On the Security of Partially Masked Software Implementations

Alessandro Barenghi and Gerardo Pelosi
Department of Electronics, Information and Bioengineering – DEIB
Politecnico di Milano, Via G. Ponzio 34/5, I-20133 Milano, Italy

Keywords: Applied Cryptography, Side-channel Attacks.

Abstract: Providing sound countermeasures against passive side channel attacks has received large interest in open literature. The scheme proposed in (Ishai et al., 2003) secures a computation against a d-probing adversary splitting it into $d+1$ shares, albeit with a significant performance overhead ($5\times$ to $20\times$). We maintain that it is possible to apply such countermeasures only to a portion of the cipher implementation, retaining the same computational security, backing a widespread intuition present among practitioners. We provide the sketch of a computationally bound attacker model, adapted as an extension of the one in (Ishai et al., 2003), and detail the resistance metric employed to estimate the computational effort of such an attacker, under sensible assumptions on the characteristic of the device leakage (which is, to the current state of the art, still lacking a complete formalization).

1 INTRODUCTION

In a world where embedded computing devices become more and more pervasive, cyber-physical systems are increasingly employed to process and store both sensitive and security-critical data. The prime mean to provide proper security and privacy warranties is represented by cryptographic primitives, which are being more and more often embedded in digital devices as either hardware accelerators or software libraries. In such a scenario an attacker can have physical access to the target device and may exploit either cipher implementation weaknesses or side channel information (f.i., power consumption, execution timing or electro-magnetic emanations) to infer the value of secret parameters intended to be stored in an un-accessible way. Tackling these attacks requires a combined effort in order to choose cryptographic primitives with sound warranties from the theoretical standpoint, as well as to consider carefully their implementation so that the large class of the so-called *implementation attacks* are warded off. The choice of sound primitives can be effectively performed among the well scrutinized ones, which have been recognized as standards by international and national entities such as the ISO/IEC committee or the US national institute of standards and technology (NIST). By contrast, warding off implementation attacks is still a complex issue. Indeed, they have been a significant threat in recent times, allowing the breach of many systems ranging from e-tickets (Garcia et al., 2009) to IP-protection on large scale reconfigurable devices (Moradi et al., 2011).

The largest class of implementation attacks is represented by the so-called *side-channel attacks*, where the attacker exploits the information leakage happening on an unintended channel, i.e., not on means of communicating with the target device intended by the designer (such as input/output ports). More concretely, the attacker exploits the fact that some environmental parameters of a regular functioning of a digital device are dependent on the input data being processed by it. In particular, the energy required to perform the computation, the Electro-Magnetic (EM) emanations of the device or the time taken to complete the execution have been reported to be effective side-channels and are exploited in (Mangard et al., 2007; Barenghi et al., 2013). Designing efficient and effective countermeasures against side-channel attacks is a topic which has received warm attention by the research community (Agosta et al., 2012; Agosta et al., 2013b; Agosta et al., 2014). Typically the countermeasures involve either modifying the cipher to the algorithmic or implementation level, or changing the underlying hardware architecture so to suppress the side-channel leakage. Despite the efforts to design such countermeasures have been significant, only very few schemes have been proven secure against

Barenghi A. and Pelosi G..
On the Security of Partially Masked Software Implementations.
DOI: 10.5220/0005120504920499
In *Proceedings of the 11th International Conference on Security and Cryptography* (SECRYPT-2014), pages 492-499
ISBN: 978-989-758-045-1

a precise attacker model (Ishai et al., 2003; Coron, 2014), and it hasn't been infrequent for non-proven countermeasures to be broken after their proposal. On the other hand, provably secure countermeasure solutions have very significant overheads in terms of performance: losses in the range of $5\times$ to $10\times$ are not infrequent, thus limiting the range of applicability of such solutions (Agosta et al., 2013a). Such an overhead comes from the inherently redundant computation mandated by the countermeasures being performed on the whole cipher execution.

In this work, we will show which part of a cipher implementation can be left unprotected by the countermeasures, preserving the computational security margin of the implementation. We will deal with symmetric block ciphers, which (by their own nature) provide a computational security margin, as their key size is finite. We will analyze the possibility of applying the countermeasures to the parts of the cipher where the computational complexity of leading a side-channel attack is higher or the same of performing an exhaustive search over the whole keyspace. In particular, we will provide a practical sketch of the attacker model against software implementation of block ciphers protected through adapting the scheme in (Ishai et al., 2003) and we will extended the definition of *instruction resistance* introduced in (Agosta et al., 2013a) detailing its properties in the context of power-based side-channel leakages of block cipher software implementations.

