Mind the Gap: Towards Secure 1st-order Masking in Software

Kostas Papagiannopoulos[1]* and Nikita Veshchikov[2]**

[1] Radboud Universiteit, Nijmegen, Netherlands
[2] Quality and Security of Information Systems, Département d'informatique, Université Libre de Bruxelles, Belgium

Abstract. Cryptographic implementations are vulnerable to side-channel analysis. Implementors often opt for masking countermeasures to protect against these types of attacks. Masking countermeasures can ensure theoretical protection against value-based leakages. However, the practical effectiveness of masking is often halted by physical effects such as glitches and distance-based leakages, which violate the *independent leakage assumption* (ILA) and result in security order reductions. This paper aims to address this gap between masking theory and practice in the following threefold manner. First, we perform an in-depth investigation of the device-specific effects that invalidate ILA in the AVR microcontroller ATMega163. Second, we provide an automated tool, capable of detecting ILA violations in AVR assembly code. Last, we craft the first (to our knowledge) *"hardened"* 1st-order ISW-based, masked Sbox implementation, which is capable of resisting 1st-order, univariate side-channel attacks. Enforcing the ILA in the masked RECTANGLE Sbox requires 1319 clock cycles, i.e. a 15-fold increase compared to a naive 1st-order ISW-based implementation.

Keywords: Masking, AVR, verification tool, simulator, independent leakage assumption, distance-based leakage, RECTANGLE, SCA

1 Introduction

Nowadays, the explosive growth of the "Internet of Things" (IoT) is reshaping modern society, pervading its infrastructure and communications. The rapid price drop in IoT components has transformed everyday products, enhancing them with network connectivity and information exchange capabilities. Amidst this new status quo, devices, ranging from cheap sensors to expensive vehicles, are required to maintain a heightened level of theoretical and physical security.

* The work described in this paper has been supported by the Netherlands Organization for Scientific Research NWO under project ProFIL (628.001.007).

** This work is the result of a short term scientific mission that has been supported by ICT COST Action IC1204: "Trustworthy Manufacturing and Utilization of Secure Devices"

For instance, side-channel attacks (SCA) allow adversaries to recover sensitive data, by observing and analyzing the physical characteristics and emanations of a cryptographic implementation [19]. Such physical attacks motivated research towards countermeasures that perform noise amplification, thus hindering the adversary's recovery capabilities. A common choice for provably secure, noise-amplifying software countermeasure is masking [9, 18]. Masking employs secret-sharing techniques that establish theoretical security against the value-based leakage model. Rephrasing, masking secures implementations against adversaries that can only extract information about the value being processed at a given time. This underlying assumption is often referred to as the *independent leakage assumption* (ILA) [24]. Unfortunately, the exact values under manipulation are not always visible at a given layer of abstraction, e.g. at assembly code and such a limited adversarial model is not applicable in many practical, software-based scenarios. For instance, devices often exhibit *distance-based leakages*, which can reduce the security of the masking countermeasure [1, 14]. Likewise, coupling effects [24] and glitches [20] can pose similar security hazards.

This work attempts to bridge the gap between theory and practice in the masking countermeasure with the following threefold contribution. First, we investigate several effects that violate ILA in an ATMega163 microcontroller and subsequently, we establish solutions that mitigate these issues. Second, we use this knowledge in order to build an assembly-oriented tool that is capable of detecting ILA violations in AVR-based masked implementations. Third, assisted by the developed tool, we craft the first (to our knowledge) 1st-order masked implementation in ATMega163 that is capable of resisting 1st-order, univariate attacks. In other words, we enforce the ILA in order to severely limit the informativeness of 1st-order leakages, forcing the adversary to resort to 2nd-order attacks. As a proof of concept, we develop a "hardened" 1st-order, ISW-based [18], bitsliced Sbox for the RECTANGLE cipher [35]. The "hardened" implementation requires 1319 clock cycles, a 15-fold increase compared to a "naive" 1st-order, ISW-based, bitsliced Sbox of the same cipher.

The rest of this paper is organized as follows. In Section 2, we provide preliminaries w.r.t. masking, the experimental setup and the evaluation techniques we employ. In Section 3 we offer a detailed description of all the ILA-breaching effects that we have identified in ATMega163. Section 4 discusses the development of the assembly verification tool. Section 5 details the construction of a "hardened" RECTANGLE, 1st-order masked Sbox for ATMega163. We conclude and discuss future work in Section 6.

2 Background

2.1 Boolean Masking & Order Reduction

Chari et al., Goubin et al. and Messerges [9, 15, 21] were among the first to suggest splitting intermediate values with a secret sharing scheme, in order to force attackers to analyze higher-order statistical moments. Analytically, a

dth-order Boolean masking scheme splits a sensitive value x into $d+1$ shares $(x_0, x_1, \ldots, x_d)$, as shown below.

$$x = x_0 \oplus x_1 \oplus \cdots \oplus x_d \tag{1}$$

The shares $(x_0, x_1, \ldots, x_d)$ are also referred to as the $(d+1)$-family of shares corresponding to x [26]. Given that the ILA holds and assuming sufficient noise, it has been shown that the number of traces required for a successful attack grows exponentially w.r.t. the order d [9, 23]. Several implementation options have been suggested for the masking countermeasure, ranging from lookup-table techniques [11, 32] to GF-based circuits [8, 16, 18, 26].

In parallel with the development of masked implementations, side-channel research focused on the practical evaluation of the countermeasure. Balasch et al. [1] put forward the concepts of value-based and distance-based leakages, as well as the notion of order reduction. We briefly restate their definitions below.

Value/Distance-based leakage function A leakage function $L(.)$ consists of a deterministic part $L_d(.)$ and random additive noise N. The leakage function is *value-based* if $L_d(.)$ can only take arguments from the set of intermediate values produced by the masking scheme. The leakage is *distance-based* if $L_d(.)$ can take arguments from the set that contains all possible pairwise combinations of intermediate values. The combination can imply operations such as XOR, concatenation, etc.

