Masking Tables—An Underestimated Security Risk

Michael Tunstall(✉), Carolyn Whitnall, and Elisabeth Oswald

Department of Computer Science, University of Bristol,
Merchant Venturers Building, Woodland Road,
Bristol BS8 1UB, UK
{michael.tunstall,carolyn.whitnall,elisabeth.oswald}@bristol.ac.uk

Abstract. The literature on side-channel analysis describes numerous masking schemes designed to protect block ciphers at the implementation level. Such masking schemes typically require the computation of masked tables prior to the execution of an encryption function. In this paper we revisit an attack which directly exploits this computation in such a way as to *recover* all or some of the masks used. We show that securely implementing masking schemes is only possible where one has access to a significant amount of random numbers.

Keywords: Side-channel analysis · Secure implementations · Block ciphers

1 Introduction

In recent years a wide range of (higher-order) masking schemes have appeared in the literature. A few of these works are dedicated hardware implementations but the majority are designed to be implemented in embedded software (e.g. as described by Akkar and Giraud [1]), which will be the focus of this article. For instance, Rivain et al. [2] showed how to achieve resistance to second-order DPA (using a table re-masking method). Recent work has discussed affine masking [3], and a hardware-oriented masking scheme proposed by Ishai et al. [4–6].

First- and higher-order masking schemes (i.e. schemes which use one or several random values as masks) are attractive because they (in theory) provide provable security against differential power analysis (DPA) attacks and do not require any specific alterations to a device. In other words, they seem (together with hiding countermeasures) the panacea when it comes to securely implementing ciphers such as AES and DES on otherwise leaky devices (i.e. devices not resistant to DPA).

In this paper we focus on the precomputation based on a rather simple observation: if masks could be extracted by attacking the precomputation, there would be no security at all in the masked encryption rounds. An attacker could simply first extract the masks and use them to correctly predict the masked intermediate values, which would then make a standard DPA attack trivial. Even if an implementation were to use on-the-fly computations of masked S-box tables, if these were vulnerable, then an attack would succeed, as demonstrated by Pan et al. [7].

In this paper we set out to provide a thorough analysis of the application of this type of attack to a variety of state-of-the-art masking approaches when the precomputation is implemented using hiding strategies. We give a thorough theoretical analysis using the evaluation approach suggested by Whitnall and Oswald [8]. This enables us to show, independently of a specific device, how well such attacks work by giving a number of key figures for varying signal-to-noise ratios (SNR, as defined by Mangard et al. [9]), such as the magnitude of resulting correlation coefficients, success probabilities for deriving masks, and the number of traces required for the subsequent key recovery step.

Furthermore, we describe some practical results of attacks on real devices for two representative platforms (an 8-bit and a 32-bit microprocessor). Our results serve as both warning and guidance: they show that the attacks work even with strong hiding countermeasures, and provide information about what SNR is required such that hiding begins to effectively mitigate our attacks.

We have structured our work as follows. We begin by briefly recalling the necessary background with regards to Boolean and affine masking, hiding countermeasures, and the working principle of standard DPA attacks in Sect. 2. Then we explain our attacks against precomputation, including how we model them for our theoretic analysis in Sect. 3. Results of this analysis are provided for all combinations of masking schemes and hiding strategies, for different SNRs. Following on from that we describe our practical processors and setups and report on practical attack outcomes in Sect. 4. We conclude in the last section of the article. After providing references we also use an appendix to collect those tables that are too unwieldy to be included in the main body of this work.

2 Background to Masking, Hiding, and DPA

The masking of intermediate values is a popular software countermeasure in practice (evidence for this is provided by the large number of articles and patents with industrial co-authors [1,3,10,11]). Boolean masking fits well to symmetric encryption schemes (such as AES) and variants such as higher-order masking or affine masking have been the topic of many recent publications. The simple underlying principle of any masking scheme is that, rather than processing the intermediate values (e.g. a key byte, plaintext byte, output of an S-box look-up) directly, one conceals these values with some random value. The hope is that the intermediate value will no longer be predictable and hence the implementation will be secure with regard to (first-order) DPA attacks.

To complicate the adversary's task even further one may also employ hiding techniques. In software this typically means using dummy (or sequences of) instructions (i.e. additional sequences of instructions operating on dummy data, which are indistinguishable from the flow of the actual algorithm) and randomising the sequence of instructions in various ways. Adding dummy instructions is simple but can be costly, moreover recent work points to the inherent difficulty of achieving indistinguishability in practice [12].

In the following sections we introduce details of Boolean and affine masking that are relevant for DPA attacks on the precomputation that we concentrate on. Further, we explain three randomization strategies which are relatively cheap to implement, and to the best of our knowledge are relevant in practice. We complete the necessary background by very briefly explaining Differential Power Analysis (DPA).

2.1 Masking

We now explain the general principle of masking schemes based on Boolean masks. Thereafter we explain how other schemes such as second-order Boolean masking and affine masking are different.

Boolean masks are random values that are exclusive-ored (short XORed) with intermediate values. In the case of AES, this implies that every state byte is masked in this way (whether or not different masks are used for different state bytes depends on efficiency considerations and on the order of DPA attacks one wants to prevent). Similarly, all keys bytes are masked (the decision for different or equal masks again depends on security and efficiency considerations). For example, Herbst et al. [13] give a full explanation of a first-order masking scheme for a typical software implementation of AES[1]. To keep this paper self-contained we briefly summarise how the masked round functions are implemented:

`AddRoundKey` remains the same but operates on masked inputs. We assume key and plaintext mask are different.

`SubBytes` is replaced by a masked table which is precomputed at the beginning of each encryption round using Algorithm 1. There are two random values involved in this precomputation: r, the *address mask* and s, the *data mask*.

`ShiftRows` remains unchanged.

The `MixColumns` function is implemented to ensure that all intermediate values remain masked throughout.

`KeySchedule` remains the same but works on masked data using the same masked substitution table as the masked SubBytes function.

Second-order Boolean Masking: Second-order masking extends first-order masking by applying a second mask to each intermediate value, i.e. a value is represented by three shares ($x = (x_1, x_2, x_3)$, such that $x = x_1 \oplus x_2 \oplus x_3$). A masking scheme for AES following this principle has been described by Rivain et al. [2]. As for Boolean masking, the majority of the round functions remain largely unchanged. However, conducting a `SubBytes` operation becomes problematic because, unlike in first-order masking, it is not possible to 're-use' a

[1] Herbst et al. describe how to mask AES and randomise the flow within rounds (the S-box precomputation is not randomised).

