Theory and Practice of a Leakage Resilient Masking Scheme

Josep Balasch[1], Sebastian Faust[2], Benedikt Gierlichs[1], and Ingrid Verbauwhede[1]

[1] KU Leuven Dept. Electrical Engineering-ESAT/SCD-COSIC and IBBT
Kasteelpark Arenberg 10, B-3001 Leuven-Heverlee, Belgium
firstname.lastname@esat.kuleuven.be
[2] Aarhus University
Åbogade 34, DK-8200 Aarhus, Denmark
sfaust@cs.au.dk

Abstract. A recent trend in cryptography is to formally prove the *leakage resilience* of cryptographic implementations – that is, one formally shows that a scheme remains provably secure even in the presence of side channel leakage. Although many of the proposed schemes are secure in a surprisingly strong model, most of them are unfortunately rather inefficient and come without practical security evaluations nor implementation attempts. In this work, we take a further step towards closing the gap between theoretical leakage resilient cryptography and more practice-oriented research. In particular, we show that masking countermeasures based on the *inner product* do not only exhibit strong theoretical leakage resilience, but moreover provide better practical security or efficiency than earlier masking countermeasures. We demonstrate the feasibility of inner product masking by giving a secured implementation of the AES for an 8-bit processor.

Keywords: Inner product masking, AES, Leakage resilience

1 Introduction

Side channel attacks (SCA) are among the most relevant threats for the security of implementations of cryptographic algorithms. Since the introduction of timing attacks to the research community in the late 1990s [22], more side channels have been discovered [13, 23, 25] and more powerful attacks have been developed [4, 6, 14]. It was soon clear that masking, i.e. concealing all sensitive intermediate values of a computation with random data, is an excellent way to prevent certain types of attacks [5, 19]. As opposed to other countermeasures aiming at introducing noise in the side channel, e.g. random delays, random order execution, dummy operations, etc., one can formally argue the security masking provides.

The idea of d^{th} order masking is to split every sensitive intermediate value in the implementation into $d + 1$ random shares, and to compute the algorithm on these shares while maintaining that each tuple of d shares is independent of any sensitive value. The challenge is not to devise the masking scheme itself, i.e. to determine how a sensitive intermediate value is split, but rather to define the masked operations that process the independent shares, while still preserving the correctness of the computation. A d^{th} order masked implementation can, in theory, always be broken by a $d+1^{th}$ order side channel attack, i.e. an attack that exploits side channel leakage of $d+1$ intermediate values in the masked implementation. However, given a sufficient amount of noise, attacking a d^{th} order masked implementation becomes exponentially more difficult in d [5]. Motivated by this result, d^{th} order masking schemes (that can be implemented at any order d) based on *boolean* masking [27] and *polynomial* masking [18, 24] have been recently proposed. Unfortunately, their security has so far been evaluated mainly by practice-oriented researchers, while a formal proof-driven analysis is either missing or is given only in a very weak security model.

In the theory community, masking-based countermeasures are analyzed within the framework of leakage resilient circuit compilers introduced by Ishai et al. [20]. A circuit compiler takes as input an arbitrary circuit C computing over some finite field and outputs a protected circuit C' that has the same functionality as C but comes with built-in security against certain classes of leakages. For the circuit compiler of [20] it can be shown that an adversary that learns up to d intermediate values during the computation of the transformed circuit C' does not learn anything beyond black-box access. That is, for instance, if C is an AES circuit then its implementation C' exhibits the standard black-box security even in the presence of side-channel leakage (in the given model).

The circuit compiler of Ishai et al. based on boolean masking with d masks has been recently extended, and a similar compiler (based on *any* linear secret sharing scheme) protecting against broader classes of leakages has been introduced [11]. Despite this progress, it has been suggested that masking schemes with greater algebraic complexity yield better resistance against side channel attacks. As boolean masking schemes only achieve weak provable security guarantees, attempts have been made to seek for alternatives. First examples are the compilers of Juma and Vahlis [21] and Goldwasser and Rothblum [15] which use as underlying masking a public key encryption scheme, i.e. every sensitive variable is encrypted with a suitable public key encryption scheme. While such compilers achieve strong security guarantees, namely, protection against *any polynomial-time computable* leakage function, they suffer from poor efficiency and rather provide theoretical feasibility results than a way towards a practical solution.

In two recent works [10, 16], it was shown that such strong theoretical security guarantees can be achieved without relying on public-key encryption schemes. Instead, these works propose a purely information theoretic solution based on the inner product. While asymptotically these constructions are comparable to schemes based on public key encryption, they have the potential to achieve much better real-world efficiency as they only require simple algebraic operations. In this work, we show that this is indeed the case, if one is willing to accept a weaker security model. In a nutshell, our work shows that advances in leakage resilient cryptography can indeed have implications to real-world implementations and may even provide better practical security or efficiency than existing schemes.

Contributions. We rely on ideas of Dziembowski and Faust [10] for the inner product (IP) masking, and adjust the masked operations to improve their efficiency. As we are particularly interested in a secure implementation of the Advanced Encryption Standard (AES), we can exploit the linearity of the squaring operation in the underlying finite field $\mathbb{F}_{2^8}$. Moreover, we slightly simplify the masked multiplication operation of [10]. All these changes are done without affecting the theoretical security analysis. The bulk of our efficiency improvements, however, comes from using a simpler method to refresh a masked secret. Such a refreshing scheme takes as input a masked secret and outputs a masking of the same value with completely fresh randomness. The construction that we use in our implementation is essentially a simple variant of a scheme proposed in [9]. As such simple schemes only satisfy weaker security properties, we need to make additional restrictions to get a sound theoretical security analysis. We provide further details on how our changes affect the security and what additional assumptions are required in Section 4.

We also evaluate the security of the IP masking for practical parameters, i.e. when the number of shares is small. Our practical analysis reveals that the information leakage of IP masking is more than two orders of magnitude smaller than that of boolean masking for low levels of noise and the same number of shares. Finally, we detail how the AES can be implemented in a secure way using the IP masking scheme, and we provide an implementation and performance results to demonstrate its correctness and feasibility. We show that in particular non-linear operations in the IP masked domain, e.g. multiplication, clearly outperform polynomial-based masking solutions that enjoy similar algebraic complexity.

2 Inner Product Masking

In this section we introduce the circuit model assumed for the execution of the masked calculations, and we provide a detailed description of the masking scheme and its building blocks, including a complexity analysis and a comparison to other masking schemes.

Circuit model. Following the model of Dziembowski and Faust [9, 10], we consider that the target device running the masked computations contains two separate processors. Each of these processors, in the following referred to as *left processor* (P_{L}) and *right processor* (P_{R}), executes a part of the masked operations. Communication between processors is performed via a bidirectional data bus. Such a model is introduced in order to provide a framework to analyze the security of the masking scheme. As will be further explained in the following sections, its main purpose is to facilitate the assumption that P_{L} and P_{R} have completely independent side channel leakage, i.e. an adversary can only retrieve information specific to each physical processor. Notice that from a practical point of view, the required independent side-channel leakage can also be obtained by temporal (rather than physical) separation of the masked computations, e.g. in the context of sequential software implementations on a single processor.

