Bounded, yet Sufficient? How to Determine Whether Limited Side Channel Information Enables Key Recovery

Xin Ye, Thomas Eisenbarth and William Martin

Worcester Polytechnic Institute, Worcester, MA 01609, USA
{xye,teisenbarth,martin}@wpi.edu

Abstract. This work presents a novel algorithm to quantify the relation between three factors that characterize a side channel adversary: the amount of observed side channel leakage, the workload of full key recovery, and its achievable success rate. The proposed algorithm can be used by security evaluators to derive a realistic bound on the capabilities of a side channel adversary. Furthermore, it provides an optimal strategy for combining subkey guesses to achieve any predefined success rate. Hence, it can be used by a side channel adversary to determine whether observed leakage suffices for key recovery before expending computation time. The algorithm is applied to a series of side channel measurements of a microcontroller AES implementation and simulations. A comparison to related work shows that the new algorithm improves on existing algorithms in several respects.
Keywords: Side Channel Analysis, Security Evaluation, Guesswork, Full Key Recovery, Weak Maximum Likelihood.

1 Motivation

Side channel analysis (SCA) of embedded cryptographic implementations has been studied for more than 15 years [6,8]. Recently, there has been a growing interest in studying and quantifying the amount of information that can be extracted from a *limited* number of side channel observations. Knowing how much leakage actually suffices for a *full* key recovery is of high practical relevance. This question is closely tied to the computational capabilities of the side channel adversary, since SCA often include an extensive key search component. A good comparison of algorithms using tradeoffs between side channel information and computation are the submissions to the DPA contest [1], where the success metric was solely based on the number of needed observations, without a clear limitation of computation. Another emerging trend in SCA are new attacks that are made feasible only by tapping into the massive parallel computing power as provided by GPUs, such as [12]. This indicates that computational power of the adversary needs to be considered as part of side channel security metrics. Finally, *leakage resilient cryptography* usually assumes limited leakage of a given key or secret state (c.f. [7,4,11,19]) before it is updated. The schemes provide security if an

adversary cannot successfully exploit more than the bounded leakage. In all of these cases, it is of high interest to know how much leakage the adversary can get from the observed measurements. Closely related is the question of the remaining attack complexity—given the limited side channel information—and the resulting search strategy.

So far only little effort has been put into the quantification of the remaining computational complexity when limited leakage is available but insufficient to narrow the key space down to a simply searchable size. While systematic metrics to quantify side channel leakage exist [10,21,17,18], many of them perform relative comparisons of implementations or attacks [5,9,16]. The most promising approach has been presented in [22,23]. The authors present a full key enumeration algorithm [22] as well as a key ranking algorithm [23] in the case where only limited side channel leakage can be extracted. These algorithms enables estimating the remaining full key recovery complexity even if the experimental verification is infeasible. However their algorithms assume the correct key to be known. In other words, their results can be used by evaluation labs, but not by the key recovering adversary.

Our Contribution This work proposes an alternative approach to evaluate side channel security for full key recovery attacks. The security level is expressed as the relation between the amount of observed leakage, the success probability and the necessary attack complexity. Following this approach, a constructive Key Space Finding (KSF) algorithm is presented. It not only provides an estimation on the remaining guessing complexity, but also allows the adversary, for the first time, to derive a probabilistic winning strategy for each specific side channel observation. Further, by statistical bootstrapping the size of returned key spaces, the algorithm can also be used by evaluation labs to approximate a *realistic* security level for various extents of information leakage.

2 Background

This section formalizes common assumptions for SCA and revisits useful metrics and algorithms that quantifies side channel leakages.

2.1 Adversarial Model and Notations

Most SCA follow the divide-and-conquer strategy. The full secret key sk is divided into b parts, i.e. $sk = k_1 \| ... \| k_b$ where each subkey k_i is a n bit string. In the general setting of SCA, an adversary runs the crypto algorithm $E_{sk}()$ and records side channel observations. This is followed by a leakage exploitation phase where the adversary independently recovers information about each subkey k_i from the measured observations. We assume the adversary to be q-limited, i.e. she can run the algorithm up to q times and get the respective leakages. We denote the inputs as $\boldsymbol{X}^q = [X_{i,j}]_{b \times q} \in (\mathbb{F}_2^n)^{bq}$ where each row $[X_{i,j}]$ with $1 \leq j \leq q$ corresponds to the inputs at the i-th part for the q queries. Similarly,

the leakages are denoted as $\boldsymbol{L}^q = [L_{i,j}]_{b\times q} \in \mathbb{R}^{bq}$. Each row $[L_{i,j}]$ with $1 \le j \le q$ represents the q leakage observations related to the i-th subkey part k_i. With the knowntexts (either plaintexts or ciphertexts) $\boldsymbol{X}^q$ and leakages $\boldsymbol{L}^q$, the adversary chooses a side channel distinguisher such as DPA, CPA, MIA, template attack, etc. and outputs an *ordered* list of subkey candidates $g_{i,[1]}, g_{i,[2]}, ...$ for guessing the correct subkey k_i. Here, $[\cdot]$ indicates a reordering of subkey candidates in the order of decreasing likelihood, i.e. $g_{i,[j]}$ refers the j-th most likely candidate. For example, for CPA it is the descending order of the absolute value of Pearson correlation coefficients; for template attacks it is the descending order of the posterior probabilities.

2.2 Existing Metrics and Maximum Likelihood Principle

For evaluating side channel security on *subkey* recovery, the framework in [18] proposes the (t-th order) Success Rate (SR) and the Guessing Entropy (GE). The t-th order SR is defined as $\mathsf{SR}^{k_i}(t) = \Pr[k_i \in \{g_{i,[1]}, ..., g_{i,[t]}\}]$. It describes the probability that the correct subkey k_i is compromised in the first t prioritized guesses. The GE is defined as the expected rank of the correct subkey, i.e., $\mathrm{GE} := \sum_{t=1}^{2^n} t \cdot \Pr[k_i = g_{i,[t]}]$. Clearly, GE can be expressed as a function from the t-th order SR.

In addition, Pliam introduces *marginal guesswork* in [14] (also referred to as *work-factor* in [13]) as a metric to benchmark password recovery attacks, or more generically, *smart* exhaustive key searches. 'Smart' refers to adversaries that have and utilize prior information about the key distribution. Thus, marginal guesswork is well suited to describe adversaries that can assign probabilities to subkey candidates. In fact, it relates the success probability to its minimum computational complexity. More specifically, let $\sigma \in [0, 1]$ be the probability of success the adversary expects to achieve, the *σ-marginal guesswork* is defined to be the minimum number t of guesses to ensure finding the correct subkey k_i with at least σ success rate, i.e. $w_\sigma(k_i) = \min\{t : \sum_{j=1}^{t} p_{i,[j]} \ge \sigma\}$. Here subkey guesses $g_{i,[j]}$ are sorted decreasingly in terms of probabilities such that $p_{i,[j]} \ge p_{i,[j+1]} \ge \ldots$ where $p_{i,[j]} = \Pr[k_i = g_{i,[j]}]$.

