Higher Order Masking of Look-Up Tables

Jean-Sébastien Coron

University of Luxembourg, Luxembourg
jean-sebastien.coron@uni.lu

Abstract. We describe a new algorithm for masking look-up tables of block-ciphers at any order, as a countermeasure against side-channel attacks. Our technique is a generalization of the classical randomized table countermeasure against first-order attacks. We prove the security of our new algorithm against t-th order attacks in the usual Ishai-Sahai-Wagner model from Crypto 2003; we also improve the bound on the number of shares from $n \geq 4t+1$ to $n \geq 2t+1$ for an adversary who can adaptively move its probes between successive executions.

Our algorithm has the same time complexity $\mathcal{O}(n^2)$ as the Rivain-Prouff algorithm for AES, and its extension by Carlet *et al.* to any look-up table. In practice for AES our algorithm is less efficient than Rivain-Prouff, which can take advantage of the special algebraic structure of the AES Sbox; however for DES our algorithm performs slightly better.

1 Introduction

Side-Channel Attacks. An implementation of a cryptographic algorithm on some concrete device, such as a PC or a smart-card, can leak additional information to an attacker through the device power consumption or electro-magnetic emanations, enabling efficient key-recovery attacks. One of the most powerful attack is the Differential Power Analysis (DPA) [KJJ99]; it consists in recovering the secret-key by performing a statistical analysis of the power consumption of the electronic device, for several executions of a cryptographic algorithm. Another powerful class of attack are template attacks [CRR02]; a template is a precise model for the noise and expected signal for all possible values of part of the key; the attack is then carried out iteratively to recover successive parts of the key.

Random Masking. A well-known countermeasure against side-channel attacks consists in masking all internal variables with a random r, as first suggested in [CJRR99]. Any internal variable x is first masked by computing $x' = x \oplus r$, and the masked variable x' and the mask r are then processed separately. An attacker trying to analyze the power consumption at a single point will obtain only random values; therefore, the implementation will be secure against first-order DPA. However, a first-order masking can be broken in practice by a second-order side channel attack, in which the attacker combines information from two leakage points [Mes00]; however such attack usually requires a larger

P.Q. Nguyen and E. Oswald (Eds.): EUROCRYPT 2014, LNCS 8441, pp. 441–458, 2014.

number of power consumption curves, which can be unfeasible in practice if the number of executions is limited (for example, by using a counter). For AES many countermeasures based on random masking have been described, see for example [HOM06].

More generally, one can split any variable x into n boolean shares by letting $x = x_1 \oplus \cdots \oplus x_n$ as in a secret-sharing scheme [Sha79]. The shares x_i must then be processed separately without leaking information about the original variable x. Most block-ciphers (such as AES or DES) alternate several rounds, each containing one linear transformation (or more), and a non-linear transformation. A linear function $y = f(x)$ is easy to compute when x is shared as $x = x_1 \oplus \cdots \oplus x_n$, as it suffices to compute $y_i = f(x_i)$ separately for every i. However securely computing a non-linear function $y = S(x)$ with shares is more difficult and is the subject of this paper.

The Ishai-Sahai-Wagner Private Circuit. The theoretical study of securing circuits against an adversary who can probe its wires was initiated by Ishai, Sahai and Wagner in [ISW03]. The goal is to protect a cryptographic implementation against side-channel attacks in a provable way. The authors consider an adversary who can probe at most t wires of the circuit. They showed how to transform any boolean circuit C of size $|C|$ into a circuit of size $\mathcal{O}(|C| \cdot t^2)$ that is perfectly secure against such adversary.

The Ishai-Sahai-Wagner (ISW) model is relevant even in the context of power attacks. Namely the number of probes in the circuit corresponds to the attack order in a high-order DPA. More precisely, if a circuit is perfectly secure against t probes, then combining t power consumption points as in a t-th order DPA will reveal no information to the adversary. To obtain useful information about the key the adversary will have to perform an attack of order at least $t + 1$. The soundness of higher-order masking in the context of power attacks was first demonstrated by Chari *et al.* in [CJRR99], who showed that in a realistic leakage model the number of acquisitions to recover the key grows exponentially with the number of shares. Their analysis was recently extended by Prouff and Rivain in [PR13]. The authors proved that the information obtained by observing the *entire* leakage of an execution (instead of the leakage of the n shares of a given variable) can be made negligible in the masking order. This shows that the number of shares n is a sound security parameter for protecting an implementation against side-channel attacks.