Overview. The IP masking scheme can be instantiated to secure operations in any finite field $|\mathbb{F}| \geq 2$, such that all elements and operations in $\mathbb{F}$ can be mapped to and performed in the masked domain. This feature is extremely useful in the context of securing cryptographic applications, as the underlying field of the masking scheme can be adapted according to the characteristics of the cryptographic algorithm and/or the target platform. Without loss of generality, and driven by our goal to implement the AES, we provide in the following an efficient instantiation of the IP masking scheme for the field $\mathbb{F}_{2^8}$ of characteristic two.

Notation. We represent field elements with upper-case letters, e.g. $X \in \mathbb{F}_{2^8}$, and we use $\oplus$ to denote field addition and $\otimes$ to denote field multiplication. Vectors are represented with bold upper-case letters, e.g. $\boldsymbol{X} \in \mathbb{F}_{2^8}^n$ such that $\boldsymbol{X} = (X_1, \ldots, X_n)$. For two vectors $\boldsymbol{X}, \boldsymbol{Y} \in \mathbb{F}_{2^8}^n$ we denote by $\boldsymbol{X} \oplus \boldsymbol{Y}$ the vector addition in $\mathbb{F}_{2^8}^n$ calculated as $(X_1 \oplus Y_1, \ldots, X_n \oplus Y_n)$. The inner product $\langle \boldsymbol{X}, \boldsymbol{Y} \rangle \in \mathbb{F}_{2^8}$ is calculated as $\bigoplus_{i=1}^n X_i \otimes Y_i$.

Construction. In the IP masking scheme each sensitive variable $X \in \mathbb{F}_{2^8}$ is split into an even number of $2n$ shares such that:

$$X = L_1 \otimes R_1 \oplus \ldots \oplus L_n \otimes R_n. \tag{1}$$

We denote $\boldsymbol{L} = (L_1, \ldots, L_n)$ as *left vector* and $\boldsymbol{R} = (R_1, \ldots, R_n)$ as *right vector*. A variable X is represented in the masked domain as $(\boldsymbol{L}, \boldsymbol{R})$, and can be recovered by calculating the inner product of these two vectors, e.g. $X = \langle \boldsymbol{L}, \boldsymbol{R} \rangle$. In order to prevent a practically exploitable bias between the shares and the masked value, it is required that elements of $\boldsymbol{L}$ belong to $\mathbb{F}_{2^8} \setminus \{0\}$. We define $n \geq 2$ as the security parameter of our masking scheme.

Note that IP masking is a generalization of previously published masking schemes. Indeed, one trivially derives boolean masking from Eq. (1) by e.g. setting all elements in $\boldsymbol{L}$ (resp. $\boldsymbol{R}$) to one. Multiplicative masking [2] can be achieved by setting $n = 2$ and either of the shares L_2 and/or R_2 (resp. L_1 and/or R_1) to zero. Affine masking, described in [12] as $V = (A \otimes X) \oplus B$, can be obtained by fixing $n = 2$, $L_1 = L_2 = A^{-1}$, $R_1 = V$, and $R_2 = B$. Finally, as a secret variable in polynomial masking [18, 24] is given by an interpolation polynomial in the Lagrange form, such masking scheme can be obtained by considering all elements in $\boldsymbol{L}$ to be public Lagrange coefficients.

Algorithm 1 depicts the procedure `IPMask`() to convert a variable X into the IP masked domain as two vectors $(\boldsymbol{L}, \boldsymbol{R})$ of size n. The function `rand`() returns a random element in $\mathbb{F}_{2^8}$, whereas the function `randNonZero`() returns a random element in $\mathbb{F}_{2^8} \setminus \{0\}$. The function `IPUnmask`() to convert a masked variable $(\boldsymbol{L}, \boldsymbol{R})$ of size n back to X consists in calculating the inner product $X = \langle \boldsymbol{L}, \boldsymbol{R} \rangle$.

Algorithm 1 Masking a variable: $(\boldsymbol{L}, \boldsymbol{R}) \leftarrow \texttt{IPMask}(X)$

```
Input: variable $X \in \mathbb{F}_{2^8}$
Output: masked variable $(\boldsymbol{L}, \boldsymbol{R})$
Ensure: $X = \langle \boldsymbol{L}, \boldsymbol{R} \rangle$
  $L_1 \leftarrow \texttt{randNonZero}()$
  for $i = 2$ to $n$ do
    $L_i \leftarrow \texttt{randNonZero}(); R_i \leftarrow \texttt{rand}()$
  end for
  $R_1 \leftarrow (X \oplus \bigoplus_{i=2}^{n} L_i \otimes R_i) \otimes L_1^{-1}$
```

2.1 Operations in the masked domain

After introducing how to convert variables between $\mathbb{F}_{2^8}$ and the IP masked domain, we need to provide a set of high-level functions that allows us to operate directly on the masked variables. In order to fulfill our security requirements, computations regarding the left vector $\boldsymbol{L}$ of masked variables should be executed in the left processor P_{L}, whereas calculations regarding $\boldsymbol{R}$ should be carried out in the right processor P_{R}. Moreover, the condition that elements of the vector $\boldsymbol{L}$ are different than zero must be inherited by all operations in order to avoid output masked values from being biased.

In the following we make use of a special operation called `IPHalfMask`(), which on input a variable X and a vector $\boldsymbol{L}$ calculates the corresponding vector $\boldsymbol{R}$ such that $X = \langle \boldsymbol{L}, \boldsymbol{R} \rangle$. It is thus a simplified version of Algorithm 1 for which the left vector $\boldsymbol{L}$ is already given and thus elements L_i do not need to be sampled.

Another operation that will be often used is `IPRefresh`(). This operation, depicted in Algorithm 2, takes as input a masked variable $(\boldsymbol{L}, \boldsymbol{R})$ and returns a new one $(\boldsymbol{L'}, \boldsymbol{R'})$ such that $\langle \boldsymbol{L}, \boldsymbol{R} \rangle = \langle \boldsymbol{L'}, \boldsymbol{R'} \rangle$. The purpose of the refreshing is to pump new randomness into the masking scheme. Algorithm 2 is tailored particular to work for the field $\mathbb{F}_{2^8}$. For a generalization we refer the reader to [9].

Algorithm 2 Refresh vector: $(\boldsymbol{L'}, \boldsymbol{R'}) \leftarrow \texttt{IPRefresh}(\boldsymbol{L}, \boldsymbol{R})$

Input: vector $\boldsymbol{L}$ in processor P_{L}, vector $\boldsymbol{R}$ in processor P_{R}
Output: vector $\boldsymbol{L'}$ in processor P_{L}, vector $\boldsymbol{R'}$ in processor P_{R}
Ensure: $\langle \boldsymbol{L}, \boldsymbol{R} \rangle = \langle \boldsymbol{L'}, \boldsymbol{R'} \rangle$

$$\begin{array}{lcl} \boxed{\qquad P_{\mathsf{L}} \qquad} & & \boxed{\qquad P_{\mathsf{R}} \qquad} \\ \boldsymbol{A} \in_R \mathbb{F}_{2^8}^n & & \\ \boldsymbol{L'} = \boldsymbol{L} \oplus \boldsymbol{A} & \xrightarrow{\boldsymbol{A}} & X = \texttt{IPUnmask}(\boldsymbol{A}, \boldsymbol{R}) \\ & \xleftarrow{X} & \\ \boldsymbol{B} = \texttt{IPHalfMask}(X, \boldsymbol{L'}) & \xrightarrow{\boldsymbol{B}} & \boldsymbol{R'} = \boldsymbol{R} \oplus \boldsymbol{B} \end{array}$$

Although not clearly specified in Algorithm 2, it is necessary that the vector $\boldsymbol{A}$ sampled by P_{L} is such that the resulting elements of $\boldsymbol{L'}$ are non-zero. In other words, we need to ensure that $A_i \neq L_i$ for all $1 \leq i \leq n$. Details on how to implement this step efficiently, in constant time and flow are given in Section 5.