Algorithm 1. Masking a Substitution Table for Boolean Masking.

```
Input: $S$ a 256-byte substitution table, random values $r, s \in \{0, \ldots, 255\}$.
Output: $S'$ a 256-byte masked substitution table.
1 for $i \leftarrow 0$ to 255 do
2     $S'[i] = S[i \oplus r] \oplus s$ ;
3 end
4 return $S'$
```

precomputed table (re-using a table with the same set of masks two or more times would produce a second-order leakage). Consequently, the entire masked S-box needs to be produced when required during the round function. Algorithm 2 shows how to securely compute such an S-box.

Algorithm 2. Masking a Substitution Table for Second-Order Boolean masking [2].

```
Input: $S$ a 256-byte substitution table, random values
       $r_1, r_2, r_3, s_1, s_2 \in \{0, \ldots, 255\}$, and $x'$ where $x = x' \oplus r_1 \oplus r_2$
Output: $S(x) \oplus s_1 \oplus s_2$.
1 $r' = (r_1 \oplus r_2) \oplus r_3$ ;
2 for $i \leftarrow 0$ to 255 do
3     $a = i \oplus r'$ ;
4     $S'[i] = (S[a \oplus x'] \oplus s_1) \oplus s_2$ ;
5 end
6 return $S'[r_3]$
```

Affine Masking: Fumaroli et al. proposed an alternative masking scheme that uses an affine transformation G rather than a Boolean mask [3]. Hence to mask a value x ones applies G where

$$G : \mathbb{F}_{2^8} \longrightarrow \mathbb{F}_{2^8} : x \longmapsto r \cdot x \oplus r' \ ,$$

with randomly chosen mask bytes $r \in \mathbb{F}_{2^8} \setminus \{0\}$ and $r' \in \mathbb{F}_{2^8}$.

Affine masking can be applied to all round functions by adapting the functions accordingly (see Fumaroli et al. [3] for details). As we focus our attacks on those operations relating to the computation required to produce a masked substitution table we only give the algorithm required to generate such a table, see Algorithm 3.

Algorithm 3. Masking a Substitution Table for Affine Masking.

```
Input: $S$ a 256-byte substitution table, $r, r'$ two random values used as masks.
Output: $S$ a 256-byte masked substitution table.
1 for $i \leftarrow 0$ to 255 do
2     $G[i] = r \cdot i \oplus r'$ ;
3 end
4 for $i \leftarrow 0$ to 255 do
5     $S'[i] = G[S[G[i]]]$ ;
6 end
7 return $G, S'$
```

2.2 Hiding

Our focus is on how to generate a masked (S-box) table prior to an encryption run in some random order. Randomly going through the loop indices can be achieved in various ways, and we list the three most generic strategies in order of increasing complexity. Using Algorithm 1 as an example, line 2 would be replaced by

$$S'[f(i)] = S[f(i) \oplus r] \oplus s$$

for some function f.

Random start index. One method to introduce some randomness into the indexing (when looking at multiple runs of the loop as in multiple traces) is to randomly choose the start index. That is

$$f : \{0, \ldots, 255\} \longrightarrow \{0, \ldots, 255\} : x \longmapsto x + k \mod 256 \ ,$$

where a fresh $k \in \{0, \ldots, 255\}$ is generated for each instance of the algorithm. This is also the method that was suggested by Herbst et al. [13].

Random walk. Another simple method, defined by Naccache et al. [14], uses an LFSR to generate a (pseudo)random walk through the indices. That is,

$$f : \{0, \ldots, 255\} \longrightarrow \{0, \ldots, 255\} : x \longmapsto (((x \oplus w) \times u) + y) \oplus z \mod 256 \ ,$$

where a fresh $w, y, z \in \{0, \ldots, 255\}$ and $u \in \{1, 3, \ldots, 255\}$ are generated for each instance of the algorithm.

Random permutation. To go through all the indices one could generate a random permutation of the 256 elements in $\{0, \ldots, 255\}$. However, creating such a random permutation requires the generation of 256 random numbers [15]. Random number generation is costly and one approach to make this more practical is to generate a shorter sequence of random numbers and apply the same sequence repeatedly to the 256 elements. That is,

$$f : \{0, \ldots, 255\} \longrightarrow \{0, \ldots, 255\} : x \longmapsto g_{x \mod n} + m \left\lfloor \frac{x}{n} \right\rfloor \mod 256 \ ,$$

where g is a random sequence of length m given $m|256$ and $n = 256/m$. As previously, a fresh random sequence is generated for each instance of the algorithm. Intuitively, the larger m is, the closer one gets to a truly random permutation.

2.3 Differential Power Analysis

We consider a 'standard' Differential Power Analysis (DPA) scenario as defined by Mangard et al. [16]. That is, we assume that the power consumption T of a cryptographic device depends on some internal value (or state) $F_{k^*}(X)$ which we call the *intermediate value*: a function $F_{k^*} : \mathcal{X} \to \mathcal{Z}$ of some part of the known plaintext (a random variable $X \in \mathcal{X}$) which is dependent on some part of a secret key $k^* \in \mathcal{K}$. Consequently, we have $T = L \circ F_{k^*}(X) + \varepsilon$, where $L : \mathcal{Z} \longrightarrow \mathbb{R}$ describes the data-dependent component and ε contains the remaining power consumption which can be modelled as independent random noise. We consider an attacker who acquires N power measurements corresponding to encryptions of N known plaintexts $x_i \in \mathcal{X}$, $i = 1, \ldots, N$ and wishes to recover the secret key k^*. The attacker can accurately compute the internal values as they would be under each key hypothesis $\{F_k(x_i)\}_{i=1}^N$, $k \in \mathcal{K}$ and uses whatever information available about the true leakage function L to construct a prediction model $M : \mathcal{Z} \to \mathcal{M}$.