To protect against an adversary with at most t probes, the ISW approach consists in secret-sharing every variable x into n shares x_i where $n = 2t + 1$, that is $x = x_1 \oplus x_2 \oplus \cdots \oplus x_n$ where $x_2, \ldots, x_n$ are uniformly and independently distributed bits. An adversary probing at most $n - 1$ variables clearly does not learn any information about x. Processing a NOT gate is straightforward since $\bar{x} = \bar{x}_1 \oplus x_2 \oplus \cdots \oplus x_n$; therefore it suffices to invert the first share x_1. To process an AND gate $z = xy$, one writes:

$$z = xy = \left(\bigoplus_{i=1}^{n} x_i \right) \cdot \left(\bigoplus_{i=1}^{n} y_i \right) = \bigoplus_{1 \leq i,j \leq n} x_i y_j \tag{1}$$

and the cross-products $x_i y_j$ are processed and recombined without leaking information about the original inputs x and y. More precisely for each $1 \leq i < j \leq n$ one generates random bits $r_{i,j}$ and computes $r_{j,i} = (r_{i,j} \oplus x_i y_j) \oplus x_j y_i$; the n shares z_i of $z = xy$ are then computed as $z_i = x_i y_i \oplus \bigoplus_{j \neq i} r_{i,j}$. Since there are n^2 such cross-products, every AND gate of the circuit is expanded to $\mathcal{O}(n^2) = \mathcal{O}(t^2)$ new gates in the circuit.

The authors also describe a very convenient framework for proving the security against any set of t probes. Namely proving the security of a countermeasure against first-order attacks ($t = 1$) is usually straightforward, as it suffices to check that every internal variable has the uniform distribution (or at least a distribution independent from the secret-key). Such approach can be extended to second-order attacks by considering pairs of internal variables (as in [RDP08]); however it becomes clearly unfeasible for larger values of t, as the number of t-uples to consider would grow exponentially with t. Alternatively the ISW framework is simulation based: the authors prove the security of their construction against a adversary with at most t probes by showing that any set of t probes can be perfectly simulated without the knowledge of the original input variables (such as x, y in the AND gate $z = xy$). In [ISW03] this is done by iteratively generating a subset I of indices of the input shares that are sufficient to simulate the t probes; then if $|I| < n$ the corresponding input shares can be perfectly simulated without knowing the original input variable, simply by generating independently and uniformly distributed bits. In the ISW construction every probe adds at most two indices in I, so we get $|I| \leq 2t$ and therefore $n \geq 2t+1$ is sufficient to achieve perfect secrecy against a t-limited adversary. A nice property of the ISW framework is that the technique easily extends from a single gate to the full circuit: it suffices to maintain a global subset of indices I that is iteratively constructed from the t probes as in a single gate.

The Rivain-Prouff Countermeasure. The Rivain-Prouff countermeasure [RP10] was the first provably secure higher-order masking scheme for the AES block-cipher. Namely, all previous masking schemes were secure against first-order or second-order attacks only. The classical randomized table countermeasure [CJRR99] is secure against first-order attacks only. The Schramm and Paar countermeasure [SP06] was designed to be secure at any order n, but an attack of order 3 was shown in [CPR07]. An alternative countermeasure based on table recomputation and provably secure against second-order attacks was described in [RDP08], but no extension to any order is known. The Rivain-Prouff countermeasure was therefore the first masking scheme for AES secure for any order $t \geq 3$.

The Rivain-Prouff countermeasure is an adaptation of the previous ISW construction to software implementations, working in the AES finite field $\mathbb{F}_{2^8}$ instead of $\mathbb{F}_2$. Namely the non-linear part of the AES Sbox can be written as $S(x) = x^{254}$ over $\mathbb{F}_{2^8}$, and as shown in [RP10] such monomial can be evaluated with only 4 non-linear multiplications (and a few linear squarings). These 4 multiplications can be evaluated with n-shared input using the previous technique based on Equation (1), by working over the field $\mathbb{F}_{2^8}$ instead of $\mathbb{F}_2$. In order to achieve

resistance against an attack of order t, the Rivain-Prouff algorithm also requires $n \geq 2t+1$ shares.