This approach is also known as *Maximum Likelihood* (ML) attack. Based on side channel information, namely the inputs $\boldsymbol{x}^q$ and leakages $\boldsymbol{l}^q$, it first assigns posterior probability $p_{i,j} = \mathtt{Pr}[g_{i,j} \mid \boldsymbol{x}^q, \boldsymbol{l}^q]$ to subkey candidates $g_{i,j}$. Next, it enumerates them in a descending order $g_{i,[j]}$ according to the posterior likelihood $p_{i,[j]}$. Since $p_{i,j}$ is interpreted by definition as the likelihood of the true subkey k_i being the candidate $g_{i,j}$, the guess $g_{i,j}$ ensures subkey success rate $p_{i,j}$. Therefore the t-th order success rate using ML approach is

$$\mathsf{SR}^{k_i}(t) = \sum_{j=1}^{t} p_{i,[j]} \tag{1}$$

It also establishes a connection between the t-th order SR and the σ-marginal guess work as : $w_\sigma(k_i) = \min\{t : \mathsf{SR}^{k_i}(t) \ge \sigma\}$. To sum up, the adversary using

maximum likelihood approach is expected to have the minimum complexity to find the correct subkey.

2.3 Full Key Ranking Algorithm

The aforementioned metrics have mostly been applied for *subkey* recovery experiments. This changed with the algorithms by Veyrat-Charvillon et al. in [22,23]. The authors present algorithms to enumerate full keys [22] and to estimate the rank of the correct full key among all full key candidates [23]. With the latter algorithm they manage, for the first time, to approximate the computational complexity of successful side channel adversaries for cases where experimental verification is no longer possible or just too expensive. This means, the work pioneers in actually getting meaningful metrics for the *expected guesswork* of an adversary achieving *full key recovery*. Furthermore, they apply statistical bootstrapping to achieve cost evaluation and approximate a ML approach adversary for full key recovery.

The rank estimation algorithm [23], referred to as VGS algorithm, works as follows: As input it receives probabilities for all subkeys from a single side channel experiment, as well as the knowledge of the correct key (and consequently its probability). After sorting each of these subkey probabilities decreasingly, the different dimensions are combined to create the key space. Next, volumes where keys have higher (or lower) probabilities than the correct key are removed from the space and their size is added to the lower (or upper) bound for the rank of the correct key. The VGS algorithm stops either after a set time or once the bounds are close enough, i.e. once the key rank has been narrowed down sufficiently. Finally, it outputs (upper and lower bounds for) the key rank of the correct key.

By itself, the key rank only provides the placement of the probability of the correct key. It cannot specify, in each *individual* side channel experiment, how much probability of success one can achieve by guessing full key candidates up to the correct key. Instead, the probability of success is derived by statistical bootstrapping: the side channel experiment is repeated e.g. $n = 100$ times, and the success probability is derived as the percentiles of the key ranks in *different* experiments are turned into success probabilities. The VGS algorithm is used for comparison and as a benchmark for our algorithm that we introduce next.

3 Evaluating Full Key Security

Side channel leakage enables assigning scores or posterior probabilities to subkey candidates. However, to verify the correctness of a guess, different subkey parts must be combined and checked. That is to say, as long as the leakage is not strong enough to reveal each subkey part with a negligible error probability, the remaining full key security is not trivially evaluated and is worthy of investigation. Conceptually, the ML approach can be extended to cover full key recovery attacks so that all the metrics described in Section 2.2 can also be applied to

evaluate *full* key security. However, the size of the key space is 2^{bn}, e.g. in AES-128 it is 2^{128}, and it makes it infeasible to calculate the posterior probabilities to all full key candidates and then to enumerate them strictly following the ML principle. In this section, we introduce a weaker but computationally efficient approach to evaluate full key security. We call this approach the weak Maximum Likelihood (wML) approach. We describe its basic idea, followed by a Key Space Finding (KSF) algorithm as its realization and explain how it differs from a true ML approach.

3.1 Weak Maximum Likelihood Approach

Since computing and enumerating probabilities for all full key candidates is infeasible, the adversary can, nevertheless, adopt the following straightforward strategy. For each subkey part k_i, the adversary only considers the top e_i subkey candidates. When making full key guesses, she checks the Cartesian product of such selected candidates from all subkey parts. More specifically, the adversary considers the prioritized guesses $\{g_{i,[1]}, ..., g_{i,[e_i]}\}$ for the true subkey part k_i and verifies all possible combinations $\{g_{1,[j_1]} \| ... \| g_{b,[j_b]}$ where $1 \leq j_i \leq e_i, 1 \leq i \leq b\}$ as full key candidates. It is clear that this approach ensures a subkey success rate of $\mathsf{SR}^{k_i}(e_i)$ with e_i guesses for the subkey part k_i. Therefore, a full key success rate of $\prod_{i=1}^{b} \mathsf{SR}^{k_i}(e_i)$ is achieved, implying a full key verification cost of $\prod_{i=1}^{b} e_i$. The vector $\boldsymbol{e} = (e_1, ..., e_b)$ is called an *effort distributor* or simply a node. The node defines how the adversary distributes her verification complexity (or guesswork) over different subkey parts. It is easy to see from the definition above that an effort distributor not only determines the full key success rate $\mathtt{Prob}(\boldsymbol{e})$ that is achieved through guessing all candidates in the Cartesian product, but also determines the full key verification cost $\mathtt{Cost}(\boldsymbol{e})$, or *guesswork*. They are expressed as

$$\mathtt{Prob}\,(\boldsymbol{e}) = \prod_{i=1}^{b} \mathsf{SR}^{k_i}(e_i) = \prod_{i=1}^{b} \sum_{j=1}^{e_i} p_{i,[j]} \qquad \mathtt{Cost}\,(\boldsymbol{e}) = \prod_{i=1}^{b} e_i \tag{2}$$

In general, the adversary is interested in finding the minimal necessary guesswork to achieve a σ success rate for a full key recovery attack. The procedure of finding minimal full key recovery guesswork through finding optimal effort distributors is referred to as the *weak Maximum Likelihood* (wML) approach. Intuitively and informally, the observed leakage $\boldsymbol{l}^q$ reveals different amounts of secret information for different subkey parts. The more information is leaked of a certain key part, the more confidence the adversary gets for prioritized subkey guesses. Therefore, she can include more subkey candidates for the subkey positions where she has less confidence in the correctness of the output hypothesis (cf. e.g. [20]).

Formally, the wML approach can be stated as an optimization problem with the objective function and restriction condition defined as below.

$$\text{Objective: } \textbf{Minimize } \texttt{Cost}(\boldsymbol{e}) \tag{3}$$

$$\text{Restriction Condition: } \texttt{Prob}(\boldsymbol{e}) \geq \sigma \tag{4}$$

We will show how to solve this optimization problem in Section 3.3.