DPA is based on the assumption that the power model values corresponding to the correct key hypothesis should have a closer resemblance to true trace measurements than the power model values corresponding to incorrect key hypotheses. This similarity can be measured using the correlation coefficient:

$$D_{\rho,T}(k) = \rho(T, M_k) = \frac{\text{cov}(T, M_k)}{\sqrt{\text{var}(T)}\sqrt{\text{var}(M_k)}} \ . \tag{1}$$

Whitnall and Oswald [8] note that the *nearest rival margin* (i.e., the distance between the correct key and the closest rival hypothesis when the theoretic distinguishing vector[2] is ranked) has a substantial bearing on practical outcomes, because the number of needed power traces (NNT) that are required to detect a statistically significant difference increases as the actual magnitude of the true difference decreases. By defining practically relevant scenarios, it is hence possible to derive true correlation coefficients, examine the resulting margins and then conclude on the number of needed traces (as explained in Chaps. 4 and

[2] The theoretic distinguishing vector represents the underlying values which an attack seeks to estimate, and is computed from known distributions rather than estimated on sampled data.

6 of [9]). The correlation coefficient in an ideal (noise-free) setting scales with the SNR as shown in (2) (which corresponds to (6.5) Chap. 6 of [9]). Given the correlation coefficient corresponding to the correct key ρ_{ck} and the correlation coefficient of the nearest rival ρ_{nr} we can use (3) (which corresponds to (4.43) in [9]) to calculate the NNT. In this equation we choose $\alpha = 0.05$ according to the usual statistical practice).

$$\rho(T, M_k) = \frac{\rho(L \circ F_{k^*}(X), M_k)}{\sqrt{1 + \frac{1}{SNR}}} \tag{2}$$

$$NNT = 3 + 8 \cdot \frac{{z^2}_{1-\alpha}}{\left(\ln \frac{1+\rho_{ck}}{1-\rho_{ck}} - \ln \frac{1+\rho_{nr}}{1-\rho_{nr}}\right)^2} \tag{3}$$

3 Mask Recovery Attacks

In an attack on the precomputation we take a single power consumption trace for one encryption run and extract the part of the trace that corresponds to the precomputation. This trace is then divided up into 256 portions that are then used as a set of traces to conduct a standard DPA. The message is the index i used to control the loop, and the unknowns that we wish to derive are the masks used.

3.1 Boolean Masking

To attack an implementation of Boolean masking (see Algorithm 1) one proceeds by determining the mask r used to blind the address of the S-box table followed by the mask s used to mask the data elements in the table. Note that the application of this strategy does not change when applying it to second-order Boolean masking: in order to target the masked S-box outputs it is sufficient to extract $r_1 \oplus r_2$ and $s_1 \oplus s_2$ as they occur in Algorithm 2—which, in *practice*, is no different to extracting r and s from Algorithm 1. Wherever we present tables and results labelled 'Boolean masking' it should be understood that they relate equally to second- and first-order outcomes.

Masking only. We now explain in more detail how the above description translates into a model that can be used to predict attack outcomes. As per our description, we first attempt to extract r. The attack outcome here is a distinguishing vector that allows us to 'rank' our hypotheses for r. We then use r to determine s. Looking at this differently: we can actually test several values of r and examine the attack outcomes for s in each case (intuitively for incorrect r the recovery of s will completely fail). In our work we settled on allowing a certain number of the best, denoted h, hypotheses for r to be tested with s. Consequently, to model the mask recovery attack for our theoretic analysis we define the $K_{x,h}$ to represent the h highest ranking hypotheses for the variable x. We can then consider the probability of complete mask recovery to be

$$\Pr(r \in K_{r,h}) \cdot \Pr(s \in K_{s,1} | r \text{ is known})$$

We also take into account the probability of *partially* uncovering the masks, by which we mean that our guess at r is correct and our guess at s is incorrect but close (i.e. a short Hamming distance from the correct s)—which is reasonable because the nearest rivals in an attack against an XOR operation *are* of this form. These probabilities can be computed, for any *given number of observations* (i.e., in our case the $N = 256$ trace-segments relating to the loops of the S-box masking procedure), via a formula related to Eqn. (3):

$$\Pr(\rho_{cm} \text{ distinguished from } \rho_{alt}) = 1 - \Phi\left[z_{1-\alpha} - \frac{\left(\ln\frac{1+\rho_{cm}}{1-\rho_{cm}} - \ln\frac{1+\rho_{alt}}{1-\rho_{alt}}\right)}{2 \cdot \sqrt{2/(N-3)}}\right] \quad (4)$$

where ρ_{cm} denotes the correct-hypothesis correlation and ρ_{alt} denotes the correlation produced by the relevant alternative (for example, the h-th ranked candidate for r). The values ρ_{cm} and ρ_{alt} are taken directly from theoretic distinguishing vector. As (4) shows we use the statistical power related to the correct-hypothesis correlation and the relevant alternative to approximate the probabilities for recovering r, and having r ranked among the first $K_{r,h}$ hypotheses respectively. Our method of retaining and confirming h hypotheses means that we are not so concerned with minimising 'false positives'—which corresponds (implicitly) with relaxing the significance criteria. For our theoretic analysis to be meaningful we need to choose, for these computations, a value of α which reflects an attacker's approach in practice, rather than obey typical statistical conventions which impose strong decision criteria as protection against false positives.[3] We settle on $\alpha = 0.2$, which we were able to experimentally confirm *does* align well with the apparent workings of our attack strategy in practice.

Based on these probabilities we can model the success of the subsequent key recovery step carried out in a practical attack. The probabilities for (partial) mask recovery describe how, in effect, an adversary would bias the masks (either remove them if masks are recovered without error, or correctly predict most of the bits effectively leaving only a small bias due to the remaining unknown bits). With this information we can compute theoretic outcomes for the key recovery

[3] Note that in many typical applications of *formal* hypothesis testing—medical treatment evaluation, for example—false positives have serious consequences. Competent analysts will opt to increase their sample sizes rather than weaken their decision criteria in order to get conclusive results. Since the unmasking phase of a mask recovery attack is constrained to a sample size of 256 an attacker does not have this option, nor are the consequences of a false positive so 'expensive'.

step and use the nearest rival margins to obtain the number of needed traces (for the entire attack) in practice for a given SNR[4].

Table 1 lists the outcomes of these theoretic, modelled attacks for different SNRs (where $h = 10$). The top line states the SNR level, increasing from high noise at the left, towards no noise on the right. The second table line then lists the percentage of masks fully recovered, and the third line lists the percentage of masks partially recovered (a single-bit error). The numbers show that, up to an SNR of two, full mask recovery is possible, but afterwards only partial recovery is possible. The precise cut-off point for full recovery is 1.897 as we determined in our theoretic model. The fourth and fifth line list then the values of the correct key correlation and the margin to the nearest rival in the key recovery step. This margin actually translates into the number of needed power traces. As the values show up to an SNR of 2^{-1} the attack is basically equally effective as would be a standard DPA attack on an unprotected device.