Order-reduction theorem A dth-order secure masking scheme under value-based leakages is $\lfloor \frac{d}{2} \rfloor$th-order secure under distance-based leakages.

The applicablity of the order-reduction theorem has been verified experimentally by Balasch et al. [1] for orders $d = 1, 2$ in AVR-based and 8051-based devices. De Groot et al. [14] have verified experimentally the theorem's applicability for orders $d = 1, 2$ in the ARM Cortex-M4.

2.2 Experimental Setup & Evaluation

The implementation and SCA evaluation is performed on a smartcard equipped with an 8-bit, AVR-based ATMega163 microcontroller[3]. The device features a 4.4 MHz clock, 1024 bytes of SRAM and 17 Kbytes of Flash memory. The acquisition of power traces is carried out using the Riscure PowerTracer[4] and the Picoscope 5203 oscilloscope. The sampling rate is set at 31.5 MSamples/sec and the only post-processing applied is signal alignment.

The evaluation of the *actual* security order of a masking scheme is, in general, an open problem. We often face the *limited attack scope*, i.e. a given attack may not be able to exploit the available leakage due to e.g. an unsuitable choice of intermediate values or an incorrect power model. To address this problem, generic side-channel distinguishers and extensive profiling techniques have been

[3] `http://www.atmel.com/images/doc1142.pdf`

[4] `https://www.riscure.com/security-tools/hardware/power-tracer`

developed [3, 28, 33]. In this work, we opt for the leakage detection methodology [10] which prioritizes leakage detection over leakage exploitation, speeding up certain evaluation aspects. In detail, we employ the random vs. fixed, non-specific, 1st-order t-test. We perform a random vs. fixed acquisition and obtain two distinct tracesets S_{fixed} and S_{random}, under the same encryption key. The input plaintext for S_{fixed} is set to a fixed value, while for S_{random}, the input is uniformly random. The implementation receives the fixed or random plaintext in a non-deterministic and randomly-interleaved way (as recommended by Schneider et al. [27]). Following the data acquisition, the 1st-order t-test will assess whether the two sets S_{fixed}, S_{random} stem from the same population, using the following statistical test.

$$\begin{aligned} H_{null}: &\quad \mu_{fixed} = \mu_{random} \\ H_{alt}: &\quad \mu_{fixed} \neq \mu_{random} \end{aligned} \tag{2}$$

$$w = \frac{\mu_{fixed} - \mu_{random}}{\sqrt{\frac{\sigma^2_{fixed}}{n} + \frac{\sigma^2_{random}}{m}}} \qquad \upsilon = \frac{\left(\frac{\sigma^2_{fixed}}{n} + \frac{\sigma^2_{random}}{m}\right)^2}{\frac{\sigma^4_{fixed}}{n^2(n-1)} + \frac{\sigma^4_{random}}{m^2(m-1)}} \tag{3}$$

Parameters μ_x and σ_x^2 are the estimated mean and variance of set x; n and m denote sizes of sets S_{fixed} and S_{random} respectively. The null hypothesis H_{null} is rejected at a given level of significance α (often set to 0.99999), if $|w| > t_{\alpha/2,\upsilon}$, where $t_{\alpha/2,\upsilon}$ is the value of the Student t distribution with υ degrees of freedom. In the evaluation context, rejecting H_{null} implies leakage detection, i.e. potential evidence of an ineffective masking scheme.

In this paper, we will use the t-test as a detection tool w.r.t. ILA-breaching effects and their solutions (see Section 3). Still, we will also employ 1st-order CPA methods [7] in order to demonstrate the exploitability of such effects. In order to reduce the computational cost of the evaluation, we use the memoryless formulas suggested by Schneider et al. [27] and the incremental approach for CPA by Botinelli et al. [6].

3 ILA-Breaching Effects

In this section, we present three effects identified in the ATMega163 microcontroller that breach ILA and pose a hazard to any masking scheme's security. Analytically, the effects below demonstrate that independent computations *do not* necessarily lead to independent leakages and thus, the order-reduction theorem can become applicable.

Every effect (Sections 3.1, 3.2 and 3.3) is described as a standalone, assembly-based scenario that manipulates two 4-bit shares x_0, x_1 originating from the sensitive, key-dependent, 4-bit value x, such that $x = x_0 \oplus x_1$. The shares x_0, x_1 are always manipulated in a theoretically sound manner, adhering to the masking scheme's requirements, i.e. we never combine the shares directly (e.g. via an exclusive-or instruction `eor x0, x1`).

For all the described scenarios, that are theoretically sound, we show experimentally that ILA is not fulfilled by employing 1st-order, univariate techniques. Namely, we perform correlation-based analysis [7], computing the correlation coefficient ρ between the Hamming weight of the sensitive, key-dependent value x and the experimentally acquired traceset. To maintain a wide attack scope, we also use the leakage detection methodology [10, 27] and compute the 1st-order, random vs. fixed t-test. We conclude every scenario by suggesting possible solutions that enforce ILA. Restating Balasch et al. [1], as we are always limited by the traces at hand, we cannot rule out the existence of 1st-order leakages, yet we establish that their informativeness is limited compared to 2nd-order leakages in the target device. Note that extra care is taken in order to assess all effects independently, i.e. we use the suggested solutions so as to isolate the effect under discussion from the rest.

The analyzed effects can manifest in several data storage units (e.g. registers, SRAM/Flash memory cells, I/O buffers, etc.) and may relate to different instructions of the AVR ISA[5], leading to a very large number of potential scenarios. In order to maintain a feasible scope, we limit our discussion to storage units and instructions that are often encountered in the context of cryptographic implementations, i.e. SRAM memory accesses and logical instructions.