The paper is organized as follows: in Section 2 we recap the workflow of side-channel attacks, the main strategies put into effect to counter them and describe the perfectly secure countermeasure proposed in (Ishai et al., 2003), in Section 3 we illustrate the data flow analysis employed to precisely pinpoint the vulnerable instruction of a software implementation and point out the properties of the bit level resistance metric. Subsequently, in Section 4 we will describe the side-channel attacker model for software implementations and validate our main statement. Section 5 will draw our conclusions.

2 PRELIMINARIES

The classic workflow for power-based SCAs aims at recovering the value of the secret parameter of a cipher (i.e., the secret key) one portion at a time. This is possible since, during a cryptographic computation, the algorithm combines the secret key bits with the intermediate values involving a limited quantity of them at a time. An analogous strategy can be applied employing EM radiations of the device as the side channel leaking information. The first step of the attack consists in measuring the power consumption of the target device with different input messages for a large number of computations. Subsequently, an intermediate operation employing a small portion of the secret key is selected, and its results are guessed for all the possible values of the key portion. From these hypotheses on the results, a series of predictions of the power consumption are made, one for each possible value of the secret key portion. Finally, the predicted consumption values are compared with the actual measured ones through the use of statistical means (f.i., linear correlation index or difference-of-means test) to find out which prediction fits best.

Power-based SCAs affect both hardware and software implementations of cryptographic primitives. Many techniques have been designed to counter this attack either at logic style- or architectural-level, since energy consumption variations are strictly related to both of them (Mangard et al., 2007). An example of low-level hardware countermeasure consists in employing a decoupling capacitor placed as close as possible to the supplied voltage pins of the target device, in such a way to significantly hinder the collection of informative measurements through flattening voltage variations. However, this does not provide protection against EM-based attacks. Indeed, as shown in (O'Flynn and Chen, 2012), it is possible to wrap the capacitor in a thin magnetic wire in such a way to exploit the measurement of the high-frequency components flowing through it. Usually, countering many different attack vectors through hardware solutions, results in an unfavorable trade-off between cost, performance and security guarantees. The importance of protected software implementations comes into play as a way to obtain more convenient tradeoffs among these figures of merit. Indeed, more and more often software cryptographic libraries are deployed either as an alternative solution to hardware ones, or as a fallback should they be breached.

Masking and Hiding. Countermeasures to protect implementations of ciphers against power-based SCA aim at concealing the relation between the power consumption and the operations performed by the target device to compute sensitive intermediate values: they are split into two categories: *hiding* and *masking*.

For software implementations, the *hiding* strategies hinder the matching between the actual power measurements and the consumption modeled for each key-portion guess through rescheduling some instructions, permuting the sequence of accesses to lookup tables, or inserting random delays built out of dummy operations (Mangard et al., 2007; Tillich and Herbst, 2008; Coron and Kizhvatov, 2010). In hardware im-

Table 1: Complexity of bitwise masked operations as a function of the masking order d and lookup table size l.

Op.s	Complexity	Ref.
xor	$3(d+1)$ xor	(Rivain and Prouff, 2010) (Ishai et al., 2003)
not	1 not	(Ishai et al., 2003)
and	$2d(d+1)$ xor $+(d+1)^2$ and	(Ishai et al., 2003)
or	$2d(d+1)$ xor + $+(d+1)^2$ and $+3$ not	$a \vee b = \neg((\neg a) \wedge (\neg b))$
table lookup	$2ld$ xor $+ld$ store + $+(ld+1)$ load	(Schramm and Paar, 2006)

plementations the common *hiding* strategies involve feeding the chip with a drifting clock so to change the instant in time when the sensitive operations are performed and adding extra hardware with the intent to reduce the Signal to Noise Ratio (SNR) of the physical measurements (Mangard et al., 2007).

