Physical Security Versus Masking Schemes

Jean-Luc Danger, Sylvain Guilley, Annelie Heuser, Axel Legay, and Tang Ming

Abstract Numerous masking schemes have been designed as provable countermeasures against side-channel attacks. However, currently, several side-channel attack models coexist, such as "probing" and "bounded moment" models, at bit or word levels. From a defensive standpoint, it is thus unclear which protection strategy is the most relevant to adopt.

In this survey article, we review adversarial hypotheses and challenge masking schemes with respect to practical attacks. In a view to explain in a pedagogical way how to secure implementations, we highlight the key aspects to be considered when implementing a masking scheme.

1 Context About the Protection Problem

Sensitive computations must be secured against non-invasive attacks, which attempt to correlate the leakage of some operations with a hypothetical model [13].

A protection against this threat is the *masking* [13, Chap. 9] countermeasure. Masking consists in changing the intermediate variables of the computation into

J.-L. Danger (✉)
Télécom ParisTech, Paris, France

Secure-IC S.A.S., Cesson-Sévigné, France
e-mail: jean-luc.danger@telecom-paristech.fr

S. Guilley
Télécom ParisTech, Paris, France

Secure-IC, Paris, France

École normale supérieure, Paris, France

A. Heuser
CNRS, IRISA, Rennes, France

A. Legay
INRIA, IRISA, Rennes, France

T. Ming
Wuhan University, Wuhan, China

Ç. K. Koç (ed.), *Cyber-Physical Systems Security*,
https://doi.org/10.1007/978-3-319-98935-8_13

randomized versions, which are thus decorrelated from the unprotected variables, each being a potential target for a side-channel attack. Indeed, leakage localization tests have been put forward to pinpoint leakages [4, 10, 14]; hence masking schemes must have a full coverage.

In particular, in the field of symmetrical encryption, the mainstream approach consists in the use of purely Boolean structures. This is the case of widely used (and standardized) ciphers, such as DES [15] and AES [16]. In both these examples, the operations (except for simple data move, which will simply amplify the leakage signal-to-noise ratio, but not create new leakage model) simply consist in XORs and in look-up tables (LUTs). It is therefore natural to restrict to so-called Boolean masking, where the only operation in terms of masking is the XOR. This is innately compatible with the functional XOR operations, and LUTs are also easy to protect, e.g., using recomputation [20].

1.1 Nature of Computation

Complex computations, like cryptographic algorithms, can be seen as a sequence of parallel basic operations. In hardware, the basic operations are *logic gates*. They take as input a small amount of bits (e.g., 1, 2, 3, up to maximum 6 usually) and yield another bit according to a Boolean function. In software, the basic operations are *instructions*. They take a couple of operands and yield another one, computed through a deterministic function. For instance, `xor r1 r2 r3` computes the exclusive-or of 32-bit registers `r2` and `r3` and saves the result `r2 xor r3` in `r1`.

1.2 Combinational or Sequential?

We notice the spatio-temporal nature of computation: many bits are manipulated *in parallel*, and sometimes, the computation has loops. In hardware, this is called an *iterative implementation*, for instance, the instantiation of gates for one round of AES-128, which are evaluated *ten* times. In software, interestingly, the loops (in the underlying hardware, i.e., the processor) cannot be unrolled, because all operations pass through a unique *integer unit*. Typically, the accumulated register is updated again and again at each instruction (at least, for most instructions—with the exception of instructions where the result is saved directly in memory).[1]

[1]This highlights a very paradoxical modelization of software, even when is it straight line. A straight-line code is seen as sequential in software where it indeed consists in looping of the hardware accumulator register into itself when operations are chained in series. Obviously, the looping of the accumulator into itself only holds for basic controllers. Performance-oriented processors may behave in a more complex way—typically, the pipeline in a processor can break those loops.

1.3 Outline of the Article

The rest of this article is structured as follows: First of all, the mainstream masking algorithms are presented and challenged from a security standpoint in Sect. 2. Second, masking is analyzed vis-à-vis technological and logical high-order leakage function in Sect. 3. A new definition of realistic security objectives is given in Sect. 4. Eventually, this chapter is concluded in Sect. 5.

2 Definition of t-Order Security by ISW [12]

2.1 Revisiting of ISW Definition

Ishai, Sahai, and Wagner (ISW) define as *stateless circuits* [12, Sec. 2] fully combinational circuits, which are acyclic. This models fully unrolled hardware implementations, which are usually prohibitive in cost, but all the same implemented in some contexts like for extremely low latency or for some sort of side-channel resistance [3]. On the other hand, *stateful circuits* are circuits with loops.

Definition 1 (t-Order Security, as per ISW) According to ISW, a circuit is t-order secure if the attacker can get no information about the unprotected variables by:

1. Using t probes at arbitrary positions in one loop[2]
2. But with the possibility to re-probe (and even to move the t probes) for free at every loop[3]

2.2 Ill-Formed Definition

The problem with this definition of probing security is that it does not characterize well some countermeasures. For instance, *perfect masking* [5] of order t can be either secure or insecure depending on the implementation. The secure

[2] We quote [12, p. 464]: "*a* t-limited adversary is one that can observe at most t wires of the circuit within a certain time period (such as during one clock cycle)."