The Rivain-Prouff countermeasure was later extended by Carlet *et al.* to any look-up table [CGP+12]. Namely using Lagrange interpolation any Sbox with k-bit input can be written as a polynomial

$$S(x) = \sum_{i=0}^{2^k-1} \alpha_i \cdot x^i$$

over $\mathbb{F}_{2^k}$, for constant coefficients $\alpha_i \in \mathbb{F}_{2^k}$. The polynomial can then be evaluated with n-shared multiplications as in the Rivain-Prouff countermeasure. The authors of [CGP+12] describe two techniques for optimizing the evaluation of $S(x)$ by minimizing the number of non-linear multiplications: the cyclotomic method and the parity-split method; the later method is asymptotically faster and requires $\mathcal{O}(2^{k/2})$ multiplications. Therefore the Carlet *et al.* countermeasure with n shares has time complexity $\mathcal{O}(2^{k/2} \cdot n^2)$, where $n \geq 2t+1$ to ensure resistance against t-th order attacks.

Extending the Randomized Table Countermeasure. Our new countermeasure is completely different from the Rivain-Prouff countermeasure and its extension by Carlet *et al.*. Namely it is essentially based on table recomputations and does not use multiplications over $\mathbb{F}_{2^k}$. To illustrate our technique we start with the classical randomized table countermeasure, secure against first order attacks only, as first suggested in [CJRR99]. The Sbox table $S(u)$ with k-bit input is first randomized in RAM by letting

$$T(u) = S(u \oplus r) \oplus s$$

for all $u \in \{0,1\}^k$, where $r \in \{0,1\}^k$ is the input mask and $s \in \{0,1\}^k$ is the output mask.[1] To evaluate $S(x)$ from the masked value $x' = x \oplus r$, it suffices to compute $y' = T(x')$, as we get $y' = T(x') = S(x' \oplus r) \oplus s = S(x) \oplus s$; this shows that y' is indeed a masked value for $S(x)$. In other words the randomized table countermeasure consists in first re-computing in RAM a temporary table with inputs shifted by r and with masked outputs, so that later it can be evaluated on a masked value $x' = x \oplus r$ to obtain a masked output.

A natural generalization at any order n would be as follows: given as input $x = x_1 \oplus \cdots \oplus x_n$ we would start with a randomized table with inputs shifted by x_1 only, and with $n-1$ output masks; then we would incrementally shift the full table by x_2 and so on until x_{n-1}, at which point the table could be evaluated at x_n. More precisely one would initially define the randomized table

$$T(u) = S(u \oplus x_1) \oplus s_2 \oplus \cdots \oplus s_n$$

where $s_2, \ldots, s_n$ are the output masks, and then progressively shift the randomized table by letting $T(u) \leftarrow T(u \oplus x_i)$ for all u, iteratively from x_2 until x_{n-1}.

[1] One can also take $s = r$. For simplicity we first assume that the Sbox has both k-bit input and k-bit output.

Eventually the table would have all its inputs shifted by $x_1 \oplus \cdots \oplus x_{n-1}$, so as previously one could evaluate $y' = T(x_n)$ and obtain $S(x)$ masked by $s_2, \ldots, s_n$.

What we have described above is essentially the Schramm and Paar countermeasure [SP06]. However as shown in [CPR07] this is insecure. Namely consider the table $T(u)$ after the last shift by x_{n-1}; at this point we have $T(u) = S(u \oplus x_1 \oplus \cdots \oplus x_{n-1}) \oplus s_2 \oplus \cdots \oplus s_n$ for all u. Now assume that we can probe $T(0)$ and $T(1)$; we can then compute $T(0) \oplus T(1) = S(x_1 \oplus \cdots \oplus x_{n-1}) \oplus S(1 \oplus x_1 \oplus \cdots \oplus x_{n-1})$, which only depends on $x_1 \oplus \cdots \oplus x_{n-1}$; therefore it suffices to additionally probe x_n to leak information about $x = x_1 \oplus \cdots \oplus x_{n-1} \oplus x_n$; this gives an attack of order 3 only for any value of n; therefore the countermeasure can only be secure against second-order attacks.