Table 1. Data complexity of mask recovery attacks against a Boolean masked AES S-box (straightforward pre-computation phase).

	2^{-5}	2^{-3}	2^{-1}	2^{1}	2^{3}	2^{5}	2^{7}	Pure signal
S-box unmasked	29.4	56.9	91.2	100.0	100.0	100.0	100.0	100.0
S-box partially unmasked	55.0	42.7	8.8	0.0	0.0	0.0	0.0	0.0
Correct key correlation	0.123	0.296	0.565	0.816	0.943	0.985	0.996	1.000
Nearest rival margin	0.100	0.241	0.459	0.663	0.766	0.800	0.809	0.812
Traces needed	538	90	22	8	5	4	4	3

Masking and hiding. We now investigate how the three hiding strategies we listed before impact on the effectiveness of the mask recovery attacks. We briefly describe how the countermeasures change the model we detailed before. When the starting index for the precomputation is chosen randomly, the first step of the unmasking procedure attempts to recover the index i *and* the address mask r, by trying each pair. In fact there is irresolvable ambiguity between two equally ranked hypotheses—the correct pair (r,i) and the shifted pair ($r+128 \mod 256$, $i+128 \mod 256$). Fortunately, this does not pose an obstacle to recovering the mask on the S-box output, as either pair will produce the same unmasked address

[4] Throughout this paper we assume in our models that the device leaks the Hamming weight and that the adversary uses this as power model. This is not a shortcoming for several reasons. Firstly, the numbers we provide are independent of the actual power model; they do however depend on the fact that we assume that the adversary's model effectively matches the leakage model of the device. If an adversary were to use an imprecise model, this would change the outcomes and the analysis would need to be done accounting for the imprecision. Secondly, leakages observed in practice from software implementations on small processors originate typically from transfers of intermediate data or address values over buses. These components typically leak the Hamming weight or the Hamming distance from some fixed value.

and therefore provide the predicted values for the second stage mask recovery attack.

When the pre-computation is performed according to the ordering given by an LFSR, the LFSR function itself must be recovered, which requires more attack steps and leads to a larger aggregate loss of precision. However, it is still feasible. If the index function is of the form

$$f : \{0, \ldots, 255\} \longrightarrow \{0, \ldots, 255\} : x \longmapsto (((x \oplus w) \times u) + y) \oplus z \mod 256 \;,$$

then, by retaining the top h hypotheses at every step (which in practice is usually smaller than for the standard attack—we take $h = 4$ in our analysis, to represent an attacker's response to the increased computational complexity), and using the following step as confirmation, we estimate the proportion unmasked as:

$$\begin{aligned}&\Pr(w \in K_{w,h}) \cdot \Pr(x \in K_{x,h} | w \text{ is known}) \cdot \Pr(y \in K_{y,h} | w, x \text{ are known}) \\ &\quad \cdot \Pr(z \oplus r \in K_{z \oplus r,h} | w, x, y \text{ are known}) \cdot \Pr(s \in K_{s,1} | w, x, y, z \oplus r \text{ are known}),\end{aligned}$$

noting that we are unable to recover r as distinct from z, but that, for the purposes of unmasking the address, it is sufficient to recover the XOR between the two.

The theoretic analysis for attacks against the implementation which permutes the indices in aligned blocks before precomputing the masked table is slightly more complicated because one must take into account the probability of uncovering only a proportion of the columns (see Sect. 2.2 for notation). Additionally, as with the random start index variant, there remains ambiguity over the correct column and mask pair: each column hypothesis will result in a maximal peak for a certain hypothesis on the mask (From an information theoretic perspective, it is clear that we cannot expect to recover 10 bits of information from an 8-bit target value). However, all of these pairs reproduce the *same* (correct) 8-bit unmasked address value, and since this is what we need for the second stage output unmasking the ambiguity does not matter.

The proportion unmasked is estimated (via the Law of Total Probability[5]) as:

$$\begin{aligned}&\sum_{c=1}^{n} \Pr(c \text{ columns are unmasked}) \cdot \Pr(s \in K_{s,1} | c \text{ columns are unmasked}) \\ &\qquad = \sum_{c=1}^{n} \binom{n}{c} \cdot \Pr(\text{column unmasked})^{c} \cdot (1 - \Pr(\text{column unmasked}))^{n-c} \\ &\qquad\qquad \cdot \Pr(s \in K_{s,1} | c \text{ columns are unmasked})\end{aligned}$$

Table 2 (which is laid out similarly to Table 1) presents the theoretic mask recovery rates and subsequent key recovery performance for the hiding countermeasures. The attack remains (theoretically) successful against all countermeasures,

[5] Law of Total Probability states that if $\{B_n : n = 1, 2, 3, \ldots\}$ is a finite or countably infinite partition of a sample space and each event B_n is measurable, then for any event A, $\Pr(A) = \sum_n \Pr(A \cap B_n) = \sum_n \Pr(A|B_n) \Pr(B_n)$.

Table 2. Data complexity of mask recovery attacks against a Boolean masked AES S-box with hiding countermeasures.

