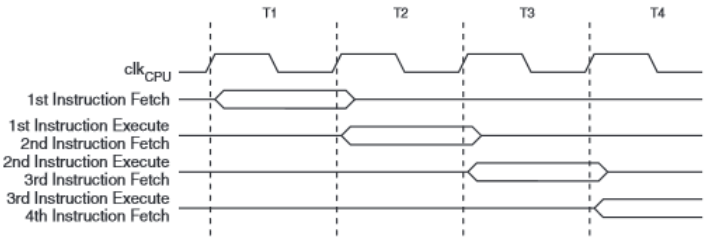


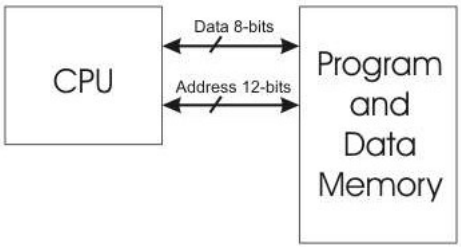
Fig. 2: 2-stage pipeline of the ATmega644 (source: [8]).

(either application code or bootloader code) as well as 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.

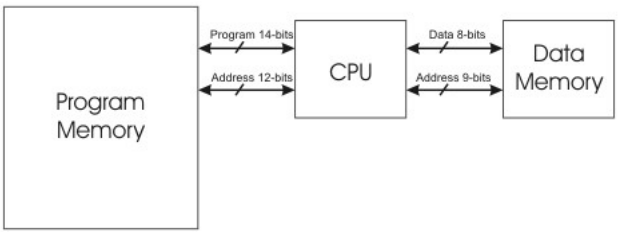
2.2 ATMega Security Features

The AVR ATmega644/1284, even though not meant to be secure hardware modules, posses certain security features. In particular, each board provides 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 via the `SPM` instruction). 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 Lock bits.

The erasing is performed with the sequence of events presented above and this is important, as one does not want to remove the access protection before removing all sensitive 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 relatively expensive as an operation) this approach makes



(a) Typical Von Neuman architecture.



(b) Strict Harvard architecture.

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

TABLE 1: Security lock bits offered by the ATmega644. BLB stands for Boot Lock Bit and LB for Lock Bit.

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

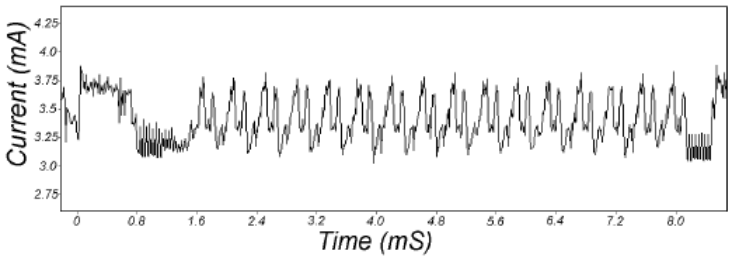


Fig. 4: Power trace during a DES encryption, where the 16 rounds are clearly visible.(source: [14]).

sense as the ultimate goal is to preserve the intellectual property on the board rather than the board itself.

Table 1 provides 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 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 [8].

3 Attacks on Hardware

make this flow with previous paragraphs. make sure you mention that you have to know what chip you are trying to reverse ([19] and hardware reveng course). for chemicals see for delayering see :http://siliconpr0n.org/wiki/doku.php?id=chemical:start , http://siliconpr0n.org/wiki/doku.php?id=delayer:start

A distinction between *passive* and *active* attacks should be made. In the former the attacker simply monitors the chip's normal operation and tries to infer the input-output mapping whereas in the latter case the attacker actively manipulates either the chip or its operating environment with the aim of obtaining insight on the chips inner workings.

Attacks on MCUs may attempt to recover a number of artefacts, including cryptographic keys the and firmware and do not need to necessarily attack the hardware itself but can exploit flaws in algorithmic design and implementation and protocol failures or inter-component communication patterns[2][14], obtain information by corrupting the memory or exploiting memory remanence[19][10].

The following discussion closely follows [19].

3.1 Non-Invasive Attacks

mention other non-invassive attacks as well and SCA, such as exposure to abnormal temperatures, exposure to radiation or using lasers damage/alter/heat specific portions of the chip. make this more technical. [15] Non-invasive attacks are attacks which require no depackaging or special preparation of the chip and hence attacks under this category leave little tamper evidence behind. These attacks might be very time consuming to find and are not guaranteed to be successful, but are very easy and cheap to replicate once found. Furthermore, non-invasive attacks could target badly implemented communication, bugs or security protocols in order to bypass security restrictions.