Masking schemes (Ishai et al., 2003; Mangard et al., 2007) invalidate the correlation between the values employed to predict the power consumption and the actual values processed by the device. The principle is to add one or more random values (*masks*) to every sensitive intermediate variable occurring during the computation. In a masked implementation, each sensitive intermediate value is represented as split in a number of *shares* (containing both the randomized sensitive value and the masks employed), which are then separately processed. To this end, the target algorithm is modified to process each share and recombine them at the end of the computation. This technique effectively hinders the attacker from formulating a correct power consumption model, as the instantaneous power consumption is independent from the original (non-masked) value. Typically, masking techniques are categorized by the number of masks, d, employed for each sensitive value, which is known as the *order* of the masking. A d-th-order masking can always be theoretically broken by a $(d+1)$-th-order attack, i.e., an attack exploiting the combination of $d+1$ measurements in different time instants, during an execution, to build a mask-independent power consumption model (Mangard et al., 2007; Schramm and Paar, 2006; Rivain and Prouff, 2010).

Hiding provides an increase in the computational security margin, as more samples must be collected to recover the secret key, resulting in both higher storage and computation requirements. By contrast, *masking* techniques are able to provide perfect security (i.e., security against a computationally unbounded attacker), provided that the number of measurements performed during the computation of an intermediate value is lower than the order of the masking.

Provably Secure Countermeasures. A theoretical framework for assessing the security of a masking-based countermeasures is provided by Ishai et al. (Ishai et al., 2003). Each sensitive operation is modeled as a Boolean circuit and a perfectly secure masking scheme (usually referred to as ISW), with order d, is defined in terms of a transformation operating on the circuit, outputting a protected version of it, which is functionally equivalent to the unprotected one. The protected circuit employs both standard logic gates and a "randomness" gate, which outputs one fresh randomly chosen bit per clock cycle.

The threat model assumes an adversary able to acquire at most d simultaneous bit-level values during per clock cycle of the computation (Rivain and Prouff, 2010; Ishai et al., 2003). The scheme is proven to provide the indistinguishability of the d values obtained by the attacker from d randomly extracted values, thus providing perfect security of the computation against probing. From a constructive point of view, the unprotected computation is substituted by three phases: *1)* an initial share-splitting, where every original bit-value is split up into $d+1$ randomized values over different wires, *2)* a transformation of the original computation into one processing all the $d+1$ shares, and *3)* a final recombination, which must yield the same result as the unprotected computation provided that the composition of the masked values is properly handled, as detailed in (Prouff and Rivain, 2013). Assuming an unprotected circuit with depth h and size of $O(n)$ gates, the transformed circuit exhibits a depth of $O(h \log d)$ and a size of $O(n\, d^2)$ gates.

Table 1 shows the computational costs to mask bitwise operations as a function of the scheme order d. For multi-bit arithmetic operations it is possible to perform conversions between Boolean masked values and arithmetic masked ones and viceversa (Debraize, 2012). In case the Boolean function is available in the form of a lookup table, the masking of the looked-up values is safe up to the 2nd order according to (Coron et al., 2007). The key idea is that, whenever two share-split operands are combined together, fresh random values should be inserted in the computation of the resulting output shares. As the masking countermeasure is particularly computationally demanding (see Table 1) applying it as sparingly as possible, without lowering the security margin of the cipher, is highly desirable.

3 SIDE CHANNEL RESISTANCE

To the end of better understanding the actual computational effort required to perform a passive side-channel attack, it is necessary to understand which intermediate values of a software computation are el-

igible as targets of the attack. In particular, the computational effort needed to exploit the leakage of a particular intermediate value depends (see Section 2) on the number of hypotheses (about a secret-key portion) that an attacker need to formulate to correlate its predictions with the actual measurements. Therefore, understanding which and how many bits of the cipher-key value contribute to the output value of every bit of each intermediate instruction of the cipher, is crucial to assess the SCA vulnerability of the cipher.

A useful instrument to be employed is the DataFlow Analysis (DFA), which is commonly used by compilers to manipulate the data dependencies among the variables involved in the computation of an implementation. The effectiveness of employing dataflow analysis techniques to analyze the intermediate values of a cipher computation has been validated in (Agosta et al., 2013a), through characterizing a software AES implementation. In particular, the aforementioned analysis provides a conservative margin on the resistance against passive side channels of an intermediate value (i.e.at most the analysis underestimates its resistance). Dataflow analysis represents a program in terms of its Control Flow Graph, which is defined as follows.