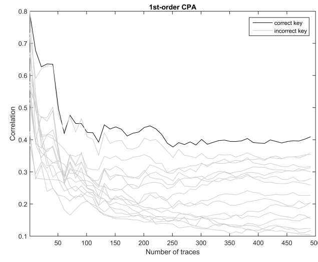
3.1 Overwrite Effect

The overwrite effect is observable when a share gets overwritten by a different share from the same family. For instance, if share x_0 in a data storage unit (register, memory cell, etc.) gets overwritten by share x_1, then the power consumption correlates with the number of bits switched i.e. $x_0 \oplus x_1$. This effect was observed by Daemen et al. [30] and later revisited by Coron et al. [12].

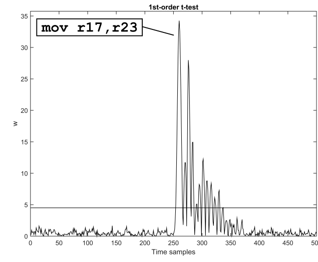
Below, we address the most common situations in which overwriting arises during a cryptographic implementation. We perform two experiments: a register-based overwrite via the instruction `mov x0, x1`, and a memory-based overwrite via the instruction `st SRAM_x0, x1`. The experiments are described in Listings 1.1 and 1.2. Their analysis follow in Figure 1.

We confirm that overwriting is indeed an ILA-breaching effect, manifesting both in registers and SRAM memory. Note that the exploitability of the effect varies according to the data storage unit: in ATMega163, register-based overwriting can be exploited with roughly 500 traces (1a), while memory-based requires at least 40k traces (1c). Preventing register and memory-based overwrites is straightforward: the corresponding register (or memory cell) needs to be cleared in advance.

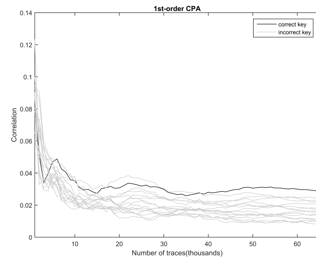
[5] `http://www.atmel.com/images/Atmel-0856-AVR-Instruction-Set-Manual.pdf`



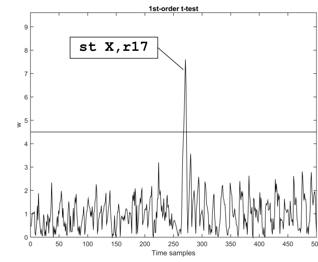
(a) Register overwrite, 1st-order CPA, HW model, 500 traces.



(b) Register overwrite, 1st-order t-test, 5k random vs. 5k fixed.



(c) Memory overwrite, 1st-order CPA, HW model, 65k traces.



(d) Memory overwrite, 1st order t-test, 50k random vs. 50k fixed.

Fig. 1: Register/memory-based overwrite effects

```
1 ;share x0 in r17
2 ;share x1 in r23
3 mov r17,r23
4 ;
5 ;
```

Listing 1.1: Register overwrite experiment.

```
1 ;share x0 in SRAM 0x0080
2 ;share x1 in r17
3 ldi r27,0x00
4 ldi r26,0x80
5 st X,r17
```

Listing 1.2: Memory overwrite experiment.

3.2 Memory Remnant Effect

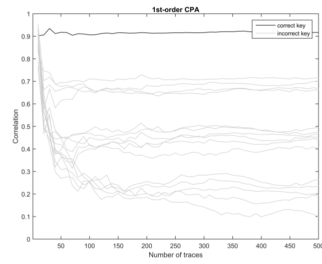
The memory remnant effect is a leakage originating from consecutive SRAM accesses to shares of the same family. Assume that shares x_0, x_1 are stored in SRAM cells and get accessed sequentially. Naturally, the first access leaks share x_0 (value-based leakage), yet it also creates a “remnant” of x_0. The second access will leak the transition of the share x_1 and the remnant x_0, reducing the security.

```
1  ;share x0 in 0x0080
2  ;share x1 in 0x0090
3  ldi r27, 0x00
4  ldi r26, 0x80
5  ld r17, X
6  ldi r27, 0x00
7  ldi r26, 0x90
8  ld r20, X
9  ;
10 ;
```

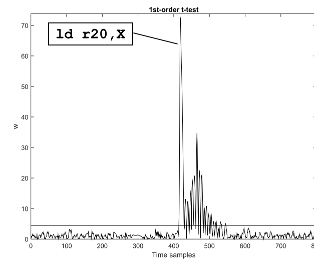
Listing 1.3: Memory remnant experiment.

```
1  ;share x0 in 0x0080
2  ;share x1 in 0x0090
3  ldi r27, 0x00
4  ldi r26, 0x80
5  ld r17, X
6  ldi r17, 0x00
7  ldi r26, 0x85
8  ld r17, X
9  ldi r26, 0x90
10 ld r20, X
```

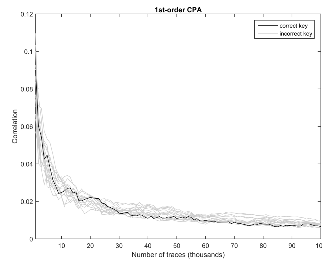
Listing 1.4: Clearing remnant experiment.



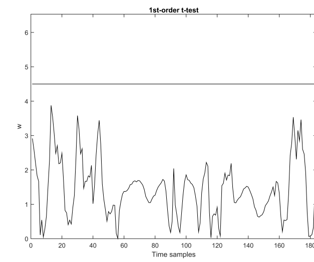
(a) Memory remnant effect,1st-order CPA, HW model, 500 traces.



(b) Memory remnant effect, 1st-order t-test, 5k random vs. 5k fixed.



(c) Clearing remnant effect,1st-order CPA, HW model, 100k traces.



(d) Clearing remnant effect, 1st-order t-test, 100k random vs. 100k fixed.

Fig. 2: Memory-based remnant effect

We address the remnant scenario with two experiments. Listing 1.3 demonstrates how two consecutive SRAM accesses `ld rA, SRAM_x0`, followed by `ld rB, SRAM_x1` produce the remnant effect. Second, in Listing 1.4, we show how clearing the register and accessing an unrelated SRAM address (`0x0085`) can remove the remnant.