There are differences between the wML and the true ML approaches. In ML, all full key candidates are ordered according to their posterior probability. In wML, this is not necessarily the case. In fact, full key candidates that are inside the Cartesian product of selected subkey guesses are prior to combinations that are not defined by the effort distributor. For example, given an effort distributor $\boldsymbol{e} = (e_1, ..., e_b)$, the full key candidate $\boldsymbol{g}_x = g_{1,[e_1]} \| g_{2,[e_2]} \| g_{3,[e_3]} \| ... \| g_{b,[e_b]}$ is inside the Cartesian product, while the candidate $\boldsymbol{g}_y = g_{1,[e_1-1]} \| g_{2,[e_2+1]} \| g_{3,[e_3]} \| ... \| g_{b,[e_b]}$ is not. The former is to be considered by the wML approach while the latter is not. Therefore wML sets priority of the former over the latter. However, it is not always the case that $\boldsymbol{g}_x$ is more probable than $\boldsymbol{g}_y$. This means using wML will unavoidably cause some ordering violation. The impact of such violation is discussed in Section 4.3 and it turns out that the penalty is rather low, which confirms the usability of wML approach.

3.2 The Search Domain and its Calculus Model

An optimization problem in the continuous domain can usually be turned into a searching problem. Tools from differential calculus such as the gradient vector can help providing efficient search directions. Here we adjust it to our search space which is a discretized domain and build the model for the problem of searching optimal effort distributors. All concepts introduced here will be used in the KSF algorithm in Section 3.3. For a clear illustration we use AES-128 as an example. It can be easily applied in other block cipher scenarios.

Structure of the Search Domain We first define the search space. Each effort distributor $\boldsymbol{e}$ is treated as a node in the b-dimensional discrete space. For AES-128, the key has 16 subkey parts (bytes) and each effort entry—the number of guesses for each subkey part—can be any integer between 1 and 256 inclusively. Therefore, the entire search space is 16 dimensional with each dimension taking integers in $[1 : 256]$, namely $\mathcal{E} = [1 : 256]^{16}$. The optimization problem is now equivalent to finding the optimum node $\boldsymbol{e}^* \in \mathcal{E}$ that minimizes the full cost or guesswork while achieving the required full key success probability. To better understand the structure of the search space and enable an efficient search, we introduce the following concepts.

Definition 1: a node $\boldsymbol{e}' = (e'_1, ..., e'_b)$ is called the j-th *decremental neighbor* of the node $\boldsymbol{e} = (e_1, ..., e_b)$ if $e'_j = e_j - 1$ and $e'_i = e_i$ for all $i \neq j$. It is also denoted as $\boldsymbol{e}_j^- = (e_1, ..., e_j - 1, ..., e_b)$). Similarly, the j-th *incremental neighbor* of node $\boldsymbol{e}$ is denoted as $\boldsymbol{e}_j^+ = (e_1, ..., e_j + 1, ..., e_b)$.

Definition 2: a node $\boldsymbol{e} \in \mathcal{E}$ is said to be σ-*feasible* if it satisfies the restriction condition (4). The set of all σ-feasible nodes is denoted as $\mathcal{E}_\sigma := \{\boldsymbol{e} \mid \texttt{Prob}(\boldsymbol{e}) \geq \sigma\}$.

Definition 3: a σ-feasible node $\boldsymbol{e} \in \mathcal{E}_\sigma$ is said to be on the boundary if none of its decremental neighbors is σ-feasible, i.e. $\boldsymbol{e}_j^- \notin \mathcal{E}_\sigma, \forall j$. The set of all nodes on the boundary is called the σ-*feasible boundary* and denoted as

$$\partial(\mathcal{E}_\sigma) := \left\{\boldsymbol{e} \in \mathcal{E}_\sigma \mid \boldsymbol{e}_j^- \notin \mathcal{E}_\sigma, \forall j\right\}$$

Definition 4: a node $\boldsymbol{e}^*$ is called σ-*optimal* if it is a σ-feasible node and has minimal complexity among all σ-feasible nodes, i.e. $\texttt{Cost}(\boldsymbol{e}^*) \leq \texttt{Cost}(\boldsymbol{e}), \forall \boldsymbol{e} \in \mathcal{E}_\sigma$

An immediate but important result can now be summarized as follows.

Boundary Property: the σ-optimal nodes are inside the σ-feasible boundary, i.e. $\boldsymbol{e}^* \in \partial(\mathcal{E}_\sigma) \subset \mathcal{E}_\sigma$.

The proof is straightforward. If $\boldsymbol{e}_j^{*-} \in \mathcal{E}_\sigma$, then

$$\texttt{Cost}\left(\boldsymbol{e}_j^{*-}\right) = \texttt{Cost}\left(\boldsymbol{e}^*\right) \cdot \frac{e_j^* - 1}{e_j^*} < \texttt{Cost}\left(\boldsymbol{e}^*\right)$$

contradicting the definition of node $\boldsymbol{e}^*$ being σ-optimal.

This property explains the fact that if making one less subkey guess at any subkey part from an optimal effort distributor, the achieved success rate does not reach the desired level σ. It indicates that the wML approach is to find an σ-optimal effort distributor from the σ-feasible boundary.

A Calculus Model for the Search Problem Now we define some calculus tools for enabling an efficient search algorithm for finding the optimum node in the discrete search domain. For a function in continuous space, the partial derivative $\frac{\partial f}{\partial x_j}$ indicates the instantaneous change of the output of the function f caused by the change at the j-th coordinate x_j of the input. We define similar concepts for the objective function $\texttt{Cost}(\boldsymbol{e})$ and restriction condition $\texttt{Prob}(\boldsymbol{e})$.

The discrete nature of our search domain $[1:256]^{16}$ gives two situations: the change caused by unit *incrementing* or *decrementing* on each effort coordinate e_j. More specifically, we define the incremental partial derivative of $\texttt{Prob}(\boldsymbol{e})$ with respect to e_j as

$$\nabla P_j^+ = \texttt{Prob}(\boldsymbol{e}_j^+) - \texttt{Prob}(\boldsymbol{e}) = [\frac{\mathsf{SR}^{k_j}(e_j+1) - \mathsf{SR}^{k_j}(e_j)}{\mathsf{SR}^{k_j}(e_j)}]\texttt{Prob}(\boldsymbol{e}) \qquad (5)$$

Each ∇P_j^+ is a non-negative value[1] and it indicates the amount of additional success rate that could be achieved by incrementing effort by 1 at the j-th coordinate. Similarly, the decremental partial derivative of $Prob(\boldsymbol{e})$ is defined as

[1] The cases are considered separately if incrementing or decrementing is impossible, i.e. $e_j = 1$ or $e_j = 256$ for equations (5), (6) and (7).

$$\nabla P_j^- = \texttt{Prob}(\boldsymbol{e}) - \texttt{Prob}(\boldsymbol{e}_j^-) = [\frac{\mathsf{SR}^{k_j}(e_j) - \mathsf{SR}^{k_j}(e_j - 1)}{\mathsf{SR}^{k_j}(e_j - 1)}]\texttt{Prob}(\boldsymbol{e}) \qquad (6)$$

This is also a non-negative value and it tells the loss of full key success rate caused by decreasing effort by 1 at the j-th coordinate.