Definition 3.1 (Control-Flow Graph)**.** *A Control Flow Graph (CFG) is a directed graph $\mathcal{G}(\mathcal{B}, \mathcal{E})$ where each node $i \in \mathcal{B}$ represents a statement of the program ($\texttt{stat}_i$). The graph is augmented with two additional nodes i_{in}, i_{out}. An edge $(i, i') \in \mathcal{E}$ is added if the statement $\texttt{stat}_{i'}$ is executed immediately after the statement $\texttt{stat}_i$, and each node has at most two immediate successors. For the first statement ($\texttt{stat}_0$) there is an edge (i_{in}, i_0), while an edge $(j, i_{\text{out}}))$ is added for each node j bound to a statement ($\texttt{stat}_j$) preceding an exit point of the program.*

It is common practice when performing dataflow analyses, to translate the program in a normal form, where every intermediate variable is defined (i.e., generated) in a single point, and only used afterwards (i.e., its content is never re-assigned to a new value). This form, known as *Static Single Assignment* (SSA), allows the analysis to map each of the intermediate variables of the program to the node of the control flow graph where it is computed for the first time. All the industry grade open source compilers (e.g., LLVM, GCC, OPEN64) make extensive use of the SSA form in their intermediate representation (IR) languages. A variable is said to be *defined* as a node outcome and *used* in any statement (node) which computes a value depending on it. Transforming a program into SSA form requires to deal with the case of a variable which is defined in more than one statement of the original program. In the basic case, a straight sequence of statements can be easily transformed in SSA form through simply adding extra variables, one for each definition of the multiply defined one. In case the multiple definitions of the same variable lie in two regions separated by a control flow divergence (e.g., in different branches of a selection-construct, or one inside and one outside of a loop body), the problem of knowing which version will be employed by the statements depending on it, can only be resolved at runtime. To overcome this issue, the SSA form employs the ϕ*-function* construct as a placeholder construct. The ϕ-function takes as arguments all the variables among which the runtime selected one will be picked to perform the computation. Note that no code is emitted as a direct translation of the ϕ-function: the compiler simply employs it as a constraint in the register allocation phase. In particular, all the arguments of the ϕ-function are stored in the same register before the statement using the result of the ϕ-function is processed.

As the SSA form of the program allows to do so, it is possible to identify each node of the CFG with the actual new variable being defined by it. From this point on we will thus be using interchangeably the two concepts without the risk of generating ambiguity. To perform a dataflow analysis, the nodes of the CFG are augmented with the actual dataflow information, which is computed via a fixed-point algorithm which behavior depends on the information the dataflow analysis should compute. To the end of determining the influence of the key-material (i.e., the set of values including the input cipher-key and the derived round-keys) on the intermediate values of the cipher algorithm, we will employ the Security DataFlow Analysis (SDFA) framework, as proposed in (Agosta et al., 2013a). In this framework, the dataflow information attached to each node is a bidimensional vector of Boolean values, which are employed to indicate which and how many bits of the key-material influence the computation of each bit of the intermediate variable associated to the node.

For the sake of clarity, we will divide the nodes of the CFG in three different categories, as follows:

Definition 3.2 (Key-material Node)**.** *A key-material node is defined recursively as either i) a node where a memory load operation of a cipher-key portion is performed, or ii) a node which uses only values produced by other key-material nodes.*

The definition of key-material node captures the practical notion of the KEYSCHEDULE computation: in fact, the set of all the key-material nodes corresponds to all the statements computing the KEYSCHEDULE of the block cipher under exam, including the initial load operations of the cipher-key.

Definition 3.3 (Known-value Manipulation Node)**.** *A known-value manipulation node is either a node where a memory load operation of a known value is performed (f.i., the* plaintext *in an encryption algorithm, or the* ciphertext *in a decryption one), or a node which uses only values produced by other known-value manipulation nodes.*

The set of known-value manipulation nodes describes all the actions which a block cipher may perform on known values alone, without the influence of the key (e.g., the initial permutation of DES encryption). This set of nodes is relevant for two reasons: *i)* it produces output values which are only dependent on known values, thus no side-channel attack may be led on them (as their computation lacks the dependency from the key), and *ii)* all the nodes of the CFG which computation does not depend directly or indirectly on them cannot be targeted in a differential side-channel attack, as the computed data does not change when the input is changed.