	2^{-5}	2^{-3}	2^{-1}	2^{1}	2^{3}	2^{5}	2^{7}	Pure signal
Randomised start index								
S-box unmasked	20.5	50.1	91.1	100.0	100.0	100.0	100.0	100.0
S-box partially unmasked	41.1	41.3	8.9	0.0	0.0	0.0	0.0	0.0
Correct key correlation	0.089	0.270	0.564	0.816	0.943	0.985	0.996	1.000
Nearest rival margin	0.073	0.220	0.459	0.663	0.766	0.800	0.809	0.812
Traces needed	1024	109	22	8	5	4	4	3
Random walk (LFSR)								
S-box unmasked	1.1	8.6	50.9	97.6	100.0	100.0	100.0	100.0
S-box partially unmasked	3.9	21.0	40.2	2.4	0.0	0.0	0.0	0.0
Correct key correlation	0.01	0.08	0.47	0.81	0.94	0.98	1.00	1.00
Nearest rival margin	0.01	0.07	0.38	0.66	0.77	0.80	0.81	0.81
Traces needed	169275	1249	34	9	5	4	4	3
Permuted in 4 columns								
S-box unmasked	24.1	42.7	87.4	100.0	100.0	100.0	100.0	100.0
S-box partially unmasked	31.7	38.1	12.0	0.0	0.0	0.0	0.0	0.0
Correct key correlation	0.083	0.237	0.557	0.816	0.943	0.985	0.996	1.000
Nearest rival margin	0.068	0.193	0.452	0.663	0.766	0.800	0.809	0.812
Traces needed	1175	142	23	8	5	4	4	3
Permuted in 8 columns								
S-box unmasked	23.3	33.0	65.6	99.5	100.0	100.0	100.0	100.0
S-box partially unmasked	23.0	28.1	23.2	0.5	0.0	0.0	0.0	0.0
Correct key correlation	0.071	0.180	0.479	0.815	0.943	0.985	0.996	1.000
Nearest rival margin	0.057	0.146	0.389	0.662	0.766	0.800	0.809	0.812
Traces needed	1644	249	32	9	5	4	4	3
Permuted in 16 columns								
S-box unmasked	23.3	28.7	45.3	86.9	100.0	100.0	100.0	100.0
S-box partially unmasked	18.7	21.0	22.9	9.0	0.0	0.0	0.0	0.0
Correct key correlation	0.065	0.148	0.361	0.765	0.943	0.985	0.996	1.000
Nearest rival margin	0.053	0.121	0.293	0.621	0.766	0.800	0.809	0.812
Traces needed	1933	369	59	10	5	4	4	3
Permuted in 32 columns								
S-box unmasked	22.9	26.2	33.1	47.2	75.3	96.7	99.4	99.4
S-box partially unmasked	15.9	16.0	16.2	16.5	12.2	2.5	0.6	0.6
Correct key correlation	0.061	0.127	0.261	0.487	0.797	0.971	0.994	0.998
Nearest rival margin	0.049	0.103	0.212	0.395	0.647	0.789	0.808	0.811
Traces needed	2223	502	117	31	9	4	4	3

although the noise threshold at which mask recovery begins to deteriorate varies. For the randomised start index this threshold is 1.897, for the random walk it is 9.409, for the column-wise permutations it is 3.959, 9.029, and 25.260 for the 4-, 8- and 16-column variants respectively, whilst for the 32-column variant irresolvable ambiguity on some of the columns means that the masks can never be perfectly recovered, even from noise-free leakage.

3.2 Affine Masking

The attack on the affine masking scheme requires the recovery of a multiplicative and a Boolean mask. As is clear from Algorithm 3, we cannot recover the Boolean mask r' without having first recovered the multiplicative mask r. But once we have recovered the Boolean mask r' we can use it to 'confirm' the correctness of the multiplicative mask r.

Masking only. The strategy for recovering the multiplicative and additive components of an affine-masked S-box output is slightly different. By retaining the top $h = 10$ (say) candidates on the multiplicative mask, then looking at the highest peak produced by the additive hypotheses for each of the 10, we hope to confirm the correct multiplicative mask at the same time as discovering the correct additive hypothesis. Because the input and output are masked with the same values we only need recover the two, e.g. by attacking the pre-computation of the affine transformation look-up table. If the outputs in the masked S-box pre-computation can be identified and targeted then the nonlinearity of the S-box improves the recovery of the second, additive mask—otherwise the margin between the correct mask and the incorrect alternatives will be small, as always when attacking a Boolean addition. We have produced two versions of the analysis accordingly—one where we suppose the S-box structure may be exploited, one where we suppose it cannot. These are presented in Table 3, from which we see that, when the S-box nonlinearity is exploited, the affine masked table precomputation is more vulnerable to mask recovery than the Boolean masked table pre-computation (the SNR thresholds at which the mask recovery begins to degrade are 0.500 when the S-box is exploited in the mask recovery stage, and 1.897—the same as for the Boolean masking—when it is not). However, the more complex nature of the mask application means that any imperfection in the mask recovery incurs a greater penalty on the number of traces needed for the *key recovery* stage (compared to the attacks on Boolean masking), so that in noisy scenarios the affine scheme is the more resilient to the *overall* attack strategy.

Masking and hiding. For the deliberately-complicated versions of the masking schemes, different problems are associated with recovering the affine transformations to those which are associated with recovering the Boolean transformations. In particular, there are far more cases where ambiguity prevents recovering the correct pairs with any confidence. In the analysis, we have generally adopted the approach that, where c candidate pairs are equally theoretically ranked, the probability of recovering the correct one is taken to be $\frac{1}{c}$-times the probability that the c will stand out together. That is, we cannot, except by chance, distinguish it from the others, but will be able to unmask a proportion $(\frac{1}{c} \times \Pr(\text{top set correctly identified}))$ which will still help us in the key-recovery phase of the attack.

The permuted columns variant requires particular adaptation, as there is increasing ambiguity as the size of the permutation increases, with some even producing constant leakage by virtue of the form of the affine transformation

Table 3. Data complexity of mask recovery attacks against an affine masked AES S-box.

	2^{-5}	2^{-3}	2^{-1}	2^{1}	2^{3}	2^{5}	2^{7}	Pure signal
Exploiting the S-box								
Both masks recovered	57.7	97.4	100.0	100.0	100.0	100.0	100.0	100.0
Multiplicative mask recovered	19.6	2.0	0.0	0.0	0.0	0.0	0.0	0.0
Correct key correlation	0.101	0.325	0.577	0.816	0.943	0.985	0.996	1.000
Nearest rival margin	0.082	0.264	0.469	0.663	0.766	0.800	0.809	0.812
Traces needed	804	74	21	8	5	4	4	3
Not exploiting the S-box								
Both masks recovered	27.2	56.3	91.2	100.0	100.0	100.0	100.0	100.0
Multiplicative mask recovered	33.7	32.8	8.8	0.0	0.0	0.0	0.0	0.0
Correct key correlation	0.048	0.188	0.526	0.816	0.943	0.985	0.996	1.000
Nearest rival margin	0.037	0.151	0.430	0.663	0.766	0.800	0.809	0.812
Traces needed	3911	233	26	9	5	4	4	3

(this does not happen with the Boolean masking). For a theoretic analysis, it is tricky in places to approximate the *best* that can be achieved by a canny attacker because different ways of combining the information and confirming candidate hypotheses will inevitably produce different outcomes, and it is not possible to explore and evaluate them all. We propose a strategy whereby each column is attacked separately (searching over the column index space as well as the mask space) and then the recovered affine transformation candidates are compared over the columns to find the most likely. Accordingly, the proportion unmasked for the key-recovery stage is computed as the probability of the correct transformation achieving a majority vote.