With the above defined partial derivatives, we can now obtain the *incremental gradient* $\nabla P^+ = (\nabla P_1^+, ...\nabla P_{16}^+)$ and the *decremental gradient* $\nabla P^- = (\nabla P_1^-, ...\nabla P_{16}^-)$ of the restriction condition $\texttt{Prob}(\boldsymbol{e})$. It is important to see that the coordinates for the largest partial derivatives in the incremental (or decremental respectively) gradient vector tells the full key success rate is increased (or decreased resp.) mostly due to a unit effort increment (or decrement resp.).

The same concept is defined for the objective function $\texttt{Cost}(\boldsymbol{e})$. The gradient vectors in both incrementing and decrementing cases result in the same expression because

$$\nabla C_j^+ = \texttt{Cost}(\boldsymbol{e}_j^+) - \texttt{Cost}(\boldsymbol{e}) = \prod_{i \neq j} e_i = \texttt{Cost}(\boldsymbol{e}) - \texttt{Cost}(\boldsymbol{e}_j^-) = \nabla C_j^- \qquad (7)$$

For notational convenience, both ∇C_j^+ and ∇C_j^- are replaced by ∇C_j and the gradient of the full key complexity $\texttt{Cost}(\boldsymbol{e})$ becomes $\nabla C = (\nabla C_1, ...\nabla C_{16})$. Again, each coordinate is a non-negative value and it indicates the change in full key recovery complexity which is caused by incrementing/decrementing effort by 1 at the j-th entry of effort node $\boldsymbol{e}$.

Lastly, we consider the *direction vector* $\boldsymbol{u}$ which is the negation of the gradient $-\nabla C$ projected onto the hyper-surface that is perpendicular to the gradient ∇P.

$$\boldsymbol{u} = -\nabla C \text{ projected onto } (\nabla P)^\perp = \frac{\nabla P \cdot \nabla C}{\|\nabla P\|^2} \nabla P - \nabla C \qquad (8)$$

where $\nabla P = (\nabla P_1, ..., \nabla P_{16})$ is the averaged gradient, i.e. $\nabla P_j = (\nabla P_j^+ + \nabla P_j^-)/2$. This direction vector $\boldsymbol{u}$ satisfies the intuition to keep the restriction condition $\texttt{Prob}(\boldsymbol{e})$ unchanged (seen from the vanishing of the inner product $\boldsymbol{u} \cdot \nabla P = 0$) while decreasing the objective function $\texttt{Cost}(\boldsymbol{e})$ as much as possible. A visualization can be seen in Figure 2.

3.3 An Optimized Key Space Finding Algorithm

We now show how to realize the weak maximum likelihood approach to find the optimum effort distributor by using the KSF algorithm.

The **inputs** of the algorithm include the desired full key success probability σ and the sorted posterior probabilities $p_{i,[j]}$ (and hence the subkey success rates $\mathsf{SR}^{k_i}(t)$ according to equation (1)) for all subkey candidates $g_{i,[j]}$. Note that this algorithm, unlike the VGS algorithm, does not require knowledge of the correct key, i.e. can also be used by a key recovering adversary. The applicability of

this algorithm is not restricted to the profiling adversary. In [22] it is suggested that a non-profiling adversary can also assign likelihoods to subkey candidates to achieve a justified full key ranking, which could also be applied in our case.

The algorithm returns two **outputs**: the minimum verification complexity $\min\{\texttt{Cost}(\boldsymbol{e}) \mid \boldsymbol{e} \in \mathcal{E}_\sigma\}$ that ensures the desired full key success rate σ together with an optimal effort distributor $\boldsymbol{e}^* = \operatorname{argmin}\{\texttt{Cost}(\boldsymbol{e}) \mid \boldsymbol{e} \in \mathcal{E}_\sigma\}$ that achieves this complexity lower bound.

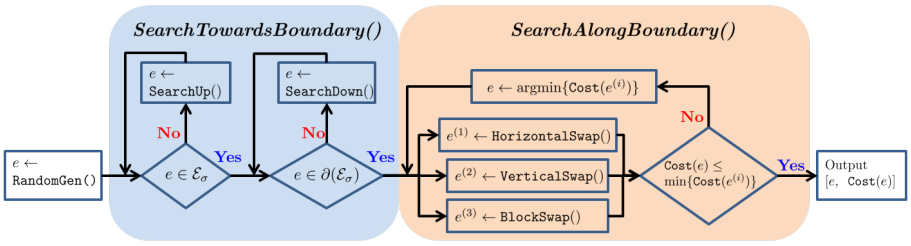


Fig. 1. Flow Chart of the KSF algorithm.

The flow chart of the KSF algorithm is shown in Fig. 1. It uses several subroutines. The algorithm begins by generating a random node $\boldsymbol{e} \leftarrow [1 : 256]^{16}$ using `RandomGen()`. This node serves as the starting point in the searching space. The initial node is then passed sequentially into two subroutines: `SearchTowardsBoundary()` and `SearchAlongBoundary()`. The former moves a node onto the feasible boundary $\partial(\mathcal{E}_\sigma)$ by calling `SearchUp()` and `SearchDown()`. The latter searches for nodes within the boundary that feature an even lower value of the objective $\texttt{Cost}(\boldsymbol{e})$. It uses the `Swap()` family of subfunctions. Note that the algorithm is a probabilistic algorithm to finding the point on the surface that has minimal cost. It finds local minima. In practice, it is executed several times to ensure that the local optimization also yields the global minimum.

The `SearchTowardsBoundary()` Function The task of this function is to move a node onto the feasible boundary $\partial(\mathcal{E}_\sigma)$. If the input node $\boldsymbol{e}$ does not satisfy the restriction condition, i.e. $\texttt{Prob}(\boldsymbol{e}) < \sigma$, it calls the function `SearchUp()` (as shown in Alg. 1) to search for a node that is σ-feasible. More specifically, `SearchUp()` iteratively increases the number of subkey guesses for some part of the subkey and updates the node. In each iteration, the search direction, i.e. the coordinate of the subkey part that needs to be incremented, is determined by the incremental gradient ∇P^+ as defined in Section 3.2. The effort coordinate that maximizes the gain in success rate through a unit effort increase is chosen, i.e. $i = \operatorname{argmax}_j\{\nabla P_j^+\}$. The node is updated by a unit increment on the chosen effort coordinate. The process continues until a σ-feasible node is reached, namely, the restriction condition is satisfied as $\texttt{Prob}(\boldsymbol{e}) \geq \sigma$.

Now we have a σ-feasible node—either it is an initially generated node that already satisfies the restriction condition or it is a node returned from `SearchUp()`.