[3] We quote footnote 6 page 464 of [12]: "By default, we allow the adversary to adaptively move its t probes between time periods, but not within a time period." See also the complement given in [12, pp. 466–467]; we quote next: "Prior to each invocation, the adversary may fix an arbitrary set of t internal wires to which it will gain access in that invocation. We stress that while this choice may be adaptive between invocations, i.e., may depend on the outputs and on wire values observed in previous invocations, the adversary is assumed to be too slow to move its probes while the values propagate through the circuit."

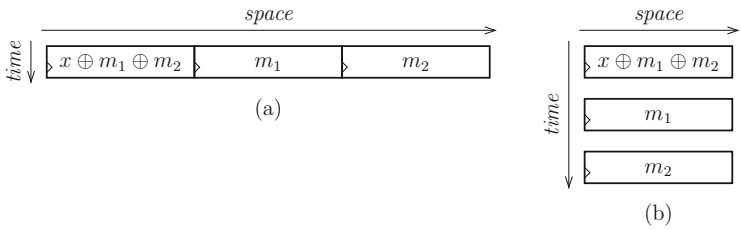

Fig. 1 Parallel (**a**) vs. sequential (**b**) implementation of *perfect masking* for $t = 2$

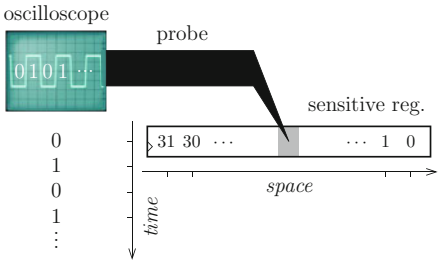

Fig. 2 Linear readout of one bit in a word of $n = 32$ bits, using one prober tip

implementation is the parallel one (see Fig. 1a), because indeed t probes are needed, whereas the insecure implementation is the sequential one (see Fig. 1b), because one probe suffices to read out one bit of the t shares over t clock cycles (even without changing the probe location). This kind of *linear readout* attack scenario is illustrated in Fig. 2; once the probe tip is installed on top on the register bit to probe, the consecutive values held in this DFF (Data Flip-Flop) are read out non-invasively, one after the other. This definition shall not be confused with the more general *(word) probing model* where whole words can be read out at once.

Admittedly (this was the hypothesis in seminal paper of ISW [12]), the relevant security parameter in probing is the number of probes. Indeed:

1. The probe tips are very small, but the probe itself is large (see Fig. 3a); hence only few of them can be placed over a circuit.
2. The step consisting in placing the probe is costly, for at least two reasons. First of all, the identification of the probe's location is time-consuming (it consists, as to say, to identify a needle in a haystack). Second, the positioning of the probe (see Fig. 3a) is slightly invasive, in that it requires to scratch the chip surface to get a reliable electrical contact with the resource to spy (see Fig. 3b).

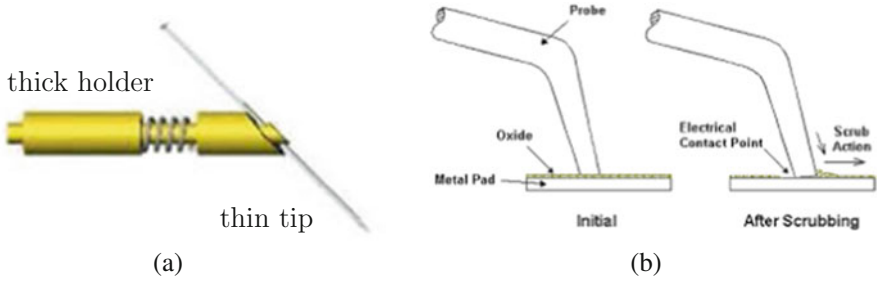

Fig. 3 Probe example, courtesy of http://www.bridgetec.com/holders.html (**a**) and operation for the probe tip to properly contact the targeted area (**b**)

2.3 Attack on Coron's Higher-Order Masking of Look-Up Tables [8]

Coron's higher-order masking of look-up tables [8] is a *software* variant of ISW scheme [12] (more precisely, it is a variant of the word-level variant of ISW, namely [22]). This scheme is proven high-order secure, but the proof is incorrect, because the given implementation and the one assumed in the proof do not match: in the given implementation (algorithms), some resources are reused over time, hence creating a security weakness, as we shall detail in this section.

We recall Coron's masked computation of look-up table $S : \mathbb{F}_2^n \to \mathbb{F}_2^n$ in Algorithm 1, which makes use of masks sharing refresh procedure given in Algorithm 2. In this latter algorithm, the operator "$\leftarrow_{\mathscr{R}}$" stands for uniformly randomized affectation.