The results corresponding to the modelling of these attacks can be found in Tables 6 and 7 of Appendix A. Essentially, they show that the attacks are less efficient than on the Boolean scheme, but that we can still expect them to succeed for realistic platforms (they work for very low SNRs).

4 Theory Put to Practice

To gain some insight into the practical effectiveness of such attacks we performed some of them on two platforms, an 8-bit and a 32-bit microprocessor. The 8-bit microprocessor was an AT89S5253, which has an 8051 architecture. In this case acquisitions were taken with a sampling rate of 500 MS/s and a clock speed of 11 MHz. No filtering was conducted since this did not have any impact on the SNR. The 32-bit microprocessor was an ARM7TDMI microprocessor, where acquisitions were taken with a sampling rate of 200 MS/s and a clock speed of 7.3728 MHz. These acquisitions were filtered using a low-pass filter with a corner frequency at 7.3728 MHz to improved the SNR.

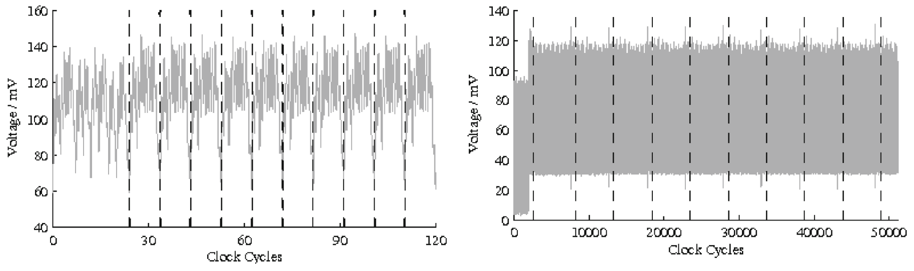

Fig. 1. The above traces show the instantaneous power consumption during the first ten rounds of Algorithm 1. The left trace corresponds to the AT89 microprocessor and the right traces to the ARM microprocessor. The power consumption showing the individual rounds are delimited by dashed lines.

The SNR (as defined by Mangard et al. [9]) of these two setups is rather different: the 8-bit controller features a very strong signal such that the overall SNR is about 22, whereas the 32-bit processor only delivers an SNR of 0.54.

Boolean masking requires a simple precomputation as described in Algorithm 1 and Algorithm 2 resp. As these algorithms suggest, one can see distinct patterns corresponding to the 256 loops when inspecting power traces corresponding to the execution of these algorithm on a device[6]. This is demonstrated in Fig. 1 where the rounds of Algorithm 1 are clearly visible.

Our experiments showed that on both platforms mask recovery worked almost perfectly. To provide some meaningful and statistically sound numbers we repeated the experiment 1000 times with different masks and produced the results shown in Table 4. These numbers give the error rates for recovering the masks r and s in Algorithm 1, and show clearly that for both platforms the fact that we have 256 traces available is sufficient to recover the masks even with the relatively poor SNR of the 32-bit platform. Note that proportions of data masks recovered with zero-bit errors correspond to the first row of Table 1 ("S-box un-masked"), while the proportions recovered with one-bit errors relate to the second row ("S-box partially un-masked"). The SNRs of the two devices mean that both can be expected to lead to almost perfect mask recovery (as indicated by the first two rows in Table 1), which is reflected in our practical experiments. Some results of the AT89 attacks are somewhat peculiar: we consistently observed a single bit error in the recovered data masks (but not always for the same bit). We are currently unable to explain this behavior in any satisfying way.

The introduction of simple hiding strategies has almost no impact, only a sufficiently strong permutation starts to degrade the attack performance in practice. We show some more results giving the error rates for data mask recovery for the ARM7 platform in Table 5. The numbers indicate that, as the size of the permutation increases, the distribution of the error rate approaches a binomial

[6] The practical attacks applied to an implementation of second-order Boolean masking.

Table 4. The error rates for identifying masks for implementations of Boolean masking.

	Address mask					Data mask				
Error (bits)	0	1	2	3	4+	0	1	2	3	4+
ARM	0.99	0.0012	0.0020	0.00075	0.00020	0.92	0.075	0.0030	0.00075	0.0029
AT89	0.98	0.0081	0.0079	0.0067	0.00010	0	0.98	0.0027	0.0047	0.015

distribution where one would not be able to conduct an attack. All the permutation lengths tested would lead to a viable attack, we refer the reader to Mangard et al. [9] for a description of how to compute the number of traces required to conduct an attack.

Table 5. Error rates for Boolean masking using different hiding strategies.

	Data mask, ARM								
Error (bits)	0	1	2	3	4	5	6	7	8
RSI	0.94	0.035	0.0040	0.0060	0.0080	0.0030	0	0.0010	0
Random walk	0.35	0.52	0.11	0.011	0.0070	0.0040	0.0020	0.0010	0
Random permutation:									
$n = 4$	0.84	0.093	0.017	0.016	0.013	0.012	0.0070	0	0
$n = 8$	0.47	0.15	0.11	0.066	0.10	0.061	0.030	0.0070	0
$n = 16$	0.064	0.11	0.19	0.23	0.21	0.12	0.065	0.015	0.0020
$n = 32$	0.011	0.052	0.13	0.25	0.27	0.19	0.081	0.015	0.0020

5 Conclusion

Masking schemes are popular in the literature, as indicated by the large number of publications in this area. Claims about the security of these schemes are typically supported by evaluation with regards to what (higher) order DPA attacks they can resist, but no focus has yet been put on scrutinising the practically inevitable precomputation of masked tables.

After explaining, for the most common and practically relevant masking approaches, how to randomize the precomputation step, we analyze the security of the resulting implementations using both a theoretic approach and practical implementations. For the theoretic analysis we explain how to model our attacks and what this allows us to conclude about the percentage of masks recovered, nearest rival margins and hence the number of needed power traces for different SNRs. This analysis is generic and to some extent independent of the power model (it can be adapted to incorporate other models).

These theoretic results indicate that our attacks are likely to work in practice, since we see good theoretic results even for low SNRs (with the exception of the largest permutation). In the penultimate section of this paper we showed results of actual attacks on two platforms. They tally with our theoretic outcomes and hence confirm that our attacks are indeed highly relevant and applicable to practice. Without much effort we can break any of the implementations employing masking and hiding in the precomputation.