As shown in Figures 2a and 2b, consecutive SRAM accesses can potentially lead to ILA violations. Exploiting (in a univariate manner) the memory remnant effect in ATMega163 needs less than 500 traces with our setup. Preventing the effect requires the clearing of the register and the insertion of a dummy SRAM access. Alternatively, the implementor could ensure that same-family shares are not accessed sequentially. Note also that the `st` instruction produces a similar effect. We speculate that the memory remnant effect is caused by the structure of the the memory access mechanism and potentially, the pipelining stages.

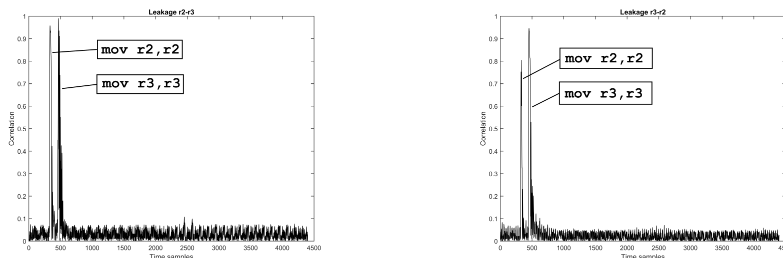
3.3 Neighbour Leakage Effect

The neighbour leakage effect implies that accessing or processing the contents of a data storage unit will cause leakage in another unit as well. For example, assume that share x_0 is stored in register `rB` and share x_1 is being processed in register `rA`. Assume also that the registers `rA, rB` are subject to the neighbour leakage effect. Processing `rB` will produce a value-based leakage of x_0. At the same time, the neighbouring leakage effect will cause `rA` to leak the value of x_1, resulting in transition between shares and the recovery of sensitive value x. The following two experiments (Listing 1.5) verify the neighbour leakage effect between registers `r2, r3`, i.e. a share stored in `r2` leaks when manipulating `r3` and vice-versa.

```
1  ; clear all registers
2  ; sensitive 'x' is in the selected register (r2 OR r3)
3  mov r0, r0
4  nop ; 5 times
5  mov r1, r1
6  nop ; 5 times
7  mov r2, r2
8  nop ; 5 times
9  mov r3, r3
10 nop ; 5 times
11 ...
12 mov r31, r31
```

Listing 1.5: Neighbour leakage experiment for `r2` and `r3`.

As shown above, we use the same code from Listing 1.5, but in the first time we put the sensitive variable x into register `r2` (*only* line 7 should result in leakage). In the second time, we put the sensitive value into the register `r3` (*only* line 9 should leak). However, Figure 3 shows that both register accesses leak. As a result, we have identified a pair of data storage units (`r2,r3`) that exhibit the neighbour leakage effect. Note that in this case the effect is symmetrical, i.e., `r2` triggers `r3` and vice-versa (Figures 3a and 3b). We also observed that the effect is persistent, i.e. the `mov` instructions will trigger the same behavior, even if performed later (not necessarily in order as in Listing 1.5). We run the same experiment in order to identify all possible neighbour leakages in the register file (all pairs in set {`r0,...,r31`}). The results are available in the Appendix, matrix R. The issue mostly affects consecutive registers, although exceptions exist,



(a) Correlation $\rho(HW(x_0), traceset)$, `r2-r3`, 5k traces.

(b) Correlation $\rho(HW(x_0), traceset)$, `r3-r2`, 5k traces.

Fig. 3: Neighbour-based leakage effect

e.g. register `r0`. We did not identify a similar effect in SRAM memory, yet our experiments were limited to a small region of cells. Neighbour-like effects have been observed in consecutive instructions, yet it remains open whether they are cause by proximity or they stem from other effects. We speculate that they relate to the structure of the register file and likely involve the storage and multiplexing mechanism of the registers. Given the pairwise manifestation of the effect, we speculate a pair-based organization of the register file. Still, note that it is hard to link architectural options at the hardware layer directly to side-channel effects. As a solution to the neighbour effect, the developer can opt to avoid storing shares in hazardous registers and keep a safety distance between consecutive instructions. Alternatively, he can store all shares in SRAM, except for the ones currently in use.

Summing up, we stress the following focal points regarding the ILA-breaching effects and their solutions:

- All identified effects are device-dependent, i.e. there is no hard guarantee that they are observable and reproducible in different AVR-based microcontrollers, let alone different architectures such as ARM, TI, PIC etc. Both intra-AVR and inter-architectural observability of the effects remains open.
- The effects are often counter-intuitive when viewed in the assembly layer of abstraction. They originate from the hardware and/or the physical layer, thus can only be detected via experimental evaluation. Linking the assembly ILA-breaching effects to a particular hardware component or physical phenomenon is non-trivial [24, 29], especially without knowledge of the underlying chip architecture and properties.
- Since the effect's detection requires experimental evaluation, different instructions or code arrangements can potentially lead to additional, unidentified ILA-breaching effects. Still, we maintain that it is possible to construct "hardened" masked operations in ATMega163 by removing the identified

effects (see Section 5). It remains open whether the suggested solutions are computationally optimal or more efficient clearing techniques can be identified.

The takeaway message of this section is that assembly-level soundness cannot enforce ILA and hence 1st-order security, due to the nature of the breaching effects. However, it is possible to acquire sufficient knowledge about effects and solutions in a particular device. These non-intuitive checks discussed above can be subsequently integrated into a code-checking tool which can identify such effects in assembly code.

4 Leakage Detection Tool

Several tools that can help designers of cryptographic systems were already suggested and discussed in literature.

SILK[6] presented in 2014 [31] can be used to generate simulated traces based on C++ code. Thus, it allows to generate tracesets during the early stages of development, in order to test an implementation against any attack. However, SILK works only with high level, C++ source code and can not take into account reordering or replacement (even removal) of instructions that is often used by compilers during optimization. Also, this tool does not detect flaws in implementations, it only allows to easily generate simulated traces.