We show that there is a second-order attack (in the sense of ISW) on Coron's scheme:

- The attacker probes at line 2 of Algorithm 2; then [one bit of] all the random numbers injected in Algorithm 1 are known. Let us call them $r_{i,j}$ for $0 \leq i \leq t$ (the ith time the RefreshMasks function is called) and for $1 \leq j \leq t$ (the jth fresh mask in invocation i of RefreshMasks).
- In parallel, the attacker also probes [one bit of] y_0 at line 12 of Algorithm 1. This value is equal to $S(\bigoplus_{i=0}^{t} x_i) \oplus \bigoplus_{i'=0}^{i} \bigoplus_{j=1}^{t} r_{i',j}$. As the attacker knows [one bit of] all the $r_{i,j}$, he can deduce [one bit of] $S(\bigoplus_{i=0}^{t} x_i) = S(x)$.

With one probe, only one bit of the $n = 8$ bit register can be probed, which allows nonetheless to recover one bit of $S(x)$ in the clear. However, this is sufficient information to break the AES: after knowing about n values of $S(x)$ targeted bit for $x = p \oplus k$ (plaintext $p \in \mathbb{F}_2^n$ xored with the key byte $k \in \mathbb{F}_2^n$) knowing the values of p, a unique k can be derived.

Input : $x_0, \ldots, x_t$ such that $x = x_0 \oplus \ldots \oplus x_t$
Output : $y_0, \ldots, y_t$ such that $y = S(x) = y_0 \oplus \ldots \oplus y_t$

1 **for** $u \in \mathbb{F}_2^n$ **do**
2 | $T(u) \leftarrow (S(u), 0, \ldots, 0)$ ▷ $\bigoplus_{j=0}^{t} T(u)[j] = S(u)$
3 **end**
4 **for** $i = 0$ to $t-1$ **do**
5 | **for** $u \in \mathbb{F}_2^n$ **do**
6 | | **for** $j = 0$ to t **do** $T'(u)[j] \leftarrow T(u \oplus x_i)[j]$
7 | **end**
| ▷ $\bigoplus_{j=0}^{t} T'(u)[j] = S(u \oplus x_0 \oplus \ldots \oplus x_i)$
8 | **for** $u \in \mathbb{F}_2^n$ **do**
9 | | $T(u) \leftarrow$ RefreshMasks$(T'(u))$
10 | **end**
| ▷ $\bigoplus_{j=0}^{t} T(u)[j] = S(u \oplus x_0 \oplus \ldots \oplus x_i)$
11 **end**
12 $(y_0, \ldots, y_t) \leftarrow$ RefreshMasks$(T(x_t))$ ▷ $\bigoplus_{j=0}^{t} T(x_n)[j] = S(x)$
13 **return** $(y_0, \ldots, y_t)$

Algorithm 1: Masked computation of $y = S(x)$ (Alg. 1 in [8])

Input : $z_0, \ldots, z_t$ such that $z = z_0 \oplus \ldots \oplus z_t$
Output : $z_0, \ldots, z_t$ such that $z = z_0 \oplus \ldots \oplus z_t$

1 **for** $j = 1$ to t **do**
2 | $tmp \leftarrow_{\mathscr{R}} \mathbb{F}_2^n$
3 | $z_0 \leftarrow z_0 \oplus tmp$
4 | $z_j \leftarrow z_j \oplus tmp$
5 **end**
6 **return** $(z_1, \ldots, z_t)$

Algorithm 2: The RefreshMasks function (Alg. 2 in [8])

The values to probe can be found as well in the reference code of https://github.com/coron/htable/blob/master/src/aes_htable.c (hash `a9e88df`, put online on 25 Sep 2015):

- Line 39, in function `subbyte_table` (line 12 of Algorithm 1)
- Line 51, in function `refreshword` (line 2 of Algorithm 2)

The online version of file `aes_htable.c` shall thus not be used as is. Its security problem can be fixed easily, by avoiding the reuse $t-1$ times of tables T and T' and $(t-1) \times t$ times of variable tmp. The corrected algorithm is Algorithm 3 (which calls Algorithms 4 and 5 as subfunctions).

Input : $x_0, \ldots, x_t$ such that $x = x_0 \oplus \ldots \oplus x_t$
Output : $y_0, \ldots, y_t$ such that $y = S(x) = y_0 \oplus \ldots \oplus y_t$