Our results provide both a warning and some guidance. The warning is that, without substantial extra effort to secure the computation of masked tables, this operation will most likely leak the masks and hence render the masking of the round function pointless. The guidance that we can give is with regards to the SNR that needs to be achieved for the discussed randomisation strategies to have some impact. Even if the device SNR itself is fixed, one can attempt to use dummy instructions (bearing recent results in mind [12]) to lower the SNR by desynchronising the loops in the precomputation. Given that the discussed randomisation strategies themselves lead to a significant performance penalty (more randomness required, increased effort in computing data and address values), a further performance loss might however be unacceptable in practical applications. Our final conclusion is hence rather pessimistic: precisely for the devices in which masking seems an inescapable necessity, the computation of masked tables will most likely render the scheme insecure.

Acknowledgments. The work described in this paper has also been supported in part the European Commission through the ICT Programme under Contract ICT-2007-216676 ECRYPT II and the EPSRC via grant EP/I005226/1.

A Tables

Table 6. Data complexity of mask recovery attacks against an affine masked AES S-box with hiding countermeasures, when the nonlinearity of the S-box is exploited in the mask-recovery stage. (SNR thresholds of degradation—where applicable—in parentheses).

	2^{-5}	2^{-3}	2^{-1}	2^{1}	2^{3}	2^{5}	2^{7}	Pure signal
Randomised start index (0.73)								
Both masks recovered	42.3	85.1	99.9	100.0	100.0	100.0	100.0	100.0
Multiplicative mask recovered	26.7	11.2	0.111	$2.36e-010$	0.000	0.000	0.000	0.000
Correct key correlation	0.0743	0.284	0.577	0.816	0.943	0.985	0.996	1.00
Nearest rival margin	0.0594	0.232	0.469	0.663	0.766	0.800	0.809	0.812
Traces needed	$1.53e+003$	96.8	21.0	8.50	5.16	4.07	3.63	3.02
Random walk (LFSR) (4.49)								
Both masks recovered	0.380	4.81	46.5	97.6	100.0	100.0	100.0	100.0
Multiplicative mask recovered	0.632	5.02	24.9	2.39	$3.30e-007$	0.000	0.000	0.000
Correct key correlation	0.00149	0.0286	0.376	0.811	0.943	0.985	0.996	1.00
Nearest rival margin	0.00121	0.0232	0.306	0.659	0.766	0.800	0.809	0.812
Traces needed	$3.71e+006$	$1.00e+004$	54.2	8.67	5.16	4.07	3.63	3.02
Permuted in 4 columns (NA)								
Both masks recovered	8.86	39.2	92.6	98.4	98.4	98.4	98.4	98.4
Multiplicative mask uncovered	32.3	35.4	6.85	1.58	1.57	1.57	1.57	1.57
Correct key correlation	0.0576	0.219	0.564	0.813	0.939	0.981	0.992	0.996
Nearest rival margin	0.0468	0.178	0.458	0.661	0.763	0.797	0.806	0.809
Traces needed	$2.47e+003$	167	22.1	8.6	5.24	4.19	3.81	3.63
Permuted in 8 columns (NA)								
Both masks recovered	0.125	1.29	24.2	89.5	95.6	95.7	95.7	95.7
Multiplicative mask uncovered	12.3	24.3	47.8	10.3	4.35	4.32	4.32	4.32
Correct key correlation	0.0162	0.0651	0.347	0.794	0.933	0.974	0.985	0.989
Nearest rival margin	0.0132	0.0529	0.282	0.646	0.758	0.791	0.801	0.804
Traces needed	$3.12e+004$	$1.93e+003$	64.6	9.21	5.4	4.38	4.05	3.92
Permuted in 16 columns (NA)								
Both masks recovered	$1.16e-005$	0.000135	0.0116	4.96	81	96.1	96.4	96.4
Multiplicative mask uncovered	0.857	1.66	6.05	41.5	18.8	3.95	3.61	3.61

Table 6. (*Continued*)

Correct key correlation	0.00112	0.00414	0.0263	0.294	0.897	0.975	0.987	0.991
Nearest rival margin	0.000909	0.00337	0.0214	0.239	0.728	0.792	0.802	0.805
Traces needed	6.55e + 006	4.78e + 005	1.19e + 004	90.9	6.28	4.35	3.99	3.86
Permuted in 32 columns (NA)								
Both masks recovered	1.49e − 014	7.11e − 014	1.99e − 012	2.1e − 009	7.97e − 005	0.775	35.7	76.1
Multiplicative mask uncovered	0.000117	0.000145	0.000248	0.00111	0.0443	4.98	29	17
Correct key correlation	1.52e − 007	3.64e − 007	1.07e − 006	6.79e − 006	0.000314	0.0444	0.572	0.889
Nearest rival margin	1.24e − 007	2.96e − 007	8.72e − 007	5.52e − 006	0.000255	0.0361	0.465	0.722
Traces needed	3.54e + 014	6.2e + 013	7.11e + 012	1.78e + 011	8.32e + 007	4.16e + 003	21.3	6.47

Table 7. Data complexity of mask recovery attacks against an affine masked AES S-box with hiding countermeasures, when the nonlinearity of the S-box is *not* exploited in the mask-recovery stage. (SNR thresholds of degradation—where applicable—in parentheses).

	2^{-5}	2^{-3}	2^{-1}	2^{1}	2^{3}	2^{5}	2^{7}	Pure signal
Randomised start index (1.90)								
Both masks recovered	19.9	49.2	91.1	100.0	100.0	100.0	100.0	100.0
Multiplicative mask recovered	37.1	38.2	8.86	0.00274	0.000	0.000	0.000	0.000
Correct key correlation	0.0355	0.165	0.526	0.816	0.943	0.985	0.996	1.00
Nearest rival margin	0.0245	0.131	0.430	0.663	0.766	0.800	0.809	0.812
Traces needed	8.99e + 003	312.0	25.7	8.50	5.16	4.07	3.63	3.02
Random walk (LFSR) (4.49)								
Both masks recovered	0.176	1.97	28.9	95.2	100.0	100.0	100.0	100.0
Multiplicative mask recovered	0.633	5.17	33.1	4.71	6.59e − 007	0.000	0.000	0.000
Correct key correlation	0.00113	0.0195	0.310	0.806	0.943	0.985	0.996	1.00
Nearest rival margin	0.000919	0.0158	0.252	0.655	0.766	0.800	0.809	0.812
Traces needed	6.41e + 006	2.16e + 004	81.6	8.84	5.16	4.07	3.63	3.02