A tool based on formal verification was presented at EUROCRYPT in 2015 [2], capable of detecting design flaws in masking schemes. This tool can analyze programs written using the EasyCrypt framework and its language and it requires the designer to transform the original implementation (e.g. assembly or C code) to EasyCrypt. Unfortunately, errors could potentially be introduced during this process and there is no guarantee that the program written using EasyCrypt will be equivalent to the program in the original programming language. To the best of our knowledge, free automated tools that can transform C or assembly (or other languages that are often used for development of cryptographic software in embedded systems) programs to EasyCrypt do not exist. Moreover, this tool is not opensource and thus can not be used by the developer community.

A simulation-based tool that can be used to analyze masking implementations was presented at FSE in 2016 [25]. It can be used with software and hardware implementations and it requires only high-level implementation source code, such as C language. Due to this fact it can also be blind to re-arrangements of operations (which can lead to side-channel leakage) created by the compiler. Until today, the source code of this tool also remains unavailable[7].

4.1 ASCOLD

In order to assess the security of implementations at the assembly level, we developed a tool called ASCOLD, standing for Assembly Code Leakage Detection

[6] `http://www.ulb.ac.be/di/dpalab/download/SILK_v0.1.zip`

[7] We have contacted the author, there is intention to eventually publish the code.

tool. The tool is written in `python` and the source code is available on our website[8]. ASCOLD uses assembly code as its input in order to run a simulation while checking for potential issues that can cause side-channel information leakage. The tool is compatible only with assembly code (which can be used as is during the development or extracted from the compiled binary file). Thus it is possible to be sure that the executed code will be exactly the same as the code which is analyzed. Otherwise, it becomes impossible to provide any guarantees on the quality of the analysis, i.e. be sure that no additional issues are introduced during compilation.

The simulation run by the tool does not use an instance of an execution i.e., we do not use specific values in order to run the program. ASCOLD starts a program in an initial state and propagates all changes such as combinations of values, their modifications and replacements of one value by another. More precisely, it keeps track of which shares or combinations of shares are stored in each register (or memory cell). During any arithmetic or logical operation, shares stored in different operands are verified, specifically we check whether we combine different shares of the same family without randomizing beforehand. Note that not all combinations are hazardous, yet we opt for such a conservative approach in order to speed-up the verification process.

In the same way, we verify the implementation for the device-specific distance-based leakages for every arithmetic/logical operation, SRAM store or load instruction that is executed. Analytically, we verify whether the previously stored value and the new value cause the overwrite effect 3.1. Similarly, our tool checks the load/store instructions for remnant effects discussed in Section 3.2. In addition, it features the matrix R of *neighbours*, which represents registers that can leak while another register is used (neighbour leakage effect, Section 3.2). In order to bootstrap the whole simulation, the developer needs to provide a configuration file. The configuration file is a simple text file that contains information about the initial state of the system i.e., it describes which registers or addresses in memory contain different secret shares of sensitive values. As the result of the simulation, ASCOLD prints out a line number and the rule that was violated by the program.

ASCOLD works with the AVR family of microcontrollers, it implements the most common memory instructions such as load and store as well as a set of commonly used (in cryptography) instructions such as arithmetic operations (`add`, `mul`, ...) and logical operations (`and`, `eor`, `or`, ...). The same core principles can be applied in order to build a similar tool for a different instruction set or to add new AVR instructions supported by newer microcontrollers.

Limitations The current version of our tool incorporates our findings which are based on the ATMega163, other models of microcontrollers might have slightly different (even additional) issues that cause unintentional information leakage. Among other things, leakage described in Section 3.2 is more likely to be different (affecting different sets of registers) in other models of AVR microcontrollers. ASCOLD does not take into account the effects of pipelining which

[8] `https://github.com/nikita-veshchikov/ascold`

might be an issue in case of a microcontroller which can potentially handle two different shares of the same sensitive value (at different stages of the pipeline) during the same clock cycle. We did not implement all AVR instructions, most importantly the current version of ASCOLD does not support loops. However, we implemented the most commonly used instructions and new instructions/rules can be added due to the tool's extensibility. The lack of jump instructions (loops) can be disregarded via loop-unrolled implementations.

5 Hardened 1st-order Masked Sbox for RECTANGLE

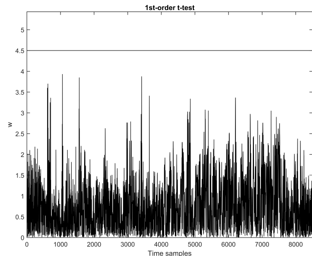
We have discussed the ILA-breaching effects in Section 3 and integrated these observations in the ASCOLD tool, described in Section 4. The current Section builds up on these advances by putting forward a "hardened", 1st-order masked, ISW-based RECTANGLE Sbox. The desired aim is to produce an assembly-based, lightweight Sbox implementation that is secure against 1st-order, univariate attacks, hence forcing the attacker to resort to 2nd-order and/or multivariate techniques.

Our implementation opts for a bitsliced [5, 13] representation, due to both the bitsliced structure of RECTANGLE and to the $GF(2)$-oriented nature of the ISW countermeasure. We employ a bitslicing factor of 2, i.e. we exploit the 8-bit AVR architecture in order to process two 4-bit Sboxes in parallel (nibble-slicing). The Sbox is decomposed into $GF(2)$ operations which can be accelerated by via SIMD-like, 8-bit assembly instructions. The decomposition suggested by Zhang et al. [35] is optimal w.r.t. $GF(2)$ multiplicative complexity, since Grosso et al. [17] established that the minimum number of non-linear operations required by 4x4 Sboxes is 4.