Initialize table $r_{i,j}$ $(0 \leq i \leq t, 1 \leq j \leq t)$ with InitRefreshMasks ▷ using Algorithm 5
for $u \in \mathbb{F}_2^n$ **do**
  $T(u) \leftarrow (S(u), 0, \ldots, 0)$ ▷ $\bigoplus_{j=0}^{t} T(u)[j] = S(u)$
end
for $i = 0$ to $t - 1$ **do**
  for $u \in \mathbb{F}_2^n$ **do**
    for $j = 0$ to t **do** $T'(u + i \times 2^n)[j] \leftarrow T((u \oplus x_i) + i \times 2^n)[j]$
  end
  ▷ $\bigoplus_{j=0}^{t} T'(u + i \times 2^n)[j] = S(u \oplus x_0 \oplus \ldots \oplus x_i)$
  for $u \in \mathbb{F}_2^n$ **do**
    $T(u + (i + 1) \times 2^n) \leftarrow$ RefreshMasks$(i, T'(u + i \times 2^n))$ ▷ using Algorithm 4
  end
  ▷ $\bigoplus_{j=0}^{t} T(u + (i + 1) \times 2^n)[j] = S(u \oplus x_0 \oplus \ldots \oplus x_i)$
end
$(y_0, \ldots, y_t) \leftarrow$ RefreshMasks$(t, T(x_t + t \times 2^n))$ ▷ using Algorithm 4
▷ $\bigoplus_{j=0}^{t} T(x_n + t \times 2^n)[j] = S(x)$
return $(y_0, \ldots, y_t)$

Algorithm 3: Masked computation of $y = S(x)$ (fixed version of Algorithm 1)

Input : Index i, $0 \leq i \leq t$, and $z_0, \ldots, z_t$ such that $z = z_0 \oplus \ldots \oplus z_t$
Output : $z_0, \ldots, z_t$ such that $z = z_0 \oplus \ldots \oplus z_t$

for $j = 1$ to t **do**
  $z_0 \leftarrow z_0 \oplus r_{i,j}$
  $z_j \leftarrow z_j \oplus r_{i,j}$
end
return $(z_1, \ldots, z_t)$

Algorithm 4: The RefreshMasks function (fixed version of Algorithm 2)

Input : None
Output : A table of $t \times (t - 1)$ random numbers

for $i = 0$ to t **do**
  for $j = 1$ to t **do**
    $r_{i,j} \leftarrow_{\mathscr{R}} \mathbb{F}_2^n$
  end
end
return $r_{i,j}$

Algorithm 5: The generation of internal masks InitRefreshMasks

2.4 Motivation for Bit-Mixing Masking Schemes

The attack in the previous Sect. 2.3 has revealed structural weaknesses in state-of-the-art masking schemes. In particular, the attack exploiting the consecutive values taken by one bit allows to break arbitrary high-order masking schemes such as perfect masking scheme [5] or Coron's table-based masking [8].

In reaction to this weakness, so-called *inner product* masking schemes have been proposed [1, 2, 18, 23], which make such attack more chancy. A comprehensive analysis between probing security at bit *versus* word levels is carried out in [7, 19].

3 Analysis of the Security Issue

In the previous section, we showed how one single probe is able to defeat *at bit level* high-order masking schemes proved *at word level*. Therefore, we recommended in Sect. 2.4 masking schemes which combine, by design, several bits together. In this section, we examine how *multi-bit* high-order leakage might arise, created either by the hardware or the software themselves (to the free benefit of the attacker).

3.1 Hardware Case

In the hardware case, coupling between bits can be due:

- *Spatially*, to:
  - *Glitches*: in combinational logic, gates do not evaluate in their order in the netlist (since they are non-synchronizing); for more information on how glitches appear in combinational circuits and contribute to lower the security with respect to side-channel attacks, we refer the reader to the didactic explanations provided in section 4 of [11] devoted to this topic.
  - *IR drop*: individual gates cannot be considered independent, since they share the same power/ground network; the effect is well illustrated in Fig. 9(b) of [9, Sec. 4.2.3].
  - *Capacitive coupling*: some gates, physically placed close one to each other, can have a capacitive coupling of their output nets; the effect is well illustrated in Fig. 9(c) of [9, Sec. 4.2.3].
  - *Unselected gates*: some gates are instantiated in a netlist and supposed to be have a useful functionality only at some times. But actually, being there (i.e., being instantiated and thus activated), they contribute to the leakage continuously, even when they handle data which is eventually not selected (i.e., not used downstream). For example, Fig. 4a illustrates a complete masking scheme, made up of $four$ algorithms: (1) data masking, (2) operation

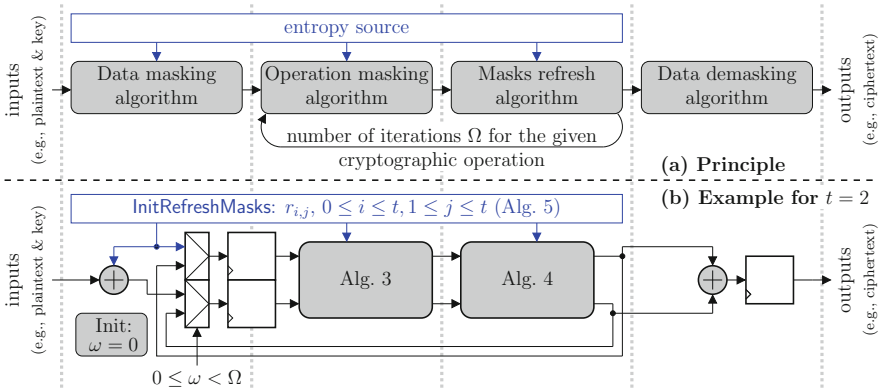

Fig. 4 Masking scheme steps (**a**) and illustration of a netlist for perfect masking with $t = 2$ shares (**b**), with an *unselected gate's* leakage flaw

masking, (3) masks refresh (*optional*), and (4) data unmasking. The last algorithm should, obviously, be executed only at the end of the computation. In the example of a masked iterative block cipher with Ω rounds, the shares can be combined only to recover the ciphertext. However, Fig. 4b shows a faulty implementation of a perfect masking scheme, wherein the data unmasking logic is executed at each round $0 \leq \omega \leq \Omega$ of the block cipher, thereby leaking information on all intermediate rounds.[4]