The main issue with the previous countermeasure is that the *same* masks $s_2, \ldots, s_n$ were used to mask all the $S(u)$ entries, so one can exclusive-or any two lines of the randomized table and remove all the output masks. A natural fix is to use *different* masks for every line $S(u)$ of the table, so one would write initially:

$$T(u) = S(u \oplus x_1) \oplus s_{u,2} \oplus \cdots \oplus s_{u,n}$$

for all $u \in \{0,1\}^k$, and as previously one would iteratively shift the table by $x_2, \ldots, x_{n-1}$, and also the masks $s_{u,i}$ separately for each i. The previous attack is thwarted because the lines of $S(u)$ are now masked with different set of masks. Eventually one would read $T(x_n)$, which would give $S(x)$ masked by $s_{x_n,2}, \ldots, s_{x_n,n}$.

Our Table-Recomputation Countermeasure. Our new countermeasure is based on using independent masks as above, with additionally a refresh of the masks between every successive shifts of the input. Since the above output masks $s_{u,j}$ are now different for all lines u of the table, we actually have a set of n randomized tables, as opposed to a single randomized table in the original Schramm and Paar countermeasure. Perhaps more conveniently one can view every line u of our randomized table as a n-dimensional vector of elements in $\{0,1\}^k$, and write for all inputs $u \in \{0,1\}^k$:

$$T(u) = (s_{u,1},\ s_{u,2}, \ldots,\ s_{u,n})$$

where initially each vector $T(u)$ is a n-boolean sharing of the value $S(u \oplus x_1)$. The vectors $T(u)$ of our randomized table are then progressively shifted for all $u \in \{0,1\}^k$, first by x_2 and so on until x_{n-1}, as in the original Schramm and Paar countermeasure. Eventually the evaluation of $T(x_n)$ gives a vector of n output shares that corresponds to $S(x)$.

To refresh the masks between successive shifts we can generate a random n-sharing of 0, that is $a_1, \ldots, a_n \in \{0,1\}^k$ such that $a_1 \oplus \cdots \oplus a_n = 0$ and we xor the vector $T(u)$ with $(a_1, \ldots, a_n)$, independently for every u. More concretely one can use the RefreshMasks procedure from [RP10], which consists given $y = y_1 \oplus y_2 \oplus \cdots \oplus y_n$ in xoring both y_1 and y_i with $tmp \leftarrow \{0,1\}^k$, iteratively from $i = 2$ to n. In summary our new countermeasure is essentially the Schramm and Paar countermeasure with independent output masks for every line of the

Sbox table, and with mask refreshing after every shift of the table; we provide a full description in Section 3.1.[2]

We show that our new countermeasure is secure against any attack of order t in the ISW model, with at least $n = 2t + 1$ shares. The proof works as follows. Assume that there are at most $n - 3$ probes; then it must be the case that at least one of the $n - 2$ shifts of the table by x_i and subsequent mask refreshings are not probed at all. Since the corresponding mask refreshings are not probed, we can perfectly simulate any subset of $n-1$ shares at the output of those mask refreshings. Therefore we can perfectly simulate all the internal variables up to the x_{i-1} shift by knowing $x_1, \ldots, x_{i-1}$, and any subset of $n - 1$ shares after the x_i shift by knowing $x_{i+1}, \ldots, x_n$. Since the knowledge of x_i is not needed in the simulation, the full simulation can be performed without knowing the original input x, which proves the security of our countermeasure.[3]

Note that it does not matter how the mask refreshing is performed; the only required property is that after a (non-probed) mask refreshing any subset of $n - 1$ shares among the n shares have independent and uniform distribution; such property is clearly satisfied by the RefreshMasks procedure from [RP10] recalled above. We stress that in the argument above only the mask refreshings corresponding to one of the x_i shift are assumed to be non-probed (which must be the case because of the limited number of probes), and that all the remaining mask refreshings can be freely probed by the adversary, and correctly simulated.

The previous argument only applies when the Sbox evaluation is considered in isolation. When combined with other operations (in particular Xor gates), we must actually apply the same technique (with the I subset) as in [ISW03], and we obtain the same bound $n \geq 2t + 1$ for the number of shares, as in the Rivain-Prouff countermeasure.