Definition 3.4 (Cipher-computation Node)**.** *A cipher-computation node is either a node using both a known-value manipulation node and a key-material node to compute its output, or it takes as input at least one other cipher-computation node.*

Cipher-computation nodes are the ones representing the statements of the program which actually compute the block cipher mixing either directly its input with the key-material, or carrying its computation to completion. As such, they are the nodes on which the SCAs focus to derive the values of the key bits.

The SDFA acts on the CFG to the end of evaluating a metric of the computational effort required to lead a SCA against each single bit of the instruction outcome represented by each node. Prior to state a formal definition of the side-channel resistance of an instruction, it is worth remembering that a core block cipher design guideline requires that the combination of the key-material with the outcomes of the intermediate values of cipher should never result in the removal of the effect of previously added key-material. For instance, a key-bit should never be combined via an `xor`-addition twice to the same value, as the second addition would cancel the first. We note that all sound block ciphers are designed striving to achieve this property. By contrast, having an internal cancellation of the key-material contributions would imply that increasing the rounds of the block cipher under exam, its security margin would be reduced, which is a clear design flaw.

Definition 3.5 (Resistance)**.** *The resistance of any bit computed by either an intermediate cipher-computation node or a key-material node, is defined as the minimum number of key-material bits required to derive its value. The resistance of any bit computed by a known-values manipulation node is defined to be infinite, as its value does not depend on any unknown.*

We note that the resistance notion takes into account the fact the attacker may choose to retrieve any of the key-material bits, instead of the cipher-key bits required to compute them. This captures the common practice of attacking the intermediate value produced during the last rounds of a block cipher. In this case, the attacker makes an hypothesis on the value of the last key-material bits used, instead of making his guesses on the cipher-key bits. This strategy allows him to recover a portion of the key-material employed in a certain round (despite the fact this may depend on the whole cipher-key), and subsequently, to exploit the algebraic relations among the bits of the key-material to invert the KEYSCHEDULE and to obtain the cipher-key bits. Computing the resistance metric taking into account only the original cipher-key bit would thus lead to a significant overestimation of the security margin of the cipher.

To efficiently compute the resistance value of all the bits of the cipher-computation nodes it is possible apply two dataflow equations defining how the key-material bit contributions are propagated when the operation corresponding to the statement represented by the node is computed. The application of the equations is repeated on all the CFG nodes until a fixed point is reached. The equations define how the operation under exam propagates the data dependencies from the key-material of its uses into the value it defines. For further details on the form of the dataflow equations we refer the reader to (Agosta et al., 2013a). We note that each application of the dataflow equations can only raise the number of key-material dependencies of the bits of a node, thus the computation always terminates within a finite time.

The worst-case time complexity of such an analysis is given by the case where a single change to the properties of a node triggers the need to re-apply the dataflow equations to all the remaining ones. In addition, the maximum number of modifications to the resistance values of the bits of a node is bounded by the product of the number of key-material bits, by the number of the bits contained in the node. The worst-case time complexity of the analysis is thus $O(|\mathcal{B}|(|\mathcal{B}| - 1) \cdot k \cdot w)$, where k is the number of key-material bits and w the largest number of bits encoding the value of the variable defined by a node. The proposed resistance metric enjoys the following:

Property 3.1 (Key-material nodes resistance)**.** *All the bits of the key material nodes have a resistance value equal to* 1.

This is a direct consequence of the fact that an attacker may directly hypothesize its value, to the end of performing a simple power analysis (SPA) attack. T

The minimum number of key-material bits involved in the computation of an output bit can be obtained, after the fixed point of the dataflow equations application on the CFG has been reached, considering the relations binding them. Thanks to the non cancelling property of the key contributions, the resistance values of the cipher computation nodes enjoy also the following

Property 3.2 (Resistance increases with distance from known-values nodes)**.** *The resistance value of the bits of the cipher-computation nodes increases monotonically on the shortest path between the node which generates them and the closest known-material manipulation node.*

This property captures the well known notion among practitioners that the central rounds of a block cipher tend to be intrinsically more robust against side-channel attacks than the first and last ones.

4 SOUNDNESS OF PARTIAL MASKING

In this section we will state our claim on resistance of a selectively masked algorithm against power-based SCAs, reducing the capabilities of the side-channel attacker to the ones of an attacker able to perform only exhaustive key searches.