Addition. The procedure `IPAdd`() to calculate the addition of two masked variables is depicted in Algorithm 3. This algorithm requires a three vector additions, two joint executions of `IPRefresh`(), one of `IPUnmask`(), and one of `IPHalfMask`().

Algorithm 3 Masked addition: $(\boldsymbol{X}, \boldsymbol{Y}) \leftarrow \texttt{IPAdd}((\boldsymbol{L}, \boldsymbol{R}), (\boldsymbol{K}, \boldsymbol{Q}))$

```
Input: vectors $\boldsymbol{L}$ and $\boldsymbol{K}$ in processor $P_{\mathsf{L}}$, vectors $\boldsymbol{R}$ and $\boldsymbol{Q}$ in processor $P_{\mathsf{R}}$
Output: vector $\boldsymbol{X}$ in processor $P_{\mathsf{L}}$, vector $\boldsymbol{Y}$ in processor $P_{\mathsf{R}}$
Ensure: $\langle \boldsymbol{X}, \boldsymbol{Y} \rangle = \langle \boldsymbol{L}, \boldsymbol{R} \rangle \oplus \langle \boldsymbol{K}, \boldsymbol{Q} \rangle$
        $P_{\mathsf{L}}$                                                   $P_{\mathsf{R}}$
              $(\boldsymbol{A}, \boldsymbol{B}) \leftarrow \texttt{IPRefresh}(\boldsymbol{K}, \boldsymbol{Q} \oplus \boldsymbol{R})$
              <====================================>
              $(\boldsymbol{C}, \boldsymbol{D}) \leftarrow \texttt{IPRefresh}(\boldsymbol{L} \oplus \boldsymbol{K}, \boldsymbol{R})$
              <====================================>
              $Z \leftarrow \texttt{IPUnmask}(\boldsymbol{C}, \boldsymbol{D})$
              <====================================>
              $\boldsymbol{Y} \leftarrow \texttt{IPHalfMask}(Z, \boldsymbol{A})$
              <====================================>
    $\boldsymbol{X} = \boldsymbol{A}$                                        $\boldsymbol{Y} = \boldsymbol{Y} \oplus \boldsymbol{B}$
```

Notice that it might be the case that the component $\boldsymbol{L} \oplus \boldsymbol{K}$ in the second execution of `IPRefresh`() has elements equal to zero. While this is a source of first-order leakage in IP masking, i.e. the probability $\Pr(Z = 0 | (L_i \oplus K_i) = 0)$ is twice than that for any other value of Z, it is in this particular case not exploitable by an attacker. This is because $\Pr(\langle \boldsymbol{X}, \boldsymbol{Y} \rangle | Z = 0)$ is uniformly distributed, i.e. knowing that the intermediate value Z is zero does not give any information about the sensitive output value $(\boldsymbol{X}, \boldsymbol{Y})$.

Addition of a constant. The procedure `IPAddConst`() to add a constant $Z \in \mathbb{F}_{2^8}$ to a masked variable $(\boldsymbol{L}, \boldsymbol{R})$ can be carried out more efficiently than Algorithm 3. Let $(\boldsymbol{L}, \boldsymbol{R})$ and Z be the input operands, and $(\boldsymbol{X}, \boldsymbol{Y})$ the output masked variable. Addition of a constant can be simply calculated by letting $\boldsymbol{X} = \boldsymbol{L}$ and $\boldsymbol{Y} = \boldsymbol{R}$, except for the first element $Y_1 = (R_1 \oplus Z) \otimes L_1^{-1}$.

Multiplication. The procedure `IPMult`() to calculate the multiplication of two masked variables is depicted in Algorithm 4. This algorithm requires $2n^2$ initial field multiplications, one execution of `IPRefresh`() with input/output vectors of size n^2, one execution of `IPUnmask`() with input vectors of size $n^2 - n$, one execution of `IPHalfMask`(), and one final vector addition.

Multiplication by a constant. The procedure `IPMulConst`() to multiply a masked variable $(\boldsymbol{L}, \boldsymbol{R})$ by a constant $Z \in \mathbb{F}_{2^8}$ is efficiently computed in IP masking. Let $(\boldsymbol{L}, \boldsymbol{R})$ and Z be the input operands, and $(\boldsymbol{X}, \boldsymbol{Y})$ be the output masked variable. Multiplication by a constant can be performed by letting $\boldsymbol{X} = \boldsymbol{L}$ and calculating $\boldsymbol{Y} = (R_0 \otimes Z, \ldots, R_n \otimes Z)$. As will be further explained in Section 4, it is not necessary to execute `IPRefresh`() after `IPMulConst`().

Algorithm 4 Masked multiplication: $(\boldsymbol{X},\boldsymbol{Y}) \leftarrow \texttt{IPMult}((\boldsymbol{L},\boldsymbol{R}),(\boldsymbol{K},\boldsymbol{Q}))$

```
Input: vectors $\boldsymbol{L}$ and $\boldsymbol{K}$ in processor $P_{\mathsf{L}}$, vectors $\boldsymbol{R}$ and $\boldsymbol{Q}$ in processor $P_{\mathsf{R}}$
Output: vector $\boldsymbol{X}$ in processor $P_{\mathsf{L}}$, vector $\boldsymbol{Y}$ in processor $P_{\mathsf{R}}$
Ensure: $\langle \boldsymbol{X},\boldsymbol{Y}\rangle = \langle \boldsymbol{L},\boldsymbol{R}\rangle \otimes \langle \boldsymbol{K},\boldsymbol{Q}\rangle$

  $P_{\mathsf{L}}$                                          $P_{\mathsf{R}}$
  for $i = 1$ to $n$ do                                 for $i = 1$ to $n$ do
    for $j = 1$ to $n$ do                                 for $j = 1$ to $n$ do
      $\tilde{U}_{i*n+j} \leftarrow L_i \otimes K_j$                     $\tilde{V}_{i*n+j} \leftarrow R_i \otimes Q_j$
                  $\Longleftarrow (\boldsymbol{U},\boldsymbol{V}) \leftarrow \texttt{IPRefresh}(\tilde{\boldsymbol{U}},\tilde{\boldsymbol{V}}) \Longrightarrow$
  $\boldsymbol{A} = (U_1,\ldots,U_n)$                             $\boldsymbol{B} = (V_1,\ldots,V_n)$
  $\boldsymbol{C} = (U_{n+1},\ldots,U_{n^2})$                        $\boldsymbol{D} = (V_{n+1},\ldots,V_{n^2})$
                  $\Longleftarrow Z \leftarrow \texttt{IPUnmask}(\boldsymbol{C},\boldsymbol{D}) \Longrightarrow$
                  $\Longleftarrow \boldsymbol{Y} \leftarrow \texttt{IPHalfMask}(Z,\boldsymbol{A}) \Longrightarrow$
  $\boldsymbol{X} = \boldsymbol{A}$                                     $\boldsymbol{Y} = \boldsymbol{Y} \oplus \boldsymbol{B}$
```

Squaring. The procedure `IPSquare`() can be carried out quite efficiently in the masked domain given that we work over a field of characteristic 2. Let the input masked variable be $(\boldsymbol{L},\boldsymbol{R})$. The output masked variable $(\boldsymbol{X},\boldsymbol{Y})$ can be calculated by squaring all elements of each vector independently, i.e. $X_i = (L_i)^2$ and $Y_i = (R_i)^2$. The masked squaring operation does not require refreshing the masks, and can be thus carried out with only $2n$ field squarings.