- *Temporally*, when some gates are reused over time, as explained in the *linear probing* issue (recall Fig. 2 of Sect. 2.2).

3.2 Software Case

In this section, we tackle the question of software security with respect to side-channel leakage. Let us first precise what is implied under the term "software." Software means that some control is written in a memory, but the execution is carried out by one (or several) processor(s). Now, processors are pieces of hardware and hence suffer from the same leakage sources as mentioned in previous Sect. 3.1.

Let us recall two optimizations occurring at *compilation* stage, which make software execution more amenable to side-channel attacks [17]:

[4]Notice that Fig. 4b purposely represents an incorrect masking scheme for an iterative block cipher (for the sake of counterexample) and shall not be implemented this way. Rather, in a secure version, the XOR demasking gate shall be enabled only for the last round (i.e., when round counter ω is equal to its maximal value Ω).

- Register packing consists in regrouping several variables into one register. Indeed, registers are usually wide and hence can accommodate the concatenation of several *words* (say *shares*), to be processed in parallel, e.g., using bitslice operations.
- In Static Single Assignment (SSA) mode, any new variable is affected to a new virtual register. However, in the next pass, registers are allocated. Dead registers are considered as fresh resources and hence are reused, which opens the door to *linear probing* issues.

For the sake of pedagogy, let us make explicit some unusual sources of leakage occurring in software. One important point to make clear is that **glitches do exist in software**. In particular, we find in CPUs the case of leakage of *unselected gates*, owing to unselected logic mentioned in the previous section. Let us illustrate this on the example of two CPUs: 1. 6502, 2. LEON3. For the sake of legibility, lines which are too long (ending by a "\" sign) have been folded.

6502

Let us analyze the integer unit of 6502 processor, described in VHDL in https://github.com/chenxiao07/vhdl-nes/tree/master. Lines 765–772 of source file `vhdl-nes-master/src/free6502.vhd` are recalled below:

```
-- The ALU adder
alu_add <= alu_in1 + alu_add_in2 + alu_add_cin;

-- The ALU itself    This is purely combinatorial logic
process (alu_add, alu_in1, alu_in2, alu_op, c_flag)
begin
  -- default for alu_add inputs
  alu_add_in2 <= alu_in2;
  alu_add_cin <= c_flag;

  case alu_op is
    -- [...]
    when MC_BIT_AND => alu_out <= alu_in1 and alu_in2;
    when MC_BIT_OR =>  alu_out <= alu_in1 or  alu_in2;
    when MC_BIT_XOR => alu_out <= alu_in1 xor alu_in2;
    when MC_BIT_ASL => alu_out <= alu_in1(7 downto 0) & "0";
    when MC_BIT_LSR => alu_out <= alu_in1(0) & \
                                  "0" & alu_in1(7 downto 1);
    when MC_BIT_ROL => alu_out <= alu_in1(7 downto 0) & c_flag;
    when MC_BIT_ROR => alu_out <= alu_in1(0) \
                                  & c_flag & alu_in1(7 downto 1);
    when others =>     alu_out <= alu_in1;
  end case;
end process;
```

LEON3

Let us also analyze the integer unit of LEON3 processor, described in VHDL in iu3.vhd excerpt below:

```
procedure logic_op(r : registers; aluin1, aluin2, mey : word;
      ymsb : std_ulogic; logicres, y : out word) is
variable logicout : word;
begin
  case r.e.aluop is
  when EXE_AND   => logicout := aluin1 and aluin2;
  when EXE_ANDN  => logicout := aluin1 and not aluin2;
  when EXE_OR    => logicout := aluin1 or aluin2;
  when EXE_ORN   => logicout := aluin1 or not aluin2;
  when EXE_XOR   => logicout := aluin1 xor aluin2;
  when EXE_XNOR  => logicout := aluin1 xor not aluin2;
  when EXE_DIV   =>
    if DIVEN then logicout := aluin2;
    else logicout := (others => '-'); end if;
  when others => logicout := (others => '-');
  end case;
  if (r.e.ctrl.wy and r.e.mulstep) = '1' then
    y := ymsb & r.m.y(31 downto 1);
  elsif r.e.ctrl.wy = '1' then y := logicout;
  elsif r.m.ctrl.wy = '1' then y := mey;
  elsif MACPIPE and (r.x.mac = '1') then \
    y := mulo.result(63 downto 32);
  elsif r.x.ctrl.wy = '1' then y := r.x.y;
  else y := r.w.s.y; end if;
  logicres := logicout;
end;
```

Analysis of the 6502 and LEON3 Codes

It clearly appears that both CPUs (6502 and LEON3) do compute all the possible bitwise operations *in parallel* before selecting the one actually relevant for the computation indicated by the current instruction. This behavior is explained in Fig. 5 and corresponds to an *unselected gate's* flaw. In particular, the arithmetic addition combines all the bits (since there is a carry propagation, i.e., the last bit depends on all previous bits) and hence defeats the *register packing* strategy. This is illustrated in Fig. 6. The structure of the full adder (FA) is recalled in Fig. 7.