Algorithm 1 SearchUp()

1: **while** $\texttt{Prob}(\boldsymbol{e}) < \sigma$ **do**
2: $\quad i \leftarrow \text{argmax}_j\{\nabla P_j^+\}$
3: $\quad e_i \leftarrow e_i + 1$
4: **end while**
5: **return** $\boldsymbol{e}$

Algorithm 2 SearchDown()

1: **while** $\boldsymbol{e} \notin \partial(\mathcal{E}_\sigma)$ **do**
2: $\quad i \leftarrow \text{argmax}_j\{\nabla C_j \text{ s.t. } \texttt{Prob}(\boldsymbol{e}) - \nabla P_j^- \geq \sigma\}$
3: $\quad e_i \leftarrow e_i - 1$
4: **end while**
5: **return** $\boldsymbol{e}$

The remaining task is to search for a node on the feasible boundary $\partial(\mathcal{E}_\sigma)$ since the optimal effort distributors can be found only on the boundary. The function `SearchDown()` is called to complete this task. In each iteration, the gradient vector ∇C of the objective function $\texttt{Cost}(\boldsymbol{e})$ is used to determine the search direction, i.e. the effort coordinate that needs to be decremented as shown in line 2 of Alg. 2. It reflects the direction where the objective function $\texttt{Cost}(\boldsymbol{e})$ has the biggest complexity drop through a unit effort decrementing while not violating the restriction condition. This means that the updated node is still σ-feasible. The process continues until the Boundary Property (as defined in Section 3.2) is satisfied. In other words, it returns a node $\boldsymbol{e} \in \partial(\mathcal{E}_\sigma)$.

The `SearchAlongBoundary()` Function So far the search algorithm has found a node on the σ-feasible boundary. The next step is to search for nodes within the boundary, which achieve σ-feasibility at a lower cost $\texttt{Cost}(\boldsymbol{e})$. The subroutine `SearchAlongBoundary()` is called to accomplish this task. We have seen from the Boundary Property in Section 3.2 that any decremental neighbor of a node on the boundary is not σ-feasible. It implies that the only way to find a node with lower full key cost is through trading-off (or swapping) efforts between different coordinates, which is realized in the `Swap()` family of subroutines.

More specifically, the coordinates for swapping are determined from the direction vector $\boldsymbol{u}$ defined in equation (8) as it follows the intuition that the search should decrease the overall guesswork while not compromising the full key success probability. The direction vector $\boldsymbol{u}$ suggests to increase effort on coordinate j if u_j is positive, and decrease if negative. The order of the effort coordinates being incremented or decremented is determined by the order of the absolute values of the entries u_j. The higher the absolute value, the higher the priority that is assigned to the coordinates for incrementing and decrementing.

Similar to search problems defined in continuous domain, the algorithm also handles the problem of local minima that prevent effective searching. In particular, we implement three different swapping modes –`HorizontalSwap`(), `VerticalSwap`() and `BlockSwap`() – to "escape" from many local minima and therefore mitigate the risk of being terminated in advance. The `HorizontalSwap()` allows trading-off multiple efforts between the positive most and negative most coordinates, i.e. u_i^+ and u_j^-. The `VerticalSwap()` in each iteration enables trading-off one effort between multiple coordinates where u_js are of different

Algorithm 3 SearchAlongBoundary()

1: $\boldsymbol{e}' \leftarrow$ `Swap`()
2: **while** `Cost`$(\boldsymbol{e}) >$ `Cost`$(\boldsymbol{e}')$ **do**
3: $\quad \boldsymbol{e} \leftarrow \boldsymbol{e}'$
4: **end while**
5: **return** $[e,$ `Cost`$(\boldsymbol{e})]$

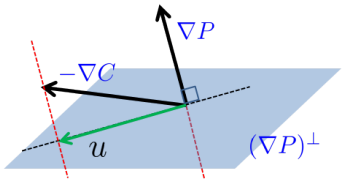


Fig. 2. Direction vector $\boldsymbol{u}$ is the projection of cost gradient $-\nabla C$ onto $(\nabla P)^{\perp}$

signs. Finally, the `BlockSwap()` mode enables trading-off multiple efforts on multiple coordinates. All three modes ensure that the swap does not compromise the required full key success probability, i.e. $\boldsymbol{e} \in \mathcal{E}_\sigma$ always hold. The updated node (after efforts being swapped) is again passed through `SearchDown()` to ensure that the search is still performed on the boundary. The three modes prevent infinite loops because the swap action occurs only if the cost of the updated node is lower than the cost for the session node.

As shown in Alg. 3, a temporary node $\boldsymbol{e}'$ is returned from the `Swap()` family of functions in each iteration. If the cost for the temporary node is lower than the current session node, then the session node $\boldsymbol{e}$ is replaced by before being passed into the next iteration. Otherwise the search is terminated and the algorithm outputs the current node $\boldsymbol{e}$ and its full key verification cost `Cost`$(\boldsymbol{e})$.

3.4 Usage of the KSF algorithm

Full key security evaluation used to stay as an analysis that is beyond computing power. The KSF algorithm provides practical meaning to the security evaluation. Firstly, the adversary can use it to determine if the leakage is strong enough to enable full key recovery at her accessible computing power. More specifically, upon a particular set of observations $(\boldsymbol{x}^q, \boldsymbol{l}^q)$, the returned global minimum of `Cost`$(\boldsymbol{e})$ serves as an *individual* lower bound of the optimum guesswork w_σ. If the guesswork is acceptable, the associated optimal effort distributor $\boldsymbol{e}$ provides a winning strategy: checking all the full key candidates defined by the Cartesian product of this optimal node. This strategy ensures he adversary with success rate being at least σ. Even if in one session the observed leakages are not strong enough, namely requires high w_σ, she can just wait for the next session until a "good" observation appeared. This can be the case if the guesswork is impacted a lot from different observations, which is in fact verified in our experiments in the next section.

Secondly, it can be used by a security evaluation lab. By feeding the algorithm with independently generated observations $(\boldsymbol{x}^q, \boldsymbol{l}^q)$, an evaluator can bootstrap the individual lower bounds and obtain the distribution of the guesswork w_σ at any fixed σ. This informs the evaluator the resistance of some DUT against a

probabilistic SCA. In other words, if the adversary intends σ success rate, how much chance does she have by waiting until a strong enough leakage occurs. A simple example would be computing the *expected* lower bound of guesswork—the average of all individual lower bounds—and using it as a metric. The metric indicates the averaged level of security of the full key as the expectation is with respect to various experiments, i.e. not only different choices of input $\boldsymbol{x}^q$, but also leakages observations $\boldsymbol{l}^q$.

4 Experiment Results and Comparison

In this section we apply the proposed wML approach to practical side channel leakage evaluation. We first explain the experimental setup. Next, we verify the validity of the KSF algorithm and discuss its possible influencing factors. Finally, we compare our approach and VGS algorithm. .