2.2 Complexity of operations

The complexity of the main operations in the IP masked domain, namely addition and multiplication, is given in Table 1. We also provide a comparison with some masked operations that can be implemented at any order d, recently published in the literature for boolean and polynomial masking schemes, namely [18, 24, 27]. The complexity numbers are given in terms of d for all the schemes, where d indicates the number of random values in each masked variable. Recall that in IP masking, this number of random values is given by $d = 2n - 1$, with $n \geq 2$.

As shown in Table 1, the complexity of the addition operation in IP masking is slightly larger than in the other proposed methods. This is mainly due to the internal use of the `IPRefresh`() operation which, as opposed to the other masking schemes, involves several field multiplications. However, the results obtained for the multiplication operation are favourable for IP masking. In particular, both polynomial masked multiplications have complexity $O(d^3)$ while IP masked multiplications have complexity $O(d^2)$. The boolean masked multiplication has a similar complexity but, as we will show in the next sections, the masking scheme itself provides considerable less security from both practical and theoretical points of view.

Table 1. Complexity of IP masked operations and comparison to d^{th} order boolean masked operations and polynomial masked operations in the literature.

Masked		**Operations in $\mathbb{F}_{2^8}$**			
Operation	**Scheme**	$\oplus$	$\otimes$	x^{-1}	**Rand**
ADDITION	Boolean [27]	$d+1$	-	-	-
	Polynomial [18]	$d+1$	-	-	-
	Polynomial [24]	$d+1$	-	-	-
	Inner Product	$(13d+1)/2$	$3d+3$	3	$(7d+3)/2$
MULTIPLICATION	Boolean [27]	d^2+d+1	$2d^2+2d$	-	$(d^2+d)/2$
	Polynomial [18]	$2d^3+7d^2+d$	$2d^3+5d^2+5d$	-	$2d^2+d$
	Polynomial [24]	$4d^3+8d^2+7d+2$	$4d^3+8d^2+3d$	-	$2d^2+d$
	Inner Product	$(5d^2+12d-9)/4$	$(5d^2+10d+5)/4$	2	$(3d^2+8d-3)/4$

3 Security Evaluation

In this section we evaluate the SCA resistance of IP masking and compare it to that of other masking schemes that can be implemented at any order, e.g. boolean masking and polynomial masking. We focus the analysis on the masking schemes themselves, i.e. we analyze the leakage of the shares of one masked value. We will show in the next section that the security relevant properties of IP masking carry over to the basic operations in the masked domain.

Attack order. We begin the evaluation by deriving the minimum order for an attack against IP masking. For this we need the following definitions:

Definition 1: We say that a variable is sensitive, if it is an intermediate result in an implementation that leaks through side channels, and if it is a function of the input (resp. output), the key and possibly other constants that is not constant with respect to the key [27].

Definition 2: We say that a masking scheme is d^{th} order SCA secure, if every tuple of d or less shares is independent of the variable that is masked. Accordingly, a masked implementation of an algorithm is d^{th} order SCA secure, if every tuple of d or less intermediate variables is independent of any sensitive variable.

1^{st} *order SCA resistance.* Clearly, IP masking with $n \geq 2$ is 1^{st} order SCA secure. This is a simple consequence of the fact that, even if the value of one of the shares in $\mathbf{L}$ or $\mathbf{R}$ is known (in the worst case one R_i is known to be zero such that $L_i \otimes R_i = 0$), the value of the variable that is masked is still information theoretically hidden by the $\oplus$ with $n-1$ terms that are all uniformly distributed over $\mathbb{F}_{2^8}$.

2^{nd} *order SCA resistance.* IP masking with $n = 2$ is not 2^{nd} order SCA secure. This is because the product of two values is determined to be zero if one of the values is zero. Multiplicative masking [2] suffers from the same problem [17]. Suppose that the values of R_1 and R_2 are known to be zero. Then, $L_1 \otimes 0 \oplus L_2 \otimes 0 = s = 0$. This leads to a bias in the distribution $p(S = s | R_1 = r_1, R_2 = r_2)$, and the mutual information $I(s; (R_1, R_2))$ is non-zero.

d^{th} *order SCA resistance.* IP masking with $2n = d + 1$ is SCA secure up to $n - 1^{th}$ (or $\frac{d+1}{2} - 1^{th}$) order, but not secure against n^{th} (or $\frac{d+1}{2}^{th}$) order SCA. Following the above examples, as long as the product of one pair $(L_i, R_i), i \in \{1, \ldots, n\}$ is unknown, the value of the variable s that is masked is still information theoretically hidden. On the other hand, if $\forall i \in \{1, \ldots, n\}$ the value of R_i is known to be zero, then the value of s is known to be zero. However, the probability that this case occurs is small and decreases rapidly with increasing n. More precisely, it is (2^{-8n}).

In summary, IP masking with $2n = d + 1$ can, in theory, be broken by a n^{th} order SCA. On the other hand, similar to polynomial masking, it creates a much more complex relation between the shares than boolean masking, which is known to be more difficult to exploit. Hence, we expect IP masking with $2n = d + 1$ to provide much higher security in practice than boolean masking of order $d+1$, i.e. with the same number of random masks. Following this line, we opt to consider the leakage of all $2n$ or $d + 1$ shares in the following analysis, since an attack exploiting all shares is more powerful in an information theoretic sense, unless the noise levels are extremely high.

In polynomial masking half of the shares are non-zero public constants and the other half are random and secret masks. In particular, there is no direct correspondence to the notion of a masked variable. In the rest of the paper we refer only to the random and secret shares, and their number determines the masking order. For example, polynomial masking of order $d - 1$ uses d random and secret shares, and can theoretically be broken by a d^{th} order SCA. We will compare polynomial masking of order $d - 1$ with boolean masking of order d ($d + 1$ shares, d masks) and with IP masking of order $2n = d + 1$ ($d + 1$ shares, d masks). One could expect IP masking with $2n = d + 1$ to provide a similar level of security as polynomial masking of order $d - 1$, i.e. both schemes should provide similar security when they use the same number of random and secret masks.

Information Leakage. As motivated and done in previous works [12, 24, 28, 29], we use the mutual information between a variable and the leakage of all shares of its masked representation as a figure of merit. We estimate it using simulations. For IP masking, we set $n = 2$ and let $R_2 \in_R \mathbb{F}_{2^8}$ and $L_1, L_2 \in_R \mathbb{F}_{2^8} \setminus \{0\}$ such that $S = L_1 \otimes R_1 \oplus L_2 \otimes R_2$. Boolean masking uses $d+1$ shares $(M_1, \ldots, M_d, V)$ where the $M_i \in_R \mathbb{F}_{2^8}$ and V is computed such that $S = M_1 \oplus \ldots \oplus M_d \oplus V$ holds. We evaluate boolean masking for $d \in \{1, 2, 3\}$. Polynomial masking uses d shares $(Y_1, \ldots, Y_d)$ with $Y_i \in_R \mathbb{F}_{2^8}$ and d public Lagrange coefficients $(\beta_1, \ldots, \beta_d)$ with $\beta_i \in_R \mathbb{F}_{2^8} \setminus \{0\}$ and pairwise distinct [18]. We evaluate polynomial masking for $d \in \{2, 3\}$.