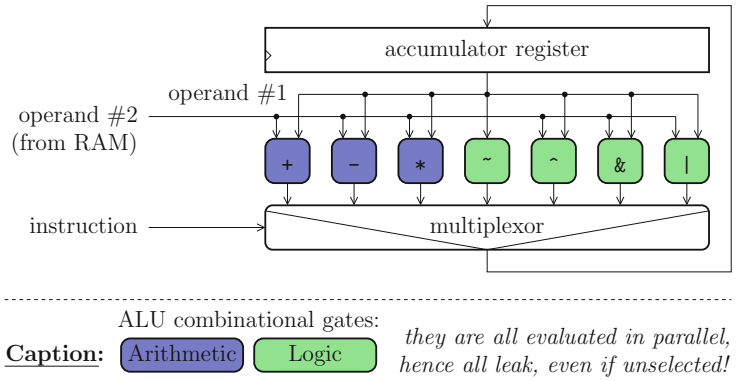

Fig. 5 Illustration of the *unselected gate's* flaw on the ALU (Arithmetic and Logic Unit) of a CPU

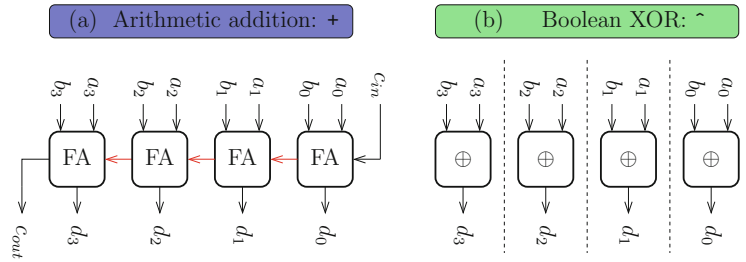

Fig. 6 Illustration addition (+) and exclusive-or (^) operations: (**a**) the addition ($a + b + c_{\text{in}} = 2^n c_{\text{out}} + d$) couples bits of the accumulator (here assumed $n = 4$ bit), through carries represented as red arrows, (**b**) the exclusive-or operation ($a \oplus b = d$) is bitslice, meaning that each bit is computed independently (as illustrated by the "⋮" separation line)

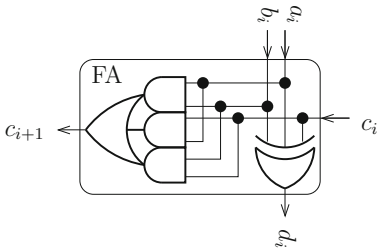

Fig. 7 Full adder, as used in the addition of Fig. 6a

4 New Definition of Security Order

Therefore, in the probing model of ISW, the defender has only one option to enhance the security: disallow the adversary from probing more than once with one probe by

- In hardware: unrolling circuits, hence designing fully combinational logic (as in [3, 24])
- In software: unrolling loops, hence using n^2 times more memory than announced in [8]

For this reason, we propose a new definition of security order. We call this definition the security order in the Noisy Non-Injective (NNI) model.

Definition 2 (t-Order Security, in the NNI Model) The implementation is t-order secure in the NNI model if no information can be recovered by measuring

- t_{space} different bits, at
- t_{time} different times,

where $t_{\text{space}} \times t_{\text{time}} \leq t$.

In this definition,

- t_{space} relates to the "high-order" aspect of side-channel attacks in the NNI model.
- t_{time} relates to the "multivariate" aspect of side-channel attacks.

We introduce the following result to motive for the definition:

Proposition 1 *Let $\mathscr{L}$ be a pseudo-Boolean function $\mathbb{F}_2^n \to \mathbb{R}$, of degree one. Then, assuming the attacker performs a zero-offset attack (i.e., $t_{\text{time}} = 1$), we have that*

$$\forall i, 0 \leq i \leq t, \quad \mathbb{E}(\mathscr{L}(Z)^i | X = x) \quad \text{does not depend on } x$$

if the implementation is t-order secure.

Proof See, for instance, Theorem 2 and/or Proposition 3 in [6].

This means that the attacker will need to raise the leakage traces to the power t_{space}; hence a noise variance raised to the power t_{space}. Assuming that the noise is independent from sample to sample, then we also have that the noise variance is raised to the power t_{time} in t-variate attacks [21].