4.1 Experiment Setup and Posterior Probabilities Derivation

We conduct the leakage evaluation experiments in two settings: real measurements and simulations. For the former, we target on an unprotected AES software implementation, the RjindaelFurious [15] running on an 8-bit AVR ATXMega A3B processor. A total of 200,000 measurements were taken using a Tektronix DPO 5104 oscilloscope at a sampling rate of 200MS/s. Among all the collected traces, 20,000 are used for building Gaussian templates. The remaining traces are used as needed for the evaluation step. In the other setting, we simulate side channel leakage using the widely accepted Hamming weight leakage model with additive Gaussian noise. In both cases the targeted leakage is that of the s-box output of the first round for each of the 16 state bytes.

As a preparation step of leakage evaluation, posterior probabilities for all subkey candidates need to be estimated from side channel observations. The probably most popular method is through Templates [2,10] where the adversary creates a precise model of the leakage in the profiling phase and derives posterior probabilities in the attack phase. An in-depth discussion of modeling errors for Gaussian templates can be found in [3]. For our experiments, we build Gaussian templates $\mathcal{N}(L; \boldsymbol{\mu}_v, \Sigma_v^2)$ regarding the internal state $Y = S(X \oplus K)$ over all the 16 bytes. In the attack phase, the adversary obtains the observations $(\boldsymbol{x}^q, \boldsymbol{l}^q)$. Since the predicted internal state for the j-th query is $y_{i,j,g} = f(x_{i,j}, g)$ under the subkey hypothesis g at the i-th subkey part, the observed leakage $l_{i,j}$ has conditional probability density $\mathrm{P}[l_{i,j} \mid g] = \mathcal{N}(l_{i,j}; \boldsymbol{\mu}_v, \Sigma_v^2)$, where $v = y_{i,j,g}$. The Bayesian formula returns posterior probabilities given single observation $l_{i,j}$ as

$$\Pr[g \mid l_{i,j}] = \frac{\mathrm{P}[l_{i,j} \mid g] \cdot \Pr[g]}{\sum_{g^*} \mathrm{P}[l_{i,j} \mid g^*] \cdot \Pr[g^*]} = \frac{\mathrm{P}[l_{i,j} \mid g]}{\sum_{g^*} \mathrm{P}[l_{i,j} \mid g^*]}$$

Further, the adversary can obtain the posterior probability $p_{i,g}$ upon observing all the q-leakages , which is expressed as

$$p_{i,g} := \Pr[g \mid l_{i,1}, ..., l_{i,q}] = \frac{\prod_{j=1}^{q} \Pr[g \mid l_{i,j}]}{\sum_{g*} \prod_{j=1}^{q} \Pr[g^* \mid l_{i,j}]} \tag{9}$$

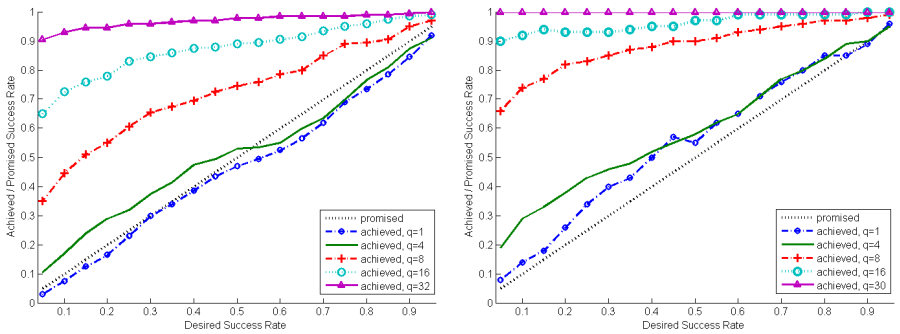
Finally the posterior probabilities $p_{i,g}$ are sorted into a descending sequence $p_{i,[g]}$ as detailed in Section 2.2. They determine the subkey success rates in equation (1) which are the inputs for the KSF algorithm and the VGS algorithm.

4.2 Correctness and Influencing Factors of the KSF Algorithm

Verifying the correctness of the KSF algorithm is rather simple: if the returned optimal effort distributor $\boldsymbol{e}^*$ covers the ranks of the posterior probability of every subkey k_i, then the search space defined by the Cartesian product includes the correct full key as explained in Section 3.1. In the following, we check if the algorithm in fact achieves the promised success rate for various experiments. We provide a set of observations for a range of q from 1 to 40: higher value for q indicates more leaked information. We furthermore set 19 different levels of desired success rate from 0.05 to 0.95 incrementing at 0.05. For each possible (q, σ), 200 experiments are performed for the scenario using real measurements, and 100 experiments for the scenario using simulated leakage.

Figure 3(a) compares the promised full key success rate of the KSF algorithm with the actually achieved success rate for real measurements. One can see that when the leakage is strong (high value of q), the achieved success rate is far beyond what is promised. However, when the leakage is weak, the two rates only differ slightly. A probable reason for the achieved success rate being *lower* than the desired success rate for small values of q is due to the assumption that the Gaussian templates fully capture the underlying leakage distribution. In fact, the empirically obtained Gaussian templates only serve as approximation to the true leakage distribution, and hence the derived posterior probabilities are unavoidably biased. This claim is also supported by the results for simulated leakage, as given in Figure 3(b), where the underachieving never happens. Nevertheless, for almost all cases, especially when $q \geq 8$, the KSF algorithm fulfills the promised full key success rate.

Other influencing factors of the KSF algorithm are the leakage observations and the number of independent initial nodes used for finding local minima, as discussed in Section 3. To investigate their impact, we run 50 experiments associated with independent sets of observations $(\boldsymbol{x}^q, \boldsymbol{l}^q)$. In each experiment, we compare the performance of KSF algorithm at fixed $\sigma = 50\%$ using 100 and 10000 initial nodes. The global minimum guessworks in each experiments are returned and compared in Figure 4(a). The x-axis is the index of experiments indicating a different set of observation $(\boldsymbol{x}^q, \boldsymbol{l}^q)$ and the y-axis is the guesswork in bits. As we can see, different leakage observations causes more than 40 bits guesswork differences while the influence from the number of initial nodes (the distance between the two curves) are rather small. In fact, the biggest difference



(a) (b)

Fig. 3. Correctness verification for real measurements (a) and simulation (b); The success rate that KSF achieves (y-axis) is more than what it promised (x-axis).

between the two curves is less than 2.5 bits and most of the times the difference is smaller than one bit.

4.3 Comparing the KSF algorithm with the VGS algorithm

As mentioned in Section 2.3, the VGS algorithm estimates the rank of the correct key among all full key candidates. By bootstrapping this rank statistic, or namely, by repeating the rank estimation from different side channel observations, one can get a security evaluation based on the success percentiles to see the rank distributions given random side channel inputs.