To quantify the amount of information leaked, we need to model the relation between the value of a variable and its physical leakage. We follow the approach that is usual in the literature [12, 24, 29]: we model that a variable leaks its Hamming weight, that each share leaks independently of all other shares, and that the leakage of each share is affected by independent Gaussian noise. The latter serves to mimic the noise effects that affect physical measurements. Putting this together, we model the leakage of IP masking as

$$\mathrm{Leak}(\mathbf{L}, \mathbf{R}) = (\mathrm{HW}(L_1) + n_1, \mathrm{HW}(R_1) + n_2, \mathrm{HW}(L_2) + n_3, \mathrm{HW}(R_2) + n_4)\,,$$

the leakage of boolean masking as

$$\mathrm{Leak}(M_1,\ldots,M_d,V) = (\mathrm{HW}(M_1)+n_1,\ldots,\mathrm{HW}(M_d)+n_d,\mathrm{HW}(V)+n_{d+1})$$

and the leakage of polynomial masking as

$$\mathrm{Leak}(Y_1,\ldots,Y_d) = (\mathrm{HW}(Y_1)+n_1,\ldots,\mathrm{HW}(Y_d)+n_d)$$

where the n_i are independent Gaussian variables with mean zero and standard deviation σ. The mutual information is then $I(S;\mathrm{Leak}(\mathbf{L},\mathbf{R}))$, $I(S;\mathrm{Leak}(M_1,\ldots,M_d,V)$ resp. $I(S;\mathrm{Leak}(Y_1,\ldots,Y_d))$. The number of measurements that a Template Attack [6], i.e. the worst case scenario of a profiled attack, requires to achieve a given success probability is directly related to this mutual information via $c\cdot I(\cdot;\cdot)^{-1}$, where the constant c is related to the success probability [29].

Figure 1 shows plots of the mutual information ($\log_{10}$) between S and the information leaked by all shares of its masked representation, over increasing noise levels σ, for all masking schemes considered[3] [4].

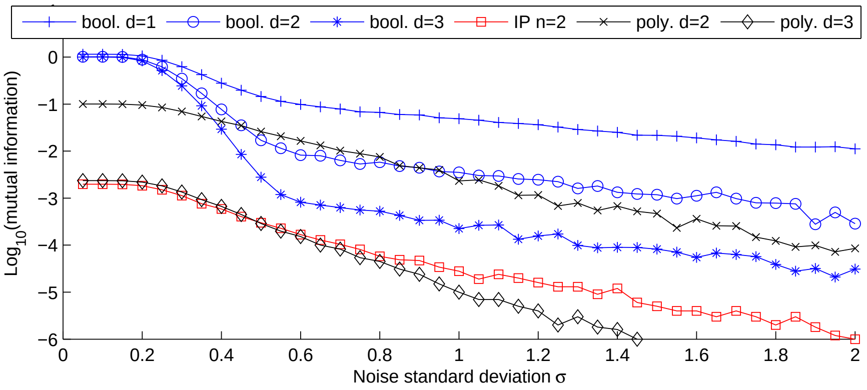

Fig. 1. Mutual information ($\log_{10}$) over increasing noise standard deviation σ for different masking schemes.

The figure shows that IP masking with $n = 2$ leaks consistently less than boolean masking with $d \in \{1, 2, 3\}$ across the range of tested noise levels, which confirms our expectation. The advantage is more pronounced for low noise levels, where e.g. for $\sigma = 0.2$ the information leakage of IP masking is about 2.5 orders of magnitude(!) smaller than that of boolean masking. As expected, polynomial

[3] Note that the mutual information values we computed for boolean masking are consistent with Figure 1 in [12] and Figure 3 in [24]. One has to take into account that the Y-axis in those figures is erroneously labeled $\log_{10}$ while it should be $\log_n$ [1].

[4] For polynomial masking with $d = 3$, reasonably accurate estimation of the mutual information values for high noise levels is beyond our computational budget.

masking with $d = 2$ leaks consistently more than IP masking with $n = 2$. Polynomial masking with $d = 3$ provides a level of security very similar to IP masking with $n = 2$ for low noise levels. However, contrary to what one could expect, for high noise levels, polynomial masking with $d = 3$ leaks less than IP masking with $n = 2$. There are several possible explanations for this observation. For instance, IP masking with $n = 2$ involves two field multiplications while polynomial masking with $d = 3$ involves three field multiplications, i.e. the algebraic complexity of the masking is greater. Furthermore, IP masking with $n = 2$ is 1^{st} order SCA secure while polynomial masking with $d = 3$ is 2^{nd} order SCA secure. It is known that leakage of lower order is easier to exploit, in particular with increasing noise [29]. We leave the careful analysis of the observed difference in information leakage as an open question for future research.

Discussion. Our evaluation shows that IP masking with $n = 2$ provides high security even if there is little noise. However, although the simulated scenario (Hamming weight leakage, independent leakage of each share, Gaussian noise) is standard in the practice-oriented literature, it is synthetic and in particular meets the requirement of the masking schemes for independent leakage perfectly. It can be hard to achieve this for real-world implementations that are affected by effects such as coupling (we show in the extended version [3] that glitches do not affect the security of IP masking). Clearly, our evaluation does not allow to blindly assume that an implementation of IP masking is secure. What it shows is the level of security that a secure implementation of IP masking can provide. An interesting topic for future research is to analyze the security provided by a real-world implementation, and to analyze how violating a requirement, e.g. independent leakage, affects practical security.

4 Theoretical Security Analysis

In this section, we review some formal security properties of the IP masking. We give the basic security properties of the masking scheme itself, including very strong security guarantees with respect to non-adaptive leakages, and argue that these properties carry over to the basic operations in the masked domain. In the full version [3] we discuss further relaxations, and argue that our construction provides security against glitches similar to the results given in [24].

Notation. In the following we let $\mathbb{F}$ be a finite field, and we typically consider row vectors. We define the statistical distance between two random variables A, B over some set $\mathcal{X}$ as

$$\Delta(A; B) := \sum_{x \in \mathcal{X}} 1/2 \left| \Pr[A = x] - \Pr[B = x] \right| .$$

4.1 Security properties of IP masking

We have argued in Section 3 that even for small n, IP masking is robust to (noisy) Hamming weight leakage from the different shares of the masking. In this section, we back up these observations with a theoretical analysis showing strong security properties for IP masking that cannot be achieved, e.g. by linear masking schemes such as Boolean masking or masking schemes based on Shamir secret sharing. The analysis strongly relies on techniques and results from [10, 11]. We repeat here part of the arguments where changes to the construction and model are required to get practical constructions. For a more formal analysis and full proof details the reader is referred to [10]. We emphasize that the theoretical analysis will typically require $n > 130$ to get meaningful security bounds.