Thus, the relevant quantity is indeed the product between $t_{\text{space}} \times t_{\text{time}}$.

This model is more satisfactory, as it allows for the designer to do:

- In hardware, fully combinational circuits ($t_{\text{time}} = 1$)
- In software, fully sequential bitslice implementations ($t_{\text{space}} = 1$))

The security notion of Definition 2 is thus more flexible in terms of design solutions to thwart attacks and also more realistic than Definition 1.

However, this model is relevant only in the case where the noise is minimal, so that taking two consecutive measurements decreases the effectiveness of the attack.

5 Conclusion

In this article, we reviewed some masking schemes of the scientific literature. We present their effectiveness with respect to real-world analysis methods and suggest some adaptations. The goal of this article is mostly to underline the discrepancy which can exist between attacks and designs. As of today, attacks arise from the academic world and are pretty virulent. The protection of sensitive circuits is evolving less fast, but a key for a good protection is to understand the risk. The analysis of several adversarial models is performed, and the attacks are confronted to real implementations. We clearly identify a gap between attacks and countermeasures and contribute to bridge it.

Acknowledgements This work was supported in part by the National Natural Science Foundation of China under Grant (61472292, 61332019) and by the key technology research of new-generation high-speed and high-level security chip for smart grid (526816160015).

References

1. J. Balasch, S. Faust, B. Gierlichs, Inner product masking revisited, in *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*, ed. by E. Oswald, M. Fischlin. Lecture Notes in Computer Science, vol. 9056 (Springer, Berlin, 2015), pp. 486–510
2. J. Balasch, S. Faust, B. Gierlichs, C. Paglialonga, F.-X. Standaert, Consolidating inner product masking, in *Advances in Cryptology - ASIACRYPT 2017 - 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3–7, 2017, Proceedings, Part I*, ed. by T. Takagi, T. Peyrin. Lecture Notes in Computer Science, vol. 10624 (Springer, Berlin, 2017), pp. 724–754
3. S. Bhasin, S. Guilley, L. Sauvage, J.-L. Danger, Unrolling cryptographic circuits: a simple countermeasure against side-channel attacks, in *Topics in Cryptology - CT-RSA 2010, The Cryptographers' Track at the RSA Conference 2010, San Francisco, CA, USA, March 1–5, 2010. Proceedings*, ed. by J. Pieprzyk. Lecture Notes in Computer Science, vol. 5985 (Springer, Berlin, 2010), pp. 195–207
4. S. Bhasin, J.-L. Danger, S. Guilley, Z. Najm, Side-channel leakage and trace compression using normalized inter-class variance, in *Proceedings of the Third Workshop on Hardware and Architectural Support for Security and Privacy, HASP '14* (ACM, New York, 2014), pp. 7:1–7:9
5. J. Blömer, J. Guajardo, V. Krummel, Provably secure masking of AES, in *Selected Areas in Cryptography*, ed. by H. Handschuh, M.A. Hasan. Lecture Notes in Computer Science, vol. 3357 (Springer, Berlin, 2004), pp. 69–83
6. J. Bringer, C. Carlet, H. Chabanne, S. Guilley, H. Maghrebi, Orthogonal direct sum masking: a smartcard friendly computation paradigm in a code, with builtin protec-