We first provide several comparisons between the bootstrapping of the `rank` statistic from repeating VGS algorithm and the bootstrapping of guesswork w_σ KSF algorithm. Figure 4(b) compares the two over the real measurement. We fix the full key success rate in KSF algorithm to $\sigma = 50\%$. For each q (x-axis), we perform 200 experiments using the algorithms on the same sets of observations. The box plot indicates quartiles and outliers of the guesswork and rank statistics. We see that the results from the two algorithms are relatively close to each other. Further, the impact of different leakages on the rank statistic using VGS algorithm is heavier than that on the guesswork returned from our algorithm. This can be seen from the difference of the height of boxes for the two algorithms. More importantly, we see that the medians of the two analyzed cases do not align exactly. In fact, ours are always slightly higher than the VGS algorithm. The reason is two folds. On one side, the KSF algorithm is following wML approach, which introduces ordering violation comparing to the true ML approach, as explained in Section 3. On another, since in each individual experiment the VGS algorithm does not return a fixed success probability $\sum_{t=1}^{\texttt{rank}} p_{[t]}$ (the ML adversary should guesses all the top `rank` full key candidates), the 50th percentile of the `rank` does not necessarily ensure the adversary achieves 50% success rate in an averaged experiment either. This is even more clearly seen from the simulated

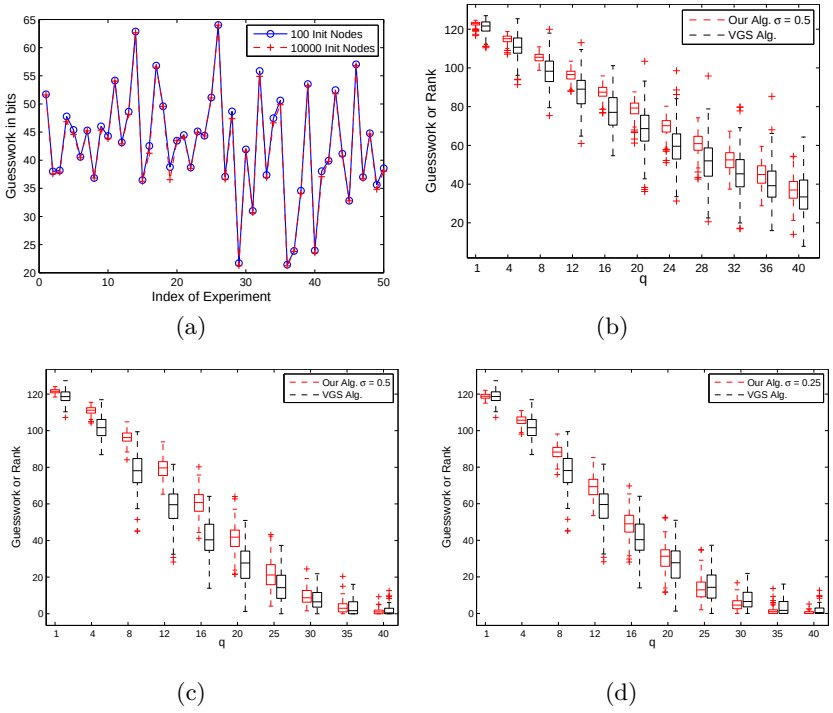
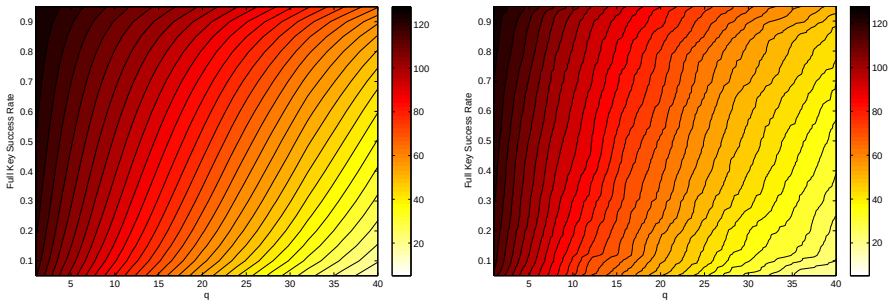


Fig. 4. Figure (a) shows the impact on guesswork (y-axis) from the number of starting nodes for KSF algorithm is far less than the impact from the set of observations $(\boldsymbol{x}^q, \boldsymbol{l}^q)$ in each experiment (X-axis); Figure (b,c,d) compares the size of the key space from the KSF algorithm to the key rank from the VGS algorithmĖExperiments are performed over real measurement with success rate $\sigma = 50\%$ (b); over simulation with $\sigma = 50\%$ (c); and over simulation with $\sigma = 25\%$ (d)

leakage scenario as shown in Figure 4(c) (`rank` compared to $w_{0.5}$) and 4(d) (`rank` compared to $w_{0.25}$). In the simulated case, the ML approach is closer to the $w_{0.25}$ bootstrapping with the weak ML approach. It indicates that the guessing the top `rank` most likely full key candidates in the ML approach roughly returns winning probability of 25%. In general, it might suggest the evaluator to find the appropriate σ level such that the bootstrapping of the guesswork w_σ matches the bootstrapping of the key `rank`. By doing so, the evaluator can estimate the success rate $\sum_{t=1}^{\texttt{rank}} p_{[t]}$ in an average experiment that the top `rank` full key candidates contain.

The next comparison of the two leakage evaluation algorithms is between the *expected* guesswork lower bound (Figure 5(a)) and the bootstrapping of the `rank` (Figure 5(b)). Experiments use the data from the microcontroller measurements. The x-axis for both represents the number q of accessible leakages in each experiment. In Figure 5(a), the y-axis is the desired full key success probability



(a) (b)

Fig. 5. Security evaluation using KSF algorithm showing the remaining guesswork (color in (a)) and using VGS algorithm showing the key rank (color in (b)) over the number of observations q (x-axis) and success rate/percentile (y-axis).

σ. The color or gray-scale for the pixel at coordinate $(x, y) = (q, \sigma)$ represents the *expected* lower bound (as explained in Section 3.4) of the guesswork in log scale. The darker a pixel is, the more guesswork is needed to achieve the specified success rate σ. In particular, the expected lower bound at each (q, σ) is derived from 200 independent experiments. Each experiment uses an independent set of observations $(\boldsymbol{x}^q, \boldsymbol{l}^q)$ which yields different posterior probabilities $p_{i,[g]}$ computed as described in Section 4.1. The number of initial node is set to 100 (Figure 4(a) already shows this number is sufficient). The global minimum guesswork from the 100 searches is returned as the *individual* lower bound of the guesswork for this single experiment. Upon the completion of the 200 experiments, the average of the 200 individual lower bounds yields the *expected* lower bound as reflected in the color of pixel in Figure 5(a). In short, the color at pixel (q, σ) indicates the expected minimum guesswork that a q-limited adversary should spend in order to achieve full key recovery with probability σ. In Figure 5(b), VGS algorithm is executed with the same sets of observations $(\boldsymbol{x}^q, \boldsymbol{l}^q)$. The returned 200 `ranks` (represented in the color of each pixel) derive the statistical bootstrapping of the success percentile (the same as in bootstrapping) which is represented on the y-axis. Two contour plots are fairly close to each other.

5 Conclusion

The presented algorithm finds the optimal key search space that allows the adversary to achieve a predefined probability of success. Unlike prior work, the algorithm provides a connection between remaining full key security and success probability even for a single set of side channel observations. It furthermore is a constructive algorithm, since it not only bounds the remaining key search space, but also provides an optimized yet simple strategy to search that space. As a consequence, the algorithm can be used by embedded security evaluators to

quantify the resistance of a device to SCA. It can also be used by an adversary to determine whether the leakage suffices for a successful key recovery attack.