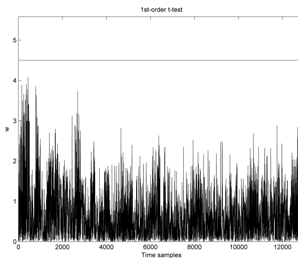
In order to "harden" the Sbox, we use the solutions suggested in Section 3 and follow two approaches: efficient and conservative. In the *efficient* approach, after processing any share, we clear the registers on a need-to basis and insert dummy `ld` instructions to avoid overwrite and remnant effects. We avoid neighbouring leakage effects by always storing the shares in SRAM, i.e. the register file contains only the shares used by the current instruction. In the *conservative* approach, we perform all the afore-mentioned clearing techniques. In addition, we insert dummy `st` instructions and perform thorough register/memory clearing. Both efficient and conservative approaches are applied to every single instruction of the implementation, i.e. the cost is linear w.r.t. the number of instructions that manipulate masked shares. The resulting computational overhead is significant: the efficient "hardened" Sbox implementation runs in 993 clock cycles, i.e. almost 12 times slower compared to the "naive" 1st-order, ISW-based RECTANGLE Sbox, which runs in 87 clock cycles. The conservative "hardened" Sbox implementation requires 1319 clock cycles, i.e. it is 15 times slower. Table 1 contains a comparison between "naive" 1st-order, "naive" 2nd-order and efficient/conservative "hardened" 1st-order bitsliced implementations of the RECTANGLE Sbox in AVR assembly.

Table 1: Masked Sbox comparison in ATMega163

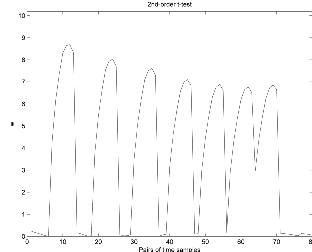
Order d	**Hardened**	**Latency cycles**	**Throughput bits/cycle $\times 10^{-3}$**	**RNG bytes**
Unprotected	no	32	250	0
1st order	no	87	91	4
	yes (eff.)	993	8	4
	yes (cons.)	1319	6	4
2nd order	no	775	10	12



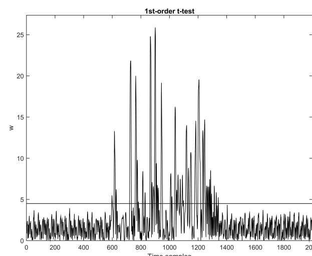
(a) Efficient hardened Sbox, 1st-order t-test, 25k random vs. 25k fixed.



(b) Conservative hardened Sbox, 1st-order t-test, 100k random vs. 100k fixed.



(c) Consevative hardened Sbox, 2nd-order t-test, 25k random vs. 25k fixed.



(d) Naive Sbox, 1st-order t-test, 1k random vs. 1k fixed.

Fig. 4: Hardened and naive Sbox evaluations

Using the random vs. fixed t-test, we evaluate the efficient and conservative "hardened" 1st-order Sboxes, as well as the "naive" 1st-order Sbox. Using a 25k random vs. 25k fixed t-test does not yield any statistically significant leakage in the efficient "hardened" version (Figure 4a). However, we note that a 50k random vs. 50k fixed t-test is able to detect leakage, i.e. trying to reduce the cost of enforcing ILA can have a detrimental effect on security. For the conservative "hardened" Sbox, a 100k random vs. 100k fixed t-test does not detect any leakage (Figure 4b). Note that a *2nd-order* 25k random vs. 25k fixed t-test on a chosen sample window is able to detect leakage. Therefore, we conclude that for the

given device, the informativeness of 1st-order attacks is substantially limited and a 2nd-order attack is the preferable adversarial strategy (Figure 4c). Naturally, the "naive" 1st-order version rejects the null hypothesis (Figure 4d) due to the ILA-breaching effects and the 1st-order leakage can be easily exploited.

So far, the only way to guarantee the actual security order of a real-world implementation was to increase the scheme's theoretical order d, in order to ensure that the implementation attains an actual order of $\lfloor \frac{d}{2} \rfloor$ [1,14]. Clearing the ILA-breaching effects requires a significant overhead and is device-dependent, yet it is the only technique known to us that can enforce 1st-order, univariate security. In addition, hardening does not increase the scheme order d, thus *the random number generation (RNG) cost is not increased*. The previous suggestions require a higher scheme order, hence a significant overhead, since both the implementation cost and the RNG cost are quadratic w.r.t. the order. We compare the "hardened" 1st-order and "naive" 2nd-order implementation costs (in clock cycles) and we observe that hardening the 1st-order Sbox is slower than increasing the scheme's order from 1 to 2 (both in the efficient and in the conservative case). Still, the solution requires no extra RNG and we maintain that removing these effects can also be beneficial to higher-order implementations, i.e. it is complimentary to masking. The extent to which higher-order implementations can benefit from removing such effects remains an open problem.

6 Conclusions

This work investigated the hazards in software masking, suggested a verification tool and established a secure, 1st-order masked Sbox implementation against 1st-order, univariate attacks. Still, several important questions for future work arise. We demonstrated that removing the ILA-breaching effects is feasible, yet identifying the best clearing mechanism and minimizing the overhead is a topic for further exploration. Similarly, the current work is limited to AVR ATMega163 and needs to be extended to different devices and platforms. It could be done by using ASCOLD tool as a base for this kind of work. Moreover, higher-order evaluation techniques are still nascent and in this work we did not focus on 1st-order, yet multivariate attacks such as those that exploit horizontality [4]. In addition, note that the ILA effects are observable throughout an implementation. Not only the cipher-related operations but any manipulation of shares during I/O, RNG routines etc. can create hazards. Thus, there is need for effort towards a fully hardened implementation. Last but not least, we stress that the effects identified depend on the architecture and the physical layer, thus preventing them in the assembly layer is, in principle, less efficient and prone to errors. Future work can strive towards custom-made microcontrollers that enforce ILA in hardware. Ideally, such a microcontroller should be able to guarantee ILA without additional countermeasures such as threshold implementations [22].