Asymptotic Complexities. With respect to the number n of shares, our new countermeasure has the same time complexity $\mathcal{O}(n^2)$ as the Rivain-Prouff and Carlet *et al.* countermeasures. However for a k-bit input table, our basic countermeasure has complexity $\mathcal{O}(2^k \cdot n^2)$ whereas the Carlet *et al.* countermeasure has complexity $\mathcal{O}(2^{k/2} \cdot n^2)$, which is better for large k.

In Section 3.3 we describe a variant of our countermeasure for processors with large register size, with the same time complexity $\mathcal{O}(2^{k/2} \cdot n^2)$ as the Carlet *et al.* countermeasure, using a similar approach as in [RDP08]. Our variant consists in packing multiple Sbox outputs into a single register, and performing the table

[2] The mask refreshing is necessary to prevent a different attack. Assume that we probe the first component of $T(0)$ for the initial configuration of the table $T(u)$, and we again probe the first component of $T(0)$ when the table $T(u)$ has eventually been shifted by $x_2 \oplus \cdots \oplus x_{n-1}$. If $x_2 \oplus \cdots \oplus x_{n-1} = 0$ then without mask refreshing those two probed values must be the same; this leaks information about $x_2 \oplus \cdots \oplus x_{n-1}$, and therefore it suffices to additionally probe x_1 and x_n to have an attack of order 4 for any n.

[3] The previous argument could be extended to the optimal number of probes $n - 1$ by considering the initial sharing of $S(u \oplus x_1)$ and by adding a final mask refreshing after the evaluation of $T(x_n)$, as actually done in Section 3.1.

recomputations at the register level first. For example for DES we can pack 8 output 4-bit nibbles into a single 32-bit register; in that case the running time is divided by a factor 8. We stress that our variant does *not* consist in putting multiple shares of the same variable into a single register, as reading such register would reveal many shares at once, and thereby decrease the number of probes t required to break the countermeasure.

Note that our countermeasure has memory complexity $\mathcal{O}(n)$, instead of $\mathcal{O}(n^2)$ for the Rivain-Prouff countermeasure as described in [RP10]. However we show in the full version of this paper [Cor13a] that the memory complexity of the Rivain-Prouff countermeasure can be reduced to $\mathcal{O}(n)$, simply by computing the variables in a different order; this extends to the Carlet *et al.* countermeasure. We summarize in Table 1 the complexity of the two countermeasures.

Table 1. Time, memory, and number of random bits used, for a k-bit input table masked with n shares and secure against any attack at order t, with $2t + 1 \leq n$

Countermeasure	Time	Memory	Randomness
Carlet *et al.* [CGP+12]	$\mathcal{O}(2^{k/2} \cdot n^2)$	$\mathcal{O}(2^{k/2} \cdot n)$	$\mathcal{O}(k2^{k/2} \cdot n^2)$
Table Recomputation	$\mathcal{O}(2^k \cdot n^2)$	$\mathcal{O}(2^k \cdot n)$	$\mathcal{O}(k2^k \cdot n^2)$
Table Recomputation (large register)	$\mathcal{O}(2^{k/2} \cdot n^2)$	$\mathcal{O}(2^k \cdot n)$	$\mathcal{O}(k2^k \cdot n^2)$

Protecting a Full Block-Cipher. We show how to integrate our countermeasure into the protection of a full block-cipher against t-th order attacks. We consider two models of security. In the *restricted* model, the adversary always probes the same t intermediate variables for different executions of the block-cipher. In the *full* model the adversary can change the position of its probes adaptively between successive executions; this is essentially the ISW model for stateful circuits.

The restricted model is relevant in practice because in a t-th order DPA attack, the statistical analysis is performed on a fixed set of t intermediate variables for all executions. In both models the key is initially provided in shared form as input, with n shares. In the full model it is necessary to re-randomize the shares of the key between executions, since otherwise the adversary could recover the key by moving its probes between successive executions; obviously this re-randomization of shares must also be secure against a t-th order attack.