Acknowledgments

This material is based upon work supported by the National Science Foundation under Grant No. #1261399 and Grant No. #1314770. We would like to thank François-Xavier Standaert for the helpful discussion. We would also like to thank the anonymous reviewers for their helpful comments.

References

1. Dpa contest (versions 1 and 2). `http://www.dpacontest.org/home/`. 1
2. S. Chari, J. Rao, and P. Rohatgi. Template Attacks. In *Cryptographic Hardware and Embedded Systems - CHES 2002*. Springer Berlin / Heidelberg, 2003. 12
3. F. Durvaux, F.-X. Standaert, and N. Veyrat-Charvillon. How to Certify the Leakage of a Chip? In *to appear in the proceedings of Eurocrypt 2014*. Springer LNCS, 2014. 12
4. S. Faust, K. Pietrzak, and J. Schipper. Practical Leakage-Resilient Symmetric Cryptography. In E. Prouff and P. Schaumont, editors, *Cryptographic Hardware and Embedded Systems Ű CHES 2012*, volume 7428 of *Lecture Notes in Computer Science*, pages 213–232. Springer Berlin Heidelberg, 2012. 1
5. B. Gierlichs, K. Lemke-Rust, and C. Paar. Templates vs. Stochastic Methods. In L. Goubin and M. Matsui, editors, *Cryptographic Hardware and Embedded Systems - CHES 2006*, volume 4249 of *Lecture Notes in Computer Science*, pages 15–29. Springer Berlin Heidelberg, 2006. 2
6. P. C. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In N. I. Koblitz, editor, *Advances in Cryptology — CRYPTO '96*, volume 1109 of *Lecture Notes in Computer Science*, pages 104–113. Springer Verlag, 1996. 1
7. P. C. Kocher. Leak-resistant cryptographic indexed key update (US patent 6539092), 2003. 1
8. P. C. Kocher, J. Jaffe, and B. Jun. Differential Power Analysis. In *CRYPTO '99: Proceedings of the 19th Annual International Cryptology Conference on Advances in Cryptology*, pages 388–397, London, UK, 1999. Springer-Verlag. 1
9. F. Macé, F.-X. Standaert, and J.-J. Quisquater. Information Theoretic Evaluation of Side-Channel Resistant Logic Styles. In P. Paillier and I. Verbauwhede, editors, *Cryptographic Hardware and Embedded Systems - CHES 2007*, Lecture Notes in Computer Science, pages 427–442. Springer Berlin Heidelberg, 2007. 2
10. S. Mangard, E. Oswald, and T. Popp. *Power Analysis Attacks: Revealing the Secrets of Smartcards*. Springer-Verlag, 2007. 2, 12
11. M. Medwed, F.-X. Standaert, and A. Joux. Towards Super-Exponential Side-Channel Security with Efficient Leakage-Resilient PRFs. In E. Prouff and P. Schaumont, editors, *Cryptographic Hardware and Embedded Systems — CHES 2012*. Springer Berlin Heidelberg, 2012. 1
12. A. Moradi, M. Kasper, and C. Paar. Black-Box Side-Channel Attacks Highlight the Importance of Countermeasures. In O. Dunkelman, editor, *Topics in Cryptology – CT-RSA 2012*, volume 7178 of *Lecture Notes in Computer Science*, pages 1–18. Springer Berlin Heidelberg, 2012. 1

13. J. Pliam. The Disparity between Work and Entropy in Cryptology. Cryptology ePrint Archive, Report 1998/024, 1998. `http://eprint.iacr.org/`. 3
14. J. Pliam. On the Incomparability of Entropy and Marginal Guesswork in Brute-Force Attacks. In B. Roy and E. Okamoto, editors, *Progress in Cryptology — INDOCRYPT 2000*, volume 1977 of *Lecture Notes in Computer Science*, pages 67–79. Springer Berlin Heidelberg, 2000. 3
15. B. Poettering. Rijndael Furious. Implementation. `http://point-at-infinity.org/avraes/`. 12
16. F. Regazzoni, S. Badel, T. Eisenbarth, J. Großschädl, A. Poschmann, Z. T. Deniz, M. Macchetti, L. Pozzi, C. Paar, Y. Leblebici, and P. Ienne. A Simulation-Based Methodology for Evaluating the DPA-Resistance of Cryptographic Functional Units with Application to CMOS and MCML Technologies. In *International Symposium on Systems, Architectures, Modeling and Simulation (SAMOS VII)*, 2007. 2
17. M. Rivain. On the Exact Success Rate of Side Channel Analysis in the Gaussian Model. In R. Avanzi, L. Keliher, and F. Sica, editors, *Selected Areas in Cryptography*, volume 5381 of *Lecture Notes in Computer Science*, pages 165–183. Springer Berlin Heidelberg, 2009. 2
18. F.-X. Standaert, T. G. Malkin, and M. Yung. A unified framework for the analysis of side-channel key recovery attacks. *Advances in Cryptology — EUROCRYPT 2009*, pages 443–461, 2009. 2, 3
19. F.-X. Standaert, O. Pereira, Y. Yu, J.-J. Quisquater, M. Yung, and E. Oswald. Leakage Resilient Cryptography in Practice. In A.-R. Sadeghi and D. Naccache, editors, *Towards Hardware-Intrinsic Security*, Information Security and Cryptography, pages 99–134. Springer Berlin Heidelberg, 2010. 1
20. A. Thillard, E. Prouff, and T. Roche. Success through Confidence: Evaluating the Effectiveness of a Side-Channel Attack. In *Cryptographic Hardware and Embedded Systems - CHES 2013*, volume 8086 of *Lecture Notes in Computer Science*, pages 21–36. Springer Berlin Heidelberg, 2013. 5
21. K. Tiri, M. Akmal, and I. Verbauwhede. A dynamic and differential CMOS logic with signal independent power consumption to withstand differential power analysis on smart cards. In *Solid-State Circuits Conference, 2002. ESSCIRC 2002. Proceedings of the 28th European*, pages 403 –406, sept. 2002. 2
22. N. Veyrat-Charvillon, B. Gérard, M. Renauld, and F.-X. Standaert. An Optimal Key Enumeration Algorithm and Its Application to Side-Channel Attacks. In *Selected Areas in Cryptography*, volume 7707 of *Lecture Notes in Computer Science*, pages 390–406. Springer Berlin Heidelberg, 2013. 2, 4, 9
23. N. Veyrat-Charvillon, B. Gérard, and F.-X. Standaert. Security Evaluations beyond Computing Power. In T. Johansson and P. Q. Nguyen, editors, *Advances in Cryptology — EUROCRYPT 2013*, volume 7881 of *Lecture Notes in Computer Science*, pages 126–141. Springer Berlin Heidelberg, 2013. 2, 4