References

1. Josep Balasch, Benedikt Gierlichs, Vincent Grosso, Oscar Reparaz, and François-Xavier Standaert. On the cost of lazy engineering for masked software implementations. In Marc Joye and Amir Moradi, editors, *Smart Card Research and Advanced Applications - 13th International Conference, CARDIS 2014, Paris, France, November 5-7, 2014. Revised Selected Papers*, volume 8968 of *Lecture Notes in Computer Science*, pages 64–81. Springer, 2014.
2. Gilles Barthe, Sonia Belaïd, François Dupressoir, Pierre-Alain Fouque, Benjamin Grégoire, and Pierre-Yves Strub. Verified proofs of higher-order masking. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I*, volume 9056 of *Lecture Notes in Computer Science*, pages 457–485. Springer, 2015.
3. Lejla Batina, Benedikt Gierlichs, Emmanuel Prouff, Matthieu Rivain, François-Xavier Standaert, and Nicolas Veyrat-Charvillon. Mutual information analysis: a comprehensive study. *J. Cryptology*, 24(2):269–291, 2011.
4. Alberto Battistello, Jean-Sébastien Coron, Emmanuel Prouff, and Rina Zeitoun. Horizontal side-channel attacks and countermeasures on the ISW masking scheme. In Benedikt Gierlichs and Axel Y. Poschmann, editors, *Cryptographic Hardware and Embedded Systems - CHES 2016 - 18th International Conference, Santa Barbara, CA, USA, August 17-19, 2016, Proceedings*, volume 9813 of *Lecture Notes in Computer Science*, pages 23–39. Springer, 2016.
5. Eli Biham. A fast new DES implementation in software. In Eli Biham, editor, *Fast Software Encryption, 4th International Workshop, FSE '97, Haifa, Israel, January 20-22, 1997, Proceedings*, volume 1267 of *Lecture Notes in Computer Science*, pages 260–272. Springer, 1997.
6. Paul Bottinelli and Joppe W. Bos. Computational aspects of correlation power analysis. *IACR Cryptology ePrint Archive*, 2015:260, 2015.
7. Eric Brier, Christophe Clavier, and Francis Olivier. Correlation power analysis with a leakage model. In Marc Joye and Jean-Jacques Quisquater, editors, *Cryptographic Hardware and Embedded Systems - CHES 2004: 6th International Workshop Cambridge, MA, USA, August 11-13, 2004. Proceedings*, volume 3156 of *Lecture Notes in Computer Science*, pages 16–29. Springer, 2004.
8. D. Canright and Lejla Batina. A very compact "perfectly masked" s-box for AES (corrected). *IACR Cryptology ePrint Archive*, 2009:11, 2009.
9. Suresh Chari, Charanjit S. Jutla, Josyula R. Rao, and Pankaj Rohatgi. Towards sound approaches to counteract power-analysis attacks. In Wiener [34], pages 398–412.
10. Jeremy Cooper, Elke DeMulder, Gilbert Goodwill, Joshua Jaffe, Gary Kenworthy, and Pankaj Rohatgi. Test vector leakage assessment (TVLA) methodology in practice, 2013. `http://icmc-2013.org/wp/wp-content/uploads/2013/09/goodwillkenworthtestvector.pdf`.
11. Jean-Sébastien Coron. Higher order masking of look-up tables. In Phong Q. Nguyen and Elisabeth Oswald, editors, *Advances in Cryptology - EUROCRYPT 2014 - 33rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Copenhagen, Denmark, May 11-15, 2014. Proceedings*, volume 8441 of *Lecture Notes in Computer Science*, pages 441–458. Springer, 2014.
12. Jean-Sébastien Coron, Christophe Giraud, Emmanuel Prouff, Soline Renner, Matthieu Rivain, and Praveen Kumar Vadnala. Conversion of security proofs from

one leakage model to another: A new issue. In Werner Schindler and Sorin A. Huss, editors, *Constructive Side-Channel Analysis and Secure Design - Third International Workshop, COSADE 2012, Darmstadt, Germany, May 3-4, 2012. Proceedings*, volume 7275 of *Lecture Notes in Computer Science*, pages 69–81. Springer, 2012.

13. Joan Daemen, René Govaerts, and Joos Vandewalle. A new approach to block cipher design. In Ross J. Anderson, editor, *Fast Software Encryption, Cambridge Security Workshop, Cambridge, UK, December 9-11, 1993, Proceedings*, volume 809 of *Lecture Notes in Computer Science*, pages 18–32. Springer, 1993.
14. Wouter de Groot, Kostas Papagiannopoulos, Antonio de la Piedra, Erik Schneider, and Lejla Batina. Bitsliced masking and ARM: friends or foes? - fifth international workshop on lightweight cryptography for security and privacy. Lecture Notes in Computer Science, Proccedings Pending, 2016.
15. Louis Goubin and Jacques Patarin. DES and differential power analysis (the "duplication" method). In Çetin Kaya Koç and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems, First International Workshop, CHES'99, Worcester, MA, USA, August 12-13, 1999, Proceedings*, volume 1717 of *Lecture Notes in Computer Science*, pages 158–172. Springer, 1999.
16. Dahmun Goudarzi and Matthieu Rivain. How fast can higher-order masking be in software? Cryptology ePrint Archive, Report 2016/264, 2016. `http://eprint.iacr.org/2016/264`.
17. Vincent Grosso, Gaëtan Leurent, François-Xavier Standaert, and Kerem Varici. LS-designs: Bitslice encryption for efficient masked software implementations. In Carlos Cid and Christian Rechberger, editors, *Fast Software Encryption - 21st International Workshop, FSE 2014, London, UK, March 3-5, 2014. Revised Selected Papers*, volume 8540 of *Lecture Notes in Computer Science*, pages 18–37. Springer, 2014.
18. Yuval Ishai, Amit Sahai, and David Wagner. Private circuits: Securing hardware against probing attacks. In Dan Boneh, editor, *Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings*, volume 2729 of *Lecture Notes in Computer Science*, pages 463–481. Springer, 2003.
19. Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In Wiener [34], pages 388–397.
20. Stefan Mangard and Kai Schramm. Pinpointing the side-channel leakage of masked AES hardware implementations. In Louis Goubin and Mitsuru Matsui, editors, *Cryptographic Hardware and Embedded Systems - CHES 2006, 8th International Workshop, Yokohama, Japan, October 10-13, 2006, Proceedings*, volume 4249 of *Lecture Notes in Computer Science*, pages 76–90. Springer, 2006.
21. Thomas S. Messerges. Securing the AES finalists against power analysis attacks. In Bruce Schneier, editor, *Fast Software Encryption, 7th International Workshop, FSE 2000, New York, NY, USA, April 10-12, 2000, Proceedings*, volume 1978 of *Lecture Notes in Computer Science*, pages 150–164. Springer, 2000.
22. Svetla Nikova, Christian Rechberger, and Vincent Rijmen. Threshold implementations against side-channel attacks and glitches. In Peng Ning, Sihan Qing, and Ninghui Li, editors, *Information and Communications Security, 8th International Conference, ICICS 2006, Raleigh, NC, USA, December 4-7, 2006, Proceedings*, volume 4307 of *Lecture Notes in Computer Science*, pages 529–545. Springer, 2006.
23. Emmanuel Prouff and Matthieu Rivain. Masking against side-channel attacks: A formal security proof. In Thomas Johansson and Phong Q. Nguyen, editors,