As mentioned in Sect. 2, we assume that the device that runs the masked computation has two processors, P_{L} and P_{R}, leaking independently. Let S_L denote the state of processor P_{L} and S_R the state of processor P_{R} (resp.), then the adversary may interact with $\Omega(\mathsf{S}_L, \mathsf{S}_R)$ by sending functions f_L and f_R to the oracle and getting back $f_L(\mathsf{S}_L)$ and $f_R(\mathsf{S}_R)$. The only additional requirement that we make is that an adversary will not learn more than λ bits from each processor P_{L} and P_{R}. We call such an adversary λ-limited and denote the process of the adversary interacting with the leakage oracle by $(\mathcal{A} \leftrightarrows \Omega(\boldsymbol{L}, \boldsymbol{R}))$. For simplicity, we always assume that the output of $\mathcal{A}$ in the above leakage game is $f_L^1(\mathsf{S}_L), f_L^2(\mathsf{S}_L), \dots, f_R^1(\mathsf{S}_R), f_R^2(\mathsf{S}_R), \dots$. We emphasize that by modeling leakage in this way, we allow it to depend on any intermediate value that may be computed during the computation of the two processors.

To analyze the security of an IP masked value S from some finite field $\mathbb{F}$, we set $\mathsf{S}_L := \boldsymbol{L}$ and $\mathsf{S}_R := \boldsymbol{R}$, where $(\boldsymbol{L}, \boldsymbol{R}) \leftarrow \mathtt{IPMask}_n(S)$, and let the adversary interact with the $\Omega(\boldsymbol{L}, \boldsymbol{R})$ leakage oracle. The following lemma was proven in [10].

Lemma 1. *Let $n \in \mathbb{N}$ and let $\mathbb{F}$ be such that $n \geq \log(|\mathbb{F}|)$. For any $1/2 > \delta > 0, \gamma > 0$, any two secrets $S, S' \in \mathbb{F}$ and any (unbounded) λ-limited adversary $\mathcal{A}$ we have*

$$\Delta((\mathcal{A} \leftrightarrows \Omega(\mathtt{IPMask}_n(S))), (\mathcal{A} \leftrightarrows \Omega(\mathtt{IPMask}_n(S'))) \leq \epsilon,$$

where $\lambda = (1/2 - \delta)n \log |\mathbb{F}| - \log \gamma^{-1} - 1$ and $\epsilon = 2(|\mathbb{F}|^{3/2 - n\delta} + |\mathbb{F}|\gamma)$.

Informally, the lemma says that for any two (different) secrets S, S' no adversary can distinguish between leakage from a masking of S and a masking of S'.

As a special case, this gives us the following corollary when the underlying field is $\mathbb{F}_{2^8}$, namely:

Corollary 1. *Let $n \in \mathbb{N}$, then for any two secrets $S, S' \in \mathbb{F}_{2^8}$ and any λ-limited adversary $\mathcal{A}$, we have*

$$\Delta((\mathcal{A} \leftrightarrows \Omega(\mathtt{IPMask}_n(S))), (\mathcal{A} \leftrightarrows \Omega(\mathtt{IPMask}_n(S'))) \leq \epsilon,$$

where $\lambda = 3n$ and $\epsilon \leq 2^{13 - 0.1n} + 2^{-n}$.

Proof. Set $\delta := 0.1$ and $\gamma := 2^{-0.2n}$, then we get $\lambda = 3.2n - 0.2n - 1 = 3n$ and $\epsilon \leq 2^{13 - 0.1n} + 2^{-n}$. □

Corollary 1 says that for sufficiently large n an adversary may learn up to $3n$ bits from each processor without being able to distinguish between a masking of S and S'. We notice that the bound on the statistical distance only gets meaningful, when $n > 130$, which, of course, is impractical.

One may ask if we can get stronger security guarantees for the masking scheme if we restrict our focus to certain special cases. To this end consider the case that the adversary cannot query adaptively the leakage oracle, i.e. he may learn only $f_L(\boldsymbol{L})$ and $f_R(\boldsymbol{R})$. In this case, it is easy to show that from the fact that the inner product is a strong randomness extractor [7, 26], we can give to the adversary the entire $\boldsymbol{L}$ and up to $3n$ bits of $\boldsymbol{R}$, and still it will be hard to decide whether $(\boldsymbol{L}, \boldsymbol{R})$ was sampled from $\texttt{IPMask}_n(S)$ or $\texttt{IPMask}_n(S')$.

Comparison with linear masking schemes. We notice that linear masking schemes, such as the additive masking over finite fields [20, 27], cannot achieve such strong security properties in our security model. Consider a secret $S \in \mathbb{F}$ that is masked by vectors $(\boldsymbol{L}, \boldsymbol{R})$ such that $(\boldsymbol{L}, \boldsymbol{R})$ are uniformly random in $\mathbb{F}^{2n}$ subject to the constraint that $S = \sum_i L_i + \sum_i R_i$. If we consider an adversary that can interact with $\Omega(\boldsymbol{L}, \boldsymbol{R})$ then already a single field element of leakage entirely breaks the security: $f(\boldsymbol{L})$ may reveal $\sum_i L_i$, while $g(\boldsymbol{R})$ reveals $\sum_i R_i$, which together reveal S completely.

For fields of characteristic 2 such as $\mathbb{F}_{2^8}$ already a single bit of leakage suffices to learn information about the secret! Recall that in characteristic 2 fields addition works bit-wise. Similar arguments work for the polynomial masking based on Shamir secret sharing introduced in [18], as Lagrange polynomial interpolation is linear. Hence, such masking schemes can be broken in our model.

We emphasize that our leakage model includes certain classes of leakages that are very frequently used in practice, e.g. to model power consumption. One example is the Hamming weight leakage model. Of course, our theoretical analysis includes Hamming weight leakages as an adversary can learn the Hamming weight of a masked value and still the masked value remains information theoretically hidden. More precisely, as shown in Corollary 1 the IP masking remains provably secure even if an adversary learns $\boldsymbol{L}$ completely and $3n$ bits of $\boldsymbol{R}$. As Hamming weight is a linear function we can compute the Hamming weight of $(\boldsymbol{L}, \boldsymbol{R})$ from just the Hamming weight of $\boldsymbol{L}$ *and* $\boldsymbol{R}$ separately. Notice that the Hamming weight of $\boldsymbol{R}$ can be compressed to $< 4 \log n$ bits, while according to Corollary 1 we are allowed to learn $3n$ bits of $\boldsymbol{R}$. We emphasize that an adversary may even learn the individual Hamming weight of each share $R_1, \ldots R_n$ of the right vector and still the IP masking remains secure. This is easy to see as we can describe the Hamming weight of the n shares for sufficiently large n by at most $n \log(8) = 3n$ bits, which according to Corollary 1 an adversary may learn from $\boldsymbol{R}$. We emphasize that for additive masking schemes, such as Boolean masking, it is not known whether such strong security guarantees hold.

4.2 Security of masked operations

So far, we looked at the robustness of the IP masking scheme in the presence of independent leakage, when we mask a secret value (or several secret values) and store the left part $\boldsymbol{L}$ on processor P_{L}, while $\boldsymbol{R}$ is stored on processor P_{R}. In the following, we "lift" the security analysis from just masking the secret state, e.g. the key of the AES, to arbitrary computation with masked values. More precisely, we describe why leakage from operations on masked values will not help to learn more about the masked value than just the leakage from a single masking. This can be viewed as a reduction from the security of "complicated" masked computation, to the security of a single masked value. The details of this analysis can be found in the full version.