3.1.1 Power Analysis

need sampling equipment with at least double the operating frequency to get a measurement per clock cycle and also check out Correlational Power Analysis [**?**][15] . could also include image from DPA paper showing DES rounds. also say set up, how and where you place the resistors. see DPA paper again. Also mention EM radiation detection.

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 widely available tools.

Simple Power Analysis(SPA) involves direct observation of the MCU when it performs cryptographic operations and can leak information about both the keys

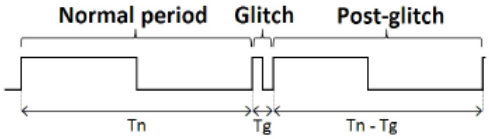


Fig. 5: Illustration of the principle of a clock-glitch attack.(source: [5]).

and the cryptographic operations themselves (i.e. nature or structure of the algorithm)[14][3].

Differential Power Analysis(DPA) extracts sensitive information by using statistical techniques on very large traces. The techniques involves obtaining 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 differences in power consumption[14][3].

One can generally avoid noise in their power measurements by sampling the voltage (usually) on the ground line[19].

3.1.2 Glitch and Fault Injection Attacks

Glitches and faults are achieved by exposing the device in operating conditions that it was not meant to operate in and attempt to exploit undefined behaviour of the MCU[19][13]. Although inducing a fault is easy, inducing an exploitable fault is hard but can be achieved by systematic search sergei:thesis[5][15].

Power glitches and *clock glitches* aim to make the CPU skip or execute incorrect instructions by applying transients. This attack 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. Clock glitches are mainly aimed against software-based protection mechanisms, affecting CPU operation by supplying the CPU with incorrect data. Power glitches work by supplying either too much power or too little, shifting transistors' threshold and causing flip-flops to read their state incorrectly. Power glitches need to be carefully synchronised with the internal clock and prolonged attacks might damage the board. Glitch attacks are especially dangerous as they may abuse the program counter in order to map out the memory[5][2][19].

3.1.3 Data Remanence

explain more, make more technical. also cite [6] who also talks about memory remanence Prolonged exposure of SRAM cells to the same values can make the cells 'remember' their state, due to material properties and stress[10]. Furthermore, cooling the memory down can prolong the time for the data to leave the memory [10][18][19]. If, for example, a start-up routine always writes security keys to the same memory location, after some time they key will be recoverable by looking at the physical state of the memory or by cooling down the chip, starting it and then reading the memory.

EEPROM suffers as well, but to a lesser extent, as material-wise one can only tell virgin-cells from used cells[19].

3.1.4 Timing Attacks

Timing attacks exploit the software implementation of cryptographic algorithms. Compiler optimisations (avoiding unnecessary branches, register and cache usage) and other implementation choices 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.

Different instructions take different time to execute(e.g. `MOV eax,[eax]` is considerably slower than `INC eax`) and thus one could collect timing information for various input messages and systematically deduce the correct 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 `INC eax` and correlating timing and power consumption information would reveal this.

3.2 Semi-Invasive Attacks

Semi-invasive attacks require depackaging of the chip but do not destroy the passivation layer as no electrical contact with the chip is needed. Semi-invasive attacks can be automated and can yield results faster and cheaper than invasive attacks.

3.2.1 UV Light Exposure

Older chips and chips that are designed to withstand low-cost non-invasive (i.e. no depackaging) attacks are susceptible to having parts of the memory altered if it is exposed under UV light. Security fuses that prevent read-back of the memory could have their state reset by sufficient exposure under UV light. The attacker must locate the security fuse though, which can be very tricky.

3.2.2 Imaging Attacks

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 though multiple layers and hence protective metal meshes(discussed in Section 4) or normal chip layers are avoided. On some chips it is possible to extract ROM contents via this technique by directly observing the memory. An alternative to IR light would be the

use of lasers for imaging. 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[19].

3.3 Invasive Attacks

Invasive attacks require direct access to the board's surface and as a result destroy the packaging in the process, therefore leaving tamper evidence[19][21]. Invasive attacks usually aim to understand how a MCU works and then develop cheaper non-invasive or semi-invasive attacks for that chip, as invasive attacks are laborious, require expensive equipment and highly skilled attackers[19].