We start by stating the side channel attacker model against which we substantiate the resistance of a selectively masked algorithm, starting from an adaptation of the ISW attacker model to an attack against software implementations. The first observation in this context concerns the fact that the protected Boolean operations in the form of combinatorial logic, computed by the hardware (i.e., `xor`, `and`, `not`) are transformed in linear sequences of instructions (referred to as "macroinstruction" from now on) to the end of being executed in software. Consequentially, the single clock cycle restriction for the measurements coming of the presented ISW scheme is translated in terms of the attacker being able to collect at most d measurements within the computation of a single macroinstruction. In addition, the measurements being collected are limited by the impossibility of collecting more than d measures related to the shares of the same value, regardless of when a computation involving them happens. Moreover, we assume that the d measurements performed by the side channel attacker are not direct acquisitions of the computed values, but instead the result of a leakage function $\mathcal{L}$ applied to them.

We thus define our attacker model as follows:

Definition 4.1 (d-th order Software Attacker Model)**.** *The d-th order attacker model is defined as an attacker able to sample any d values of the leakage function $\mathcal{L}$ of the underlying platform, during the computation of each single macroinstruction of a $d+1$ shares protected computation. Once a value has been measured, it counts as measured in all the other macroinstructions. In addition to the measured values, the software attacker is entitled to know the inputs and outputs of the algorithm, except for the values of the secret key bits. From the computational standpoint, the attacker is polynomially computationally bound in the size of the cipher key.*

We note that this attacker model includes the common notion of d-th order attacker employed in practice (Rivain and Prouff, 2010; Mangard et al., 2007), i.e., the one where d samples coming from the measurement of an algorithm execution are employed to lead the attack. Constraining the attacker to the use of d samples from the measurement of the algorithm execution allows the constraint of not collecting more than d measurements of the shares of the same value to be implicitly satisfied. The described d-th order Software Attacker model thus provides a more powerful attacker than the one usually employed to lead a high-order SCA (Rivain and Prouff, 2010; Mangard et al., 2007). In particular, if up to d measurements per macroinstruction are employed to lead a d-order SCA, it is possible to obtain more than one d-wide subset of them leading to a successful attack, as each of the d shares of the same value may be measured in more than one instruction computation. These instructions are the ones computing the macroinstruction defining the target value and the ones using it.

Definition 4.2 (Ideal Attacker)**.** *Given a software cipher implementation running on a target platform, the ideal attacker aims at recovering the value of the cipher-key. He has the capability to choose arbitrary plaintexts/ciphertexts to be encrypted/decrypted and obtain the corresponding outcomes, a number of times polynomially bounded in the cipher key size. Also, he may resort to an amount of computational power which is polynomially bound in the size of the cipher-key.*

Proposition 4.1 (Security of a partially masked implementation)**.** *Let $\mathcal{A}_{\mathtt{sw},d}$ be a d-th order software attacker, and $\mathcal{A}_{\mathtt{ideal}}$ an ideal one (i.e. the one able to perform only an exhaustive search of the whole key). Let C be a software cipher implementation with a k-bit key running on a target platform with leakage*

function $\mathcal{L}$, and C' the result of the application of a $d+1$ shares ISW transformation to all the operations in C having a resistance value r of at least one of their output bits lower than the cipher key length, k. The side channel attacker $\mathcal{A}_{\mathtt{sw},d}$ cannot perform any attack more efficient on C' than the ones its counterpart $\mathcal{A}_{\mathtt{ideal}}$ can lead on C.