In the security proof, we follow Dziembowski and Faust [10] and show two simple properties for the basic masked operations. These properties were introduced in [11] and are called *rerandomizing* and *reconstructability*. The first guarantees that for a masked operation the encoded output of the operation is distributed as a uniformly and independently sampled encoding. We show in the full version that all our masked operations satisfy this property. We notice that the algorithms for squaring and multiplication by a constant require only local computation, and hence do not require a refreshing.

To show reconstructability for a masked operation, we need to build a *reconstructor*. A reconstructor is a simulator that given the operations's masked inputs and outputs can reproduce the internal computation of the operation. The main requirement is that leakage from the reconstructor's output distribution (namely the internal computation) is indistinguishable from the leakage obtained from a real execution of the operation. At an intuitive level, this property guarantees that leakage from the internals of a masked operation will not reveal "more" information about the underlying secret than just the leakage from the masked inputs and outputs itself.

For practical reasons, we slightly adapt the construction of [10]. The three main differences are as follows: (1) the way in which we refresh masked secrets, (2) dedicated efficient masked operations for squaring and multiplication by a constant, and (3) a simplified masked multiplication operation (instead of a NAND we only build a simple multiplication). We discuss some details below. A more thorough discussion is deferred to the full version.

In the implementation, we use Algorithm 2 to refresh a masking of $(\boldsymbol{L}, \boldsymbol{R})$, which is a simple variant of the scheme given in [10]. To enable a security proof, we will in the following assume that the refreshing does not leak. This is required as Dziembowski and Faust show a theoretical attack on a similar refreshing scheme in [9]. Unfortunately, their attack also applies on the refreshing from Algorithm 2. The attack presented in [9] recovers the masked secret and requires an adversary to learn for n consecutive rounds the exact value of 3 field elements. While in theory such an attack completely breaks the masking scheme, we emphasize that for a real-world adversary it is very hard to learn the exact value of field elements. If learning the exact value of 3 field elements over n consecutive rounds is possible, then from a practical point of view it seems hard to argue why the adversary should not be able to learn the exact value of all $2n$ shares in one round of the refreshing. Notice also that practical SCA attacks typically require some knowledge of the inputs/outputs of the algorithm.

For the refreshing algorithm this is not possible as both inputs and outputs are unknown and random. This makes attacking the refreshing a hard target. One may ask why we do not use alternative approaches of provably secure refreshing as presented in [9] and [16]. Our choice is motivated by practical limitations as existing refreshing schemes result in a quadratic blow-up.

5 Performance Evaluation

In this section we evaluate the performance and correctness of IP masking. We provide a general overview on how to implement the IP masking building blocks on an 8-bit embedded platform, and describe how to use them to protect an implementation of the AES.

5.1 Implementation of masked operations

The 8-bit Atmel AVR ATMega128 [8] is chosen as target platform. This device provides an advanced RISC architecture with 133 low-level instructions and it offers 128 kBytes of flash program memory and 4 kBytes of internal SRAM. The independent side channel leakage required by our model is in our implementation achieved by temporal separation, i.e. instead of using two physically separated processors P_{L} and P_{R}, we use a single 8-bit processor and we ensure independent leakage by not overlapping their respective operations.

For the sake of optimization, we have implemented all operations in assembly language. The ATMega128 does not provide an internal random number generator to implement the `rand`() and `randNonZero`() functionalities. Therefore, and only for the purposes of evaluating the implementation, the required random bytes are provided to the microcontroller externally previous to the encryption process. We note that modern devices with built-in TRNG or PRNG elements running in parallel would allow to generate such randomness internally.

Addition in $\mathbb{F}_{2^8}$ is carried out in a single clock cycle via the available XOR instruction, whereas the rest of field operations (multiplication, inversion, raisings to the power of 2) are implemented via lookup tables, requiring a total of 1,536 bytes in program memory. Besides the squaring, we have also implemented as lookup tables the rising to the powers of 4 and 16 required in the power function of the AES `SubBytes` step (see extended version for more details [3]). On devices with limited program memory these raisings can be alternatively carried out by consecutive squarings, effectively saving 512 bytes of program memory.

Special care has been taken in order to make the implementation not only time-constant, but flow-constant i.e. conditional execution paths, which can be a potential source of side channel leakage, have been avoided. A typical example of a function with conditional execution is the multiplication in $\mathbb{F}_{2^8}$ using log/alog tables. This method only works when both input operands are different than zero; otherwise, the result of the multiplication must be equal to zero. Implementing this routine in constant flow requires to calculate the potential outputs of *all* conditional paths, and thus it ends up requiring 22 clock cycles.

Worth mentioning is the implementation of the first part of Algorithm 2 for mask refreshing, namely sampling a vector $\boldsymbol{A}$ such that $A_i \neq L_i$ for $1 \leq i \leq n$. This step is carried out as follows for each element A_i. First, we sample two elements $A'_i \in \mathbb{F}_{2^8}$ and $A''_i \in \mathbb{F}_{2^8} \setminus \{0\}$. If $A'_i \neq L_i$ we simply set $A_i = A'_i$; otherwise, we assign $A_i = A'_i \oplus A''_i$. Independently of the sampled values A'_i and A''_i, this conditional statement ensures that i) the final value A_i is different than L_i, and ii) the final value of A_i is uniformly distributed over $\mathbb{F}_{2^8}$. Needless to say, such implementation is also performed in constant flow execution to prevent conditional execution branches.

5.2 Application to the AES

We have implemented and verified the correctness of a protected instance of AES using the IP masking scheme with $n = 2$. Due to space restrictions, we provide a high-level description about how to apply IP masking to the AES in the extended version of this work [3]. As shown in Table 2, our implementation requires around $1.9 \cdot 10^6$ clock cycles to perform a protected AES encryption (including on-the-fly key schedule calculation).

Table 2. Performance evaluation (in clock cycles) of AES round transformations and AES encryption with IP masking scheme with $n = 2$.

`AddRoundKey`	`SubBytes` (Inverse)	`SubBytes` (Aff.Transf.)	`ShiftRows`	`MixColumns`	**Full AES**
8,796	45,632	72,128	200	27,468	1,912,000

We stress that these results should not be simply taken as an indicator to judge the practicality of IP masking, as they are obtained using a legacy general-purpose device without any type of hardware enhancements. If multiplication in $\mathbb{F}_{2^8}$ was available in the instruction set of the controller our timing for AES encryption would be instantly reduced to less than a million cycles. This could be achieved e.g. by providing instruction set extensions to the target device.

6 Conclusion

This work narrows the gap between the fields of theoretical leakage resilient cryptography and practice-oriented research, and it represents a first joint step towards the development and evaluation of common masking schemes. Although the levels of security required for each model differ considerably, we expect tighter bounds that allow to lower the value of the security parameters as the theory of leakage resilient cryptography advances. At the same time, technology advances steadily and what was impractical yesterday will be “normal” tomorrow. As a consequence one might expect that schemes such as IP masking can become practical for higher security levels.

Acknowledgment. This work was supported in part by the European Commission's ECRYPT II NoE (ICT-2007-216676), by the Flemish Government FWO G.0550.12N, by the Hercules Foundation AKUL/11/19, and by the Research Council KU Leuven: GOA TENSE (GOA/11/007). Benedikt Gierlichs is a Postdoctoral Fellow of the Fund for Scientific Research - Flanders (FWO). Sebastian Faust acknowledges support from the Danish National Research Foundation and The National Science Foundation of China (under the grant 61061130540) for the Sino-Danish Center for the Theory of Interactive Computation, within part of this work was performed; and from the CFEM research center, supported by the Danish Strategic Research Council. Josep Balasch is funded by a PhD grant within the covenant between KU Leuven and R.U. Nijmegen.