3.3.0.1 Exposing the die surface: This usually involves destroying the packaging by using chemicals or drilling(or other methods). While this is a process that is not very complicated[19], one might have trouble finding the chemicals required. An alternative for depackaging the chip is to send it to a failure analysis lab[12].

3.3.0.2 Reverse engineering: both hardware and the software. Hardware reverse engineering requires using reflected light microscopes or SEM (Scanning Electron Microscope) for constructing a complete image of the surface. maybe present layer removal techniquesLayer removal might be required if deeper layers are not visible in order to have a complete view of the device. Software reverse engineering can be accomplished when one has obtained access to the memory.

3.3.0.3 Micro-probing and Modification: [16] estimate around $2000 for optical microscope + micropositioners + moving base + electronics , [19] estimates about the same. In micro-probing sub-micrometer thickness probes are used to establish contact with the bus lines in order to observe and manipulate bus signals. To achieve reliable results a micro-probing workstation[6] is used, consisting of a microscope, micro-positioners for the probes, a movable base and a test socket to place the chip. In order to establish contact with the bus lines the passivation layer should be removed, usually done with UV or green lasers, and access to bus lines of deeper layers can be achieved by using a FIB workstation.

Modifications to the MCU's components (adding new interconnects or destroying circuits) are not always necessary, but could prove useful[3]. For chip modifications to be successful, the attacker must be sophisticated and must have at least partially reverse-engineered the board.

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[19][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 [7] or even try to make their MCUs look like ASICs [8] in order to scare away potential attackers by making their product appear more secure[19][21], as this will make information gathering for he chip trickier. A further step in making the MCU harder to analyse is hardware obfuscation by making the interconnects into a maze [21] 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[19].

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[19], while still being able to safely deliver firmware update[9]. Further protection is introduced by avoiding side-channel leakage by masking

6. all moving components should have micrometer precisionprovide citation

7. Related legal issues discussed in Sec. 6hardware reveng legal issues

8. abbreviation?

all relationships between data input and power consumption, thermal and electromagnetic radiation and timing relationships[14][19]. 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][19] 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][9]. 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][21][13]. 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 [21][19] 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 [19] to using acids and other chemicals [21][19] 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[21], resetting of the device [19] 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[19][21].

The next level of protection came when radiation attacks became known[19], 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[19][21]. 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[21], 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 fanalysis of the die[21], making the localisation of critical components harder and are commonly found in smart-cards[19] and SIM cards as well[21]. 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[19], as well as chemical or mechanical planarization[10] of the layers of the MCU[19]. 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[19]. 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[21] or one could paint over the sensors with black paint[19]. Tampering and micro-probing is in reality being made harder by the shrinking sizes of the various components

9. maybe read this about CPA as well https://www.iacr.org/archive/ches2004/31560016/31560016.pdf

10. planarization techniques here http://www.stanford.edu/class/ee311/NOTES/Deposition_Planarization.pdf

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 [19][21].

5 Attacking The ATmega

In this section we present the suggested 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 researches have succeeded in bypassing the protection imposed by the AVR family in numerous ways. All researches used 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. [17] against an ATmega328P and Kizhvatov [13] managed to retrieve AES and DES cryptographic keys from an ATmega16 and ATXmega128A1. Furthermore, Skorobogatov [19] 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 descent 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 taking a maximum of 5 cycle 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. Subtleties that should be noted 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][19][13].

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. In our attack scenario suppose 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 [5] like `BREQ` does, on 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.

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 condition 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 ...
```

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][19] 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]. Another successful approach for obtaining access to cryptographic material would be power analysis, either SPA or DPA. For a DPA key extraction we would feed the AVR a big number of plain-texts and measure its power consumption, write some more

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. [5] also observed that the result of the glitch is highly dependent on the current instructions executing, the pre-fetched instructions and the glitch period, but even if the glitch results in an invalid op-code being fed to the CPU then the CPU will treat it as a `NOP` .

6 Evaluation

In this report we 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 [19] [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]. mention equipment needed?

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 [19], keeping information hidden and not always being rigorous with their analysis [19]. Individuals can build their home probing station for probing attacks for a few thousand dollars [19] [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 [19], 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 smartcard 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] [19]. 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 [19].

References

[1] Atmel MCU Reverse Engineering. http://break-ic.com/microcontroller_read/Atmel_mcu_reverse_engineer.htm. Accessed: 2014-05-04.

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

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

[19] 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.

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

[21] 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.