Proof. To the end of validating the equivalence of the attacks of the $\mathcal{A}_{\mathtt{sw},d}$ on C' with respect to the ones of $\mathcal{A}_{\mathtt{ideal}}$ on C we will analyze the how $\mathcal{A}_{\mathtt{sw},d}$ can exploit the information derived from the measurements to obtain a more efficient attack. Performing a measure of the values computed by known-values manipulation nodes does not yield any advantage to the $\mathcal{A}_{\mathtt{sw},d}$ with respect to the ideal attacker, as he either already possesses that information or he can compute it in polynomial time. Any d measurements performed on operations of C' concerning the masked values in a macroinstruction yield an information equivalent to a randomly generated one, as proven in (Ishai et al., 2003). Consequentially the side channel attacker $\mathcal{A}_{\mathtt{sw},d}$ does not gain any advantage from these measurements too (note that no information would be gained even in a scenario of an attacker with unbounded computational capabilities). If $\mathcal{A}_{\mathtt{sw},d}$ chooses to perform one or more of his measurements on any sensitive intermediate instruction of C' where the ISW masking has not been applied, he can obtain up to d samples of the $\mathcal{L}$ function for values which depend on at least $r{=}k$ key material bits. We are assuming that the attacker has no way of deriving directly the input value of $\mathcal{L}$ from a single output, a widely accepted assumption in Differential Power Analysis (we note that defining the precise form of a generic $\mathcal{L}$ is still an open problem (Whitnall et al., 2014)). In particular, the attacker is only able to obtain meaningful information through comparing the acquired measures against the outputs of a key-value parametric distinguisher $\mathcal{D}(k)$, typically by means of a statistical test. The distinguisher may be computed in two ways: *1)* either through an a-priori synthetic calculation on the known values and a key hypothesis (f.i., as in classical DPA or CPA attacks), *2)* or through modeling $\mathcal{L}$ through recording the behavior of an identical computing device (f.i., as in a template attack). Analyzing the efficiency of perform the former, we note that the attacker must evaluate the distinguisher function $\mathcal{D}(k)$ on all the 2^k possible key values, thus resulting in a total complexity of $\Theta(2^k f_{\mathcal{D}(k)})$, where $f_{\mathcal{D}(k)}$ is the time complexity of the distinguisher evaluation. Consequentially the complexity of this attack strategy is lower bounded by the one of the plain subkey enumeration, $\Omega(2^k)$. As the only instructions to which the ISW masking is not applied are the ones involving all the key bits, this results into the same computational effort of the ideal attacker. Analyzing the computational effort of the second attack strategy, the attacker needs to collect enough measurements to be able to distinguish the actual measured behavior from one generated by a different key. To this end, he will need to collect at least one measurement per every value taken by all the key bits involved in the computation of the values under attack. As the computation of these values is influenced at minimum by $r{=}k$ key bits, the attacker will thus need to store at least 2^k measurements, fully profiling the behavior of the target device. As the spatial complexity of a computation provides always a lower bound for its time complexity, we can state that the time complexity of this approach is also $\Omega(2^k)$. Thus, in both cases, the advantage provided by measuring the unprotected part only allows to reduce the computational effort of $\mathcal{A}_{\mathtt{sw},d}$ up to $\Omega(2^k)$, which is the same as the ideal attacker one. □

Willing to analyze the relevance of the hypotheses made in the side-channel attacker model, we will now highlight the effect of lifting either the one on the number of measurements, or the assumption that she is not able to derive the input of leakage function $\mathcal{L}$. If the bound of d measurements is lifted, despite the fact that attacking ISW protected cipher portions becomes feasible, the side-channel attacker is unable to gain any information from the portion of the cipher with a resistance equal to the number of key bits, as extracting it still requires a computational effort exponential in the size of the cipher key. Assuming the attacker can obtain the input of the leakage function $\mathcal{L}$ from a single output implies that she can attack even intermediate values of the cipher having a resistance equal to the number of key bits. This can be done measuring two values separated only by a key material addition, and deriving the added key material by difference. This hypothesis relaxation captures the expensive and invasive microprobing attacks, where the attacker directly taps the on-die lines via a small metallic probe. If such attacks fit into the attacker model under consideration, we note that a partial masking of the implementation would not hinder them.

5 CONCLUDING REMARKS

In this work we substantiate that applying a provably secure d-order masking only to a portion of a block cipher yields the same security of an all-out application, against a computationally bound attacker. We based our claim on the reasonable hypothesis that the

attacker is not able to derive directly from a single measurement of the side channel the actual intermediate value being computed. Providing a description of the leakage function, both formally analyzable and modeling the actual experimental evidence, is still a subject for open debate (Galea et al., 2014). Moreover, an interesting research direction is to provide tighter lower bounds for the attacker effort, once such a leakage function has been specified.

REFERENCES

Agosta, G., Barenghi, A., Maggi, M., and Pelosi, G. (2013a). Compiler-based Side Channel Vulnerability Analysis and Optimized Countermeasures Application. In *DAC*, page 81. ACM.

Agosta, G., Barenghi, A., and Pelosi, G. (2012). A Code Morphing Methodology to Automate Power Analysis Countermeasures. In Groeneveld, P., Sciuto, D., and Hassoun, S., editors, *DAC*, pages 77–82. ACM.