Advances in Cryptology - EUROCRYPT 2013, 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings, volume 7881 of *Lecture Notes in Computer Science*, pages 142–159. Springer, 2013.
24. Mathieu Renauld, François-Xavier Standaert, Nicolas Veyrat-Charvillon, Dina Kamel, and Denis Flandre. A formal study of power variability issues and side-channel attacks for nanoscale devices. In Kenneth G. Paterson, editor, *Advances in Cryptology - EUROCRYPT 2011 - 30th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tallinn, Estonia, May 15-19, 2011. Proceedings*, volume 6632 of *Lecture Notes in Computer Science*, pages 109–128. Springer, 2011.
25. Oscar Reparaz. Detecting flawed masking schemes with leakage detection tests. In Thomas Peyrin, editor, *Fast Software Encryption - 23rd International Conference, FSE 2016, Bochum, Germany, March 20-23, 2016, Revised Selected Papers*, volume 9783 of *Lecture Notes in Computer Science*, pages 204–222. Springer, 2016.
26. Matthieu Rivain and Emmanuel Prouff. Provably secure higher-order masking of AES. In Stefan Mangard and François-Xavier Standaert, editors, *Cryptographic Hardware and Embedded Systems, CHES 2010, 12th International Workshop, Santa Barbara, CA, USA, August 17-20, 2010. Proceedings*, volume 6225 of *Lecture Notes in Computer Science*, pages 413–427. Springer, 2010.
27. Tobias Schneider and Amir Moradi. Leakage assessment methodology - A clear roadmap for side-channel evaluations. In Tim Güneysu and Helena Handschuh, editors, *Cryptographic Hardware and Embedded Systems - CHES 2015 - 17th International Workshop, Saint-Malo, France, September 13-16, 2015, Proceedings*, volume 9293 of *Lecture Notes in Computer Science*, pages 495–513. Springer, 2015.
28. François-Xavier Standaert, Tal Malkin, and Moti Yung. A unified framework for the analysis of side-channel key recovery attacks. In Antoine Joux, editor, *Advances in Cryptology - EUROCRYPT 2009, 28th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cologne, Germany, April 26-30, 2009. Proceedings*, volume 5479 of *Lecture Notes in Computer Science*, pages 443–461. Springer, 2009.
29. Marc Stöttinger. *Mutating runtime architectures as a countermeasure against power analysis attacks.* PhD thesis, Darmstadt University of Technology, Germany, 2012.
30. Keccak team. Note on side-channel attacks and their countermeasures. `http://keccak.noekeon.org/NoteSideChannelAttacks.pdf`.
31. Nikita Veshchikov. SILK: high level of abstraction leakage simulator for side channel analysis. In Mila Dalla Preda and Jeffrey Todd McDonald, editors, *Proceedings of the 4th Program Protection and Reverse Engineering Workshop, PPREW@ACSAC 2014, New Orleans, LA, USA, December 9, 2014*, pages 3:1–3:11. ACM, 2014.
32. Junwei Wang, Praveen Kumar Vadnala, Johann Großschädl, and Qiuliang Xu. Higher-order masking in practice: A vector implementation of masked AES for ARM NEON. In Kaisa Nyberg, editor, *Topics in Cryptology - CT-RSA 2015, The Cryptographer's Track at the RSA Conference 2015, San Francisco, CA, USA, April 20-24, 2015. Proceedings*, volume 9048 of *Lecture Notes in Computer Science*, pages 181–198. Springer, 2015.
33. Carolyn Whitnall, Elisabeth Oswald, and Luke Mather. An exploration of the Kolmogorov-Smirnov test as a competitor to mutual information analysis. In Emmanuel Prouff, editor, *Smart Card Research and Advanced Applications - 10th*

IFIP WG 8.8/11.2 International Conference, CARDIS 2011, Leuven, Belgium, September 14-16, 2011, Revised Selected Papers, volume 7079 of *Lecture Notes in Computer Science*, pages 234–251. Springer, 2011.

34. 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*. Springer, 1999.
35. Wentao Zhang, Zhenzhen Bao, Dongdai Lin, Vincent Rijmen, Bohan Yang, and Ingrid Verbauwhede. RECTANGLE: a bit-slice lightweight block cipher suitable for multiple platforms. *SCIENCE CHINA Information Sciences*, 58(12):1–15, 2015.

7 Appendix

Below, we include the 32x32 matrix R that is generated experimentally, while investigating all possible neighbouring leakage effects in the ATMega163 register file (by performing 32 experiments similar to Listing 1.5). Value '1' denotes the presence of leakage and '0' the absence. The tool ASCOLD uses R in order to detect neighbour-based ILA violations.

	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
00	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
01	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
02	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
03	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
04	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
05	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
06	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
07	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
08	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
09	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0
19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0
20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0
21	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0
22	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0
23	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0
24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0
25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0
26	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0
27	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0
28	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0
29	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0
30	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
31	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1