References

1. Personal communication with the respective authors, 2012.
2. M.-L. Akkar and C. Giraud. An implementation of DES and AES, secure against some attacks. In Çetin K. Koç, D. Naccache, and C. Paar, editors, *Proceedings of CHES 2001*, volume 2162 of *LNCS*, pages 309–318. Springer, 2001.
3. J. Balasch, S. Faust, B. Gierlichs, and I. Verbauwhede. Theory and practice of a leakage resilient masking scheme – extended version –. Cryptology ePrint Archive, 2012. http://eprint.iacr.org/.
4. E. Brier, C. Clavier, and F. Olivier. Correlation power analysis with a leakage model. In M. Joye and J.-J. Quisquater, editors, *Proceedings of CHES 2004*, volume 3156 of *LNCS*, pages 16–29. Springer, 2004.
5. S. Chari, C. S. Jutla, J. R. Rao, and P. Rohatgi. Towards sound approaches to counteract power-analysis attacks. In M. J. Wiener, editor, *Proceedings of CRYPTO 1999*, volume 1666 of *LNCS*, pages 398–412. Springer, 1999.
6. S. Chari, J. R. Rao, and P. Rohatgi. Template attacks. In B. S. Kaliski Jr., Çetin K. Koç, and C. Paar, editors, *Proceedings of CHES 2002*, volume 2523 of *LNCS*, pages 13–28. Springer, 2003.
7. B. Chor and O. Goldreich. Unbiased bits from sources of weak randomness and probabilistic communication complexity. *SIAM J. Comput.*, 17(2):230–261, 1988.
8. A. Corporation. ATmega128 data sheet.
9. S. Dziembowski and S. Faust. Leakage-resilient cryptography from the inner-product extractor. In D. Lee and X. Wang, editors, *Proceedings of ASIACRYPT 2011*, volume 7073 of *LNCS*, pages 702–721. Springer, 2011.
10. S. Dziembowski and S. Faust. Leakage-resilient circuits without computational assumptions. In R. Cramer, editor, *Proceedings of TCC 2012*, volume 7194 of *LNCS*, pages 230–247. Springer, 2012.
11. S. Faust, T. Rabin, L. Reyzin, E. Tromer, and V. Vaikuntanathan. Protecting circuits from leakage: the computationally-bounded and noisy cases. In H. Gilbert, editor, *Proceedings of EUROCRYPT 2010*, volume 6110 of *LNCS*, pages 135–156. Springer, 2010.
12. G. Fumaroli, A. Martinelli, E. Prouff, and M. Rivain. Affine Masking against Higher-Order Side Channel Analysis. In A. Biryukov, G. Gong, and D. R. Stinson, editors, *Proceedings of SAC 2010*, volume 6544 of *LNCS*, pages 262–280. Springer, 2011.

13. K. Gandolfi, C. Mourtel, and F. Olivier. Electromagnetic analysis: Concrete results. In Çetin K. Koç, D. Naccache, and C. Paar, editors, *Proceedings of CHES 2001*, volume 2162 of *LNCS*, pages 251–261. Springer, 2001.
14. B. Gierlichs, L. Batina, P. Tuyls, and B. Preneel. Mutual information analysis – a generic side-channel distinguisher. In E. Oswald and P. Rohatgi, editors, *Proceedings of CHES 2008*, volume 5154 of *LNCS*, pages 426–442. Springer, 2008.
15. S. Goldwasser and G. N. Rothblum. Securing computation against continuous leakage. In T. Rabin, editor, *Proceedings of CRYPTO 2010*, volume 6223 of *LNCS*, pages 59–79. Springer, 2010.
16. S. Goldwasser and G. N. Rothblum. How to compute in the presence of leakage. *Electronic Colloquium on Computational Complexity (ECCC)*, 19:10, 2012.
17. J. D. Golic and C. Tymen. Multiplicative masking and power analysis of AES. In B. S. K. Jr., Çetin Kaya Koç, and C. Paar, editors, *Proceedings of CHES 2002*, volume 2523 of *LNCS*, pages 198–212. Springer, 2003.
18. L. Goubin and A. Martinelli. Protecting AES with Shamir's secret sharing scheme. In B. Preneel and T. Takagi, editors, *Proceedings of CHES 2011*, volume 6917 of *LNCS*, pages 79–94. Springer, 2011.
19. L. Goubin and J. Patarin. DES and differential power analysis the "duplication" method. In Çetin K. Koç and C. Paar, editors, *Proceedings of CHES 1999*, volume 1717 of *LNCS*, pages 158–172. Springer, 1999.
20. Y. Ishai, A. Sahai, and D. Wagner. Private circuits: Securing hardware against probing attacks. In D. Boneh, editor, *Proceedings of CRYPTO 2003*, volume 2729 of *LNCS*, pages 463–481. Springer, 2003.
21. A. Juma and Y. Vahlis. Protecting cryptographic keys against continual leakage. In T. Rabin, editor, *Proceedings of CRYPTO 2010*, volume 6223 of *LNCS*, pages 41–58. Springer, 2010.
22. P. C. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In N. Koblitz, editor, *Proceedings of CRYPTO 1996*, volume 1109 of *LNCS*, pages 104–113. Springer, 1996.
23. P. C. Kocher, J. Jaffe, and B. Jun. Differential power analysis. In M. J. Wiener, editor, *Proceedings of CRYPTO 1999*, volume 1666 of *LNCS*, pages 388–397. Springer, 1999.
24. E. Prouff and T. Roche. Higher-Order Glitches Free Implementation of the AES Using Secure Multi-party Computation Protocols. In B. Preneel and T. Takagi, editors, *Proceedings of CHES 2011*, volume 6917 of *LNCS*, pages 63–78. Springer, 2011.
25. J.-J. Quisquater and D. Samyde. Electromagnetic analysis (EMA): Measures and counter-measures for smart cards. In I. Attali and T. P. Jensen, editors, *Proceedings of E-smart 2001*, volume 2140 of *LNCS*, pages 200–210. Springer, 2001.
26. A. Rao. An exposition of bourgain's 2-source extractor. *Electronic Colloquium on Computational Complexity (ECCC)*, 14(034), 2007.
27. M. Rivain and E. Prouff. Provably Secure Higher-Order Masking of AES. In S. Mangard and F.-X. Standaert, editors, *Proceedings of CHES 2010*, volume 6225 of *LNCS*, pages 413–427. Springer, 2010.
28. F.-X. Standaert, T. Malkin, and M. Yung. A unified framework for the analysis of side-channel key recovery attacks. In A. Joux, editor, *Proceedings of EUROCRYPT 2009*, volume 5479 of *LNCS*, pages 443–461. Springer, 2009.
29. F.-X. Standaert, N. Veyrat-Charvillon, E. Oswald, B. Gierlichs, M. Medwed, M. Kasper, and S. Mangard. The world is not enough: Another look on second-order DPA. In M. Abe, editor, *Proceedings of ASIACRYPT 2010*, volume 6477 of *LNCS*, pages 112–129. Springer, 2010.