Agosta, G., Barenghi, A., Pelosi, G., and Scandale, M. (2013b). Enhancing Passive Side-Channel Attack Resilience through Schedulability Analysis of Data-Dependency Graphs. In Lopez, J., Huang, X., and Sandhu, R., editors, *NSS*, volume 7873 of *Lecture Notes in Computer Science*, pages 692–698. Springer.

Agosta, G., Barenghi, A., Pelosi, G., and Scandale, M. (2014). A Multiple Equivalent Execution Trace Approach to Secure Cryptographic Embedded Software. In *DAC*, pages 1–6. ACM.

Barenghi, A., Pelosi, G., and Terraneo, F. (2013). Secure and Efficient Design of Software Block Cipher Implementations on Microcontrollers. *IJGUC*, 4(2/3):110–118.

Coron, J.-S. (2014). Higher order masking of look-up tables. In Nguyen, P. Q. and Oswald, E., editors, *EUROCRYPT*, volume 8441 of *LNCS*, pages 441–458. Springer.

Coron, J.-S. and Kizhvatov, I. (2010). Analysis and Improvement of the Random Delay Countermeasure of CHES 2009. In *Cryptographic Hardware and Embedded Systems*, pages 95–109.

Coron, J.-S., Prouff, E., and Rivain, M. (2007). Side Channel Cryptanalysis of a Higher Order Masking Scheme. In Paillier, P. and Verbauwhede, I., editors, *CHES*, volume 4727 of *LNCS*, pages 28–44. Springer.

Debraize, B. (2012). Efficient and provably secure methods for switching from arithmetic to boolean masking. In Prouff, E. and Schaumont, P., editors, *CHES*, volume 7428 of *LNCS*, pages 107–121. Springer.

Galea, J. L., Martin, D., Oswald, E., Page, D., and Stam, M. (2014). Making and breaking leakage simulators. Cryptology ePrint Archive, Report 2014/357. http://eprint.iacr.org/.

Garcia, F. D., van Rossum, P., Verdult, R., and Schreur, R. W. (2009). Wirelessly Pickpocketing a Mifare Classic Card. In *IEEE Symposium on Security and Privacy*, pages 3–15. IEEE CS.

Ishai, Y., Sahai, A., and Wagner, D. (2003). Private Circuits: Securing Hardware against Probing Attacks. In Boneh, D., editor, *CRYPTO*, volume 2729 of *LNCS*, pages 463–481. Springer.

Mangard, S., Oswald, E., and Popp, T. (2007). *Power Analysis Attacks - Revealing the Secrets of Smart Cards*. Springer.

Moradi, A., Barenghi, A., Kasper, T., and Paar, C. (2011). On the Vulnerability of FPGA Bitstream Encryption against Power Analysis Attacks: Extracting Keys from Xilinx Virtex-II FPGAs. In Chen, Y., Danezis, G., and Shmatikov, V., editors, *ACM CCS*, pages 111–124. ACM.

O'Flynn, C. and Chen, Z. (2012). A Case Study of Side-Channel Analysis Using Decoupling Capacitor Power Measurement with the OpenADC. In García-Alfaro et al., J., editor, *FPS*, volume 7743 of *LNCS*, pages 341–356. Springer.

Prouff, E. and Rivain, M. (2013). Masking against Side-Channel Attacks: A Formal Security Proof. In Johansson, T. and Nguyen, P. Q., editors, *EUROCRYPT*, volume 7881 of *LNCS*, pages 142–159. Springer.

Rivain, M. and Prouff, E. (2010). Provably Secure Higher-Order Masking of AES. In *Cryptographic Hardware and Embedded Systems, CHES*, pages 413–427.

Schramm, K. and Paar, C. (2006). Higher Order Masking of the AES. In Pointcheval, D., editor, *CT-RSA*, volume 3860 of *LNCS*, pages 208–225. Springer.

Tillich, S. and Herbst, C. (2008). Attacking State-of-the-Art Software Countermeasures-A Case Study for AES. In Oswald, E. and Rohatgi, P., editors, *CHES*, volume 5154 of *LNCS*, pages 228–243. Springer.

Whitnall, C., Oswald, E., and Standaert, F.-X. (2014). The Myth of Generic DPA...and the Magic of Learning. In Benaloh, J., editor, *CT-RSA*, volume 8366 of *LNCS*, pages 183–205. Springer.