Table 7. (*Continued*)

Permuted in 4 columns (NA)								
Both masks recovered	4.35	14.9	51.2	87.4	92.3	92.4	92.4	92.4
Multiplicative mask uncovered	33.9	49.5	45	12.6	7.68	7.64	7.64	7.64
Correct key correlation	0.0519	0.173	0.49	0.791	0.925	0.966	0.977	0.981
Nearest rival margin	0.0421	0.141	0.398	0.643	0.751	0.785	0.794	0.797
Traces needed	3.04e + 003	269	30.4	9.34	5.59	4.59	4.3	4.19
Permuted in 8 columns (NA)								
Both masks recovered	0.0339	0.254	4.63	47	79.7	81.7	81.8	81.8
Multiplicative mask uncovered	12.3	24.6	60.1	52.5	20.3	18.3	18.2	18.2
Correct key correlation	0.0161	0.0623	0.287	0.705	0.895	0.94	0.951	0.954
Nearest rival margin	0.0131	0.0506	0.233	0.573	0.727	0.764	0.772	0.775
Traces needed	3.17e + 004	2.11e + 003	95.8	12.8	6.32	5.23	4.97	4.88
Permuted in 16 columns (NA)								
Both masks recovered	1.82e − 006	1.42e − 005	0.000619	0.16	9.35	26.1	28.2	28.2
Multiplicative mask uncovered	0.857	1.66	6.05	43.6	89.7	73.9	71.8	71.8
Correct key correlation	0.00112	0.00414	0.0262	0.268	0.723	0.803	0.817	0.821
Nearest rival margin	0.000909	0.00337	0.0213	0.218	0.587	0.652	0.664	0.667
Traces needed	6.55e + 006	4.78e + 005	1.19e + 004	110	12	8.94	8.47	8.37
Permuted in 32 columns (NA)								
Both masks recovered	8.73e − 019	4.03e − 018	1.03e − 016	7.99e − 014	1.36e − 009	6.57e − 006	0.000549	0.00234
Multiplicative mask uncovered	0.000117	0.000145	0.000248	0.00111	0.0443	5.01	45.2	71.3
Correct key correlation	1.52e − 007	3.64e − 007	1.07e − 006	6.79e − 006	0.000313	0.037	0.337	0.535
Nearest rival margin	1.24e − 007	2.96e − 007	8.72e − 007	5.52e − 006	0.000254	0.0301	0.274	0.435
Traces needed	3.54e + 014	6.2e + 013	7.11e + 012	1.78e + 011	8.36e + 007	5.97e + 003	68.4	25

References

1. Akkar, M.-L., Giraud, C.: An implementation of DES and AES, secure against some attacks. In: Koç, Ç.K., Naccache, D., Paar, C. (eds.) CHES 2001. LNCS, vol. 2162, pp. 309–318. Springer, Heidelberg (2001)
2. Rivain, M., Dottax, E., Prouff, E.: Block ciphers implementations provably secure against second order side channel analysis. In: Nyberg, K. (ed.) FSE 2008. LNCS, vol. 5086, pp. 127–143. Springer, Heidelberg (2008)
3. Fumaroli, G., Martinelli, A., Prouff, E., Rivain, M.: Affine masking against higher-order side channel analysis. In: Biryukov, A., Gong, G., Stinson, D.R. (eds.) SAC 2010. LNCS, vol. 6544, pp. 262–280. Springer, Heidelberg (2011)
4. Ishai, Y., Sahai, A., Wagner, D.: Private circuits: securing hardware against probing attacks. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 463–481. Springer, Heidelberg (2003)
5. Rivain, M., Prouff, E.: Provably secure higher-order masking of AES. In: Mangard, S., Standaert, F.-X. (eds.) CHES 2010. LNCS, vol. 6225, pp. 413–427. Springer, Heidelberg (2010)
6. Kim, H.S., Hong, S., Lim, J.: A fast and provably secure higher-order masking of AES S-box. In: Preneel, B., Takagi, T. (eds.) CHES 2011. LNCS, vol. 6917, pp. 95–107. Springer, Heidelberg (2011)
7. Pan, J., den Hartog, J.I., Lu, J.: You cannot hide behind the mask: power analysis on a provably secure S-box implementation. In: Youm, H.Y., Yung, M. (eds.) WISA 2009. LNCS, vol. 5932, pp. 178–192. Springer, Heidelberg (2009)
8. Whitnall, C., Oswald, E.: A fair evaluation framework for comparing side-channel distinguishers. J. Cryptogr. Eng. **1**(2), 145–160 (2011)
9. Mangard, S., Oswald, E., Popp, T.: Power Analysis Attacks: Revealing the Secrets of Smart Cards. Springer, New York (2007)
10. Chari, S., Jutla, C.S., Rao, J.R., Rohatgi, P.: Towards sound approaches to counteract power-analysis attacks. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 398–412. Springer, Heidelberg (1999)
11. Genelle, L., Prouff, E., Quisquater, M.: Thwarting higher-order side channel analysis with additive and multiplicative maskings. In: Preneel, B., Takagi, T. (eds.) CHES 2011. LNCS, vol. 6917, pp. 240–255. Springer, Heidelberg (2011)
12. Durvaux, F., Renauld, M., Standaert, F.-X., van Oldeneel tot Oldenzeel, L., Veyrat-Charvillon, N.: Efficient removal of random delays from embedded software implementations using hidden Markov models. In: Mangard, S. (ed.) CARDIS 2012. LNCS, vol. 7771, pp. 123–140. Springer, Heidelberg (2013)
13. Herbst, C., Oswald, E., Mangard, S.: An AES smart card implementation resistant to power analysis attacks. In: Zhou, J., Yung, M., Bao, F. (eds.) ACNS 2006. LNCS, vol. 3989, pp. 239–252. Springer, Heidelberg (2006)
14. Naccache, D., Nguyên, P.Q., Tunstall, M., Whelan, C.: Experimenting with faults, lattices and the DSA. In: Vaudenay, S. (ed.) PKC 2005. LNCS, vol. 3386, pp. 16–28. Springer, Heidelberg (2005)
15. Knuth, D.: The Art of Computer Programming, vol. 2, 3rd edn. Addison-Wesley, Reading (1998)
16. Mangard, S., Oswald, E., Standaert, F.X.: One for all–all for one: unifying standard DPA attacks. IET Inf. Secur. **5**(2), 100–119 (2011)