tion against side-channel and fault attacks. Cryptology ePrint Archive, Report 2014/665, 2014. http://eprint.iacr.org/2014/665/ (extended version of conference paper (J. Bringer, C. Carlet, H. Chabanne, S. Guilley, H. Maghrebi, Orthogonal direct sum masking – a smartcard friendly computation paradigm in a code, with builtin protection against side-channel and fault attacks, in *WISTP International Conference on Information Security Theory and Practice*. Lecture Notes in Computer Science, vol. 8501 (Springer, Berlin, 2014), pp. 40–56. Heraklion, Greece))
7. C. Carlet, S. Guilley, Statistical properties of side-channel and fault injection attacks using coding theory. Cryptogr. Commun. **10**(5), 909–933 (2018). https://doi.org/10.1007/s12095-017-0271-4
8. J.-S. Coron, Higher order masking of look-up tables, in *Annual International Conference on the Theory and Applications of Cryptographic Techniques, EUROCRYPT*, ed. by P.Q. Nguyen, E. Oswald. Lecture Notes in Computer Science, vol. 8441 (Springer, Berlin, 2014), pp. 441–458
9. J.-L. Danger, S. Guilley, P. Nguyen, R. Nguyen, Y. Souissi, Analyzing security breaches of countermeasures throughout the refinement process in hardware design flow, in *Design, Automation & Test in Europe Conference & Exhibition, DATE 2017, Lausanne, Switzerland, March 27–31, 2017*, ed. by D. Atienza, G. Di Natale (IEEE, Piscataway, 2017), pp. 1129–1134
10. R.J. Easter, J.-P. Quemard, J. Kondo, Text for ISO/IEC 1st CD 17825 – Information technology – Security techniques – Non-invasive attack mitigation test metrics for cryptographic modules, March 22 2014. Prepared within ISO/IEC JTC 1/SC 27/WG 3. http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=60612
11. S. Guilley, A. Heuser, O. Rioul, Codes for side-channel attacks and protections, in *Codes, Cryptology and Information Security - Second International Conference, C2SI 2017, Rabat, Morocco, April 10–12, 2017, Proceedings - In Honor of Claude Carlet*, ed. by S. El Hajji, A. Nitaj, E.M. Souidi. Lecture Notes in Computer Science, vol. 10194 (Springer, Berlin, 2017), pp. 35–55
12. Y. Ishai, A. Sahai, D. Wagner, Private circuits: securing hardware against probing attacks, in *Annual International Cryptology Conference, CRYPTO*. Lecture Notes in Computer Science, vol. 2729 (Springer, Berlin, 2003), pp. 463–481. Santa Barbara, California
13. S. Mangard, E. Oswald, T. Popp, *Power Analysis Attacks: Revealing the Secrets of Smart Cards* (Springer, Berlin, 2006). http://www.springer.com/. ISBN 0-387-30857-1, http://www.dpabook.org/
14. A. Moradi, S. Guilley, A. Heuser, Detecting hidden leakages, in *Applied Cryptography and Network Security*, ed. by I. Boureanu, P. Owesarski, S. Vaudenay, vol. 8479 (Springer, Berlin, 2014). 12th International Conference on Applied Cryptography and Network Security, Lausanne, Switzerland
15. NIST/ITL/CSD, Data Encryption Standard. FIPS PUB 46-3, Oct 1999. http://csrc.nist.gov/publications/fips/fips46-3/fips46-3.pdf
16. NIST/ITL/CSD, Advanced Encryption Standard (AES). FIPS PUB 197, Nov 2001. http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf (also ISO/IEC 18033-3:2010)
17. K. Papagiannopoulos, N. Veshchikov, Mind the gap: towards secure 1st-order masking in software, in *Constructive Side-Channel Analysis and Secure Design: 8th International Workshop, Paris, France, COSADE* (Springer, Berlin, 2017)
18. R. Poussier, Q. Guo, F.-X. Standaert, C. Carlet, S. Guilley, Connecting and improving direct sum masking and inner product masking, in *Smart Card Research and Advanced Applications - 16th International Conference, CARDIS 2017, Lugano, Switzerland, November 13–15, 2017, Revised Selected Papers*, ed. by Y. Teglia, T. Eisenbarth. Lecture Notes in Computer Science (Springer, Berlin, 2017)
19. R. Poussier, Q. Guo, F.-X. Standaert, C. Carlet, S. Guilley, Connecting and improving direct sum masking and inner product masking, in *Smart Card Research and Advanced Applications - 16th International Conference, CARDIS 2017, Lugano, Switzerland, November 13–15, 2017, Revised Selected Papers*, ed. by T. Eisenbarth, Y. Teglia. Lecture Notes in Computer Science, vol. 10728 (Springer, Berlin, 2017), pp. 123–141

20. E. Prouff, M. Rivain, A generic method for secure SBox implementation, in *International Workshop on Information Security Applications WISA*, ed. by Sehun Kim, Moti Yung, and Hyung-Woo Lee. Lecture Notes in Computer Science, vol. 4867 (Springer, Berlin, 2007), pp. 227–244
21. E. Prouff, M. Rivain, Masking against side-channel attacks: a formal security proof, in *Advances in Cryptology - EUROCRYPT 2013, 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26–30, 2013. Proceedings*, ed. by T. Johansson, P.Q. Nguyen. Lecture Notes in Computer Science, vol. 7881 (Springer, Berlin, 2013), pp. 142–159
22. M. Rivain, E. Prouff, Provably secure higher-order masking of AES, in *International Workshop on Cryptographic Hardware and Embedded Systems, CHES*, ed. by S. Mangard, F.-X. Standaert. *Lecture Notes in Computer Science*, vol. 6225 (Springer, Berlin, 2010), pp. 413–427
23. W. Wang, F.-X. Standaert, Y. Yu, S. Pu, J. Liu, Z. Guo, D. Gu, Inner product masking for bitslice ciphers and security order amplification for linear leakages, in *Smart Card Research and Advanced Applications - 15th International Conference, CARDIS 2016, Cannes, France, November 7–9, 2016, Revised Selected Papers*, ed. by K. Lemke-Rust, M. Tunstall. Lecture Notes in Computer Science, vol. 10146 (Springer, Berlin, 2016), pp. 174–191
24. V. Yli-Mäyry, N. Homma, T. Aoki, Improved power analysis on unrolled architecture and its application to PRINCE block cipher, in *Lightweight Cryptography for Security and Privacy - 4th International Workshop, LightSec 2015, Bochum, Germany, September 10–11, 2015, Revised Selected Papers*, ed. by T. Güneysu, G. Leander, A. Moradi. Lecture Notes in Computer Science, vol. 9542 (Springer, Berlin, 2015), pp. 148–163