We show that $n \geq 2t + 1$ is sufficient to achieve security against t-th order attacks in both models. In particular, this improves the bound $n \geq 4t + 1$ from [ISW03] for stateful circuits.[4] We get an improved bound because for every execution we use both an initial re-randomization of the key shares (before they are used to evaluate the block-cipher) and a final re-randomization of the key shares (before they are given as input to the next execution), whereas in [ISW03] only a final re-randomization was used. With the same technique we can obtain

[4] In [ISW03] the bounds are $n \geq 2t+1$ for stateless circuits and $n \geq 4t+1$ for stateful circuits.

the same improved bound in the full model for the Rivain-Prouff countermeasure and its extension by Carlet *et al.*.

Note that in the full model the bound $n \geq 2t+1$ is actually optimal. Namely as noted in [ISW03] the adversary can probe t of the key shares at the end of one execution and then another t of the key shares at the beginning of the next execution, hence a total of $2t$ key shares of the same n-sharing of the secret-key. Hence $n \geq 2t+1$ shares are necessary.[5]

Practical Implementation. Finally we have performed a practical implementation of our new countermeasure for both AES and DES, using a 32-bit architecture so that we could apply our large register variant. For comparison we have also implemented the Rivain-Prouff countermeasure for AES and the Carlet *et al.* countermeasure for DES; for the latter we have used the technique from [RV13], in which the evaluation of a DES Sbox requires only 7 non-linear multiplications. We summarize the result of our practical implementations in Section 5. We obtain that in practice for AES our algorithm is less efficient than Rivain-Prouff, which can take advantage of the special algebraic structure of the AES Sbox; however for DES our algorithm performs slightly better. Our implementation is publicly available [Cor13b].

2 Definitions

In this section we first recall the Ishai-Sahai-Wagner (ISW) framework [ISW03] for proving the resistance of circuits against probing attacks. In [RP10] Rivain and Prouff describe an adaptation of the ISW model for software implementations. We follow the same approach and describe two security models: a restricted model in which the adversary always probes the same t intermediate variables (which is essentially the model considered in [RP10]), and a full model in which the t probes can be changed adaptively between executions (which is essentially the ISW model for stateful circuits).

2.1 The Ishai-Sahai-Wagner Framework

Privacy for Stateless Circuits. A *stateless* circuit over $\mathbb{F}_2$ is a directed acyclic graph whose sources are labeled with the input variables, sinks are labeled with output variables, and internal vertices stand for function gates. A stateless circuit can be *randomized*, if it additionally contains *random gates*; every such gate has no input, and its only output at each invocation of the circuit is a uniform random bit.

A *t-limited adversary* can probe up to t wires in the circuit, and has unlimited computational power. A stateless circuit C is called (perfectly) secure against such adversary, if the distribution of the probes can be efficiently and perfectly simulated, without access to the internal wires of C. For stateless circuits one

[5] At least this holds for the shares of the secret-key. It could be that $n = t+1$ shares are sufficient for the other variables.

assumes that the inputs and outputs of the circuit must remain private. For example in a block-cipher the input key must remain private. To prevent the adversary for learning the inputs and outputs one uses an *input encoder* I and an *output decoder* O, whose internal wires cannot be probed. Additionally, the inputs of I and the outputs of O are also assumed to be protected against probing. However, the outputs of I and the inputs to O can be probed. Finally the *t-private stateless transformer* (T, I, O) maps a stateless circuit C into a (randomized) stateless circuit C', such that C' is secure against the t-limited adversary, and $O \circ C' \circ I$ has the same input-output functionality as C.

The ISW Construction. As recalled in introduction in [ISW03] the authors showed how to transform any boolean circuit C of size $|C|$ into a circuit of size $\mathcal{O}(|C| \cdot t^2)$ that is perfectly secure against a t-limited adversary. The approach in [ISW03] consists in secret-sharing every variable x into n shares x_i where $n = 2t+1$, that is $x = x_1 \oplus x_2 \oplus \cdots \oplus x_n$ where $x_2, \ldots, x_n$ are uniformly and independently distributed bits. The authors prove the security of their construction against a t-limited adversary by showing that any set of t probes can be perfectly simulated without knowing the internal wires of the circuit, for $n \geq 2t + 1$.

Extension to Stateful Circuits. The ISW model and construction can be extended to stateful circuits, that is a circuit containing memory cells. In the stateful model the inputs and outputs are known to the attacker and one does not use the input encoder I and output decoder O. For a block-cipher the secret key sk would be originally incorporated in a shared form sk_i inside the memory cells of the circuit; the key shares sk_i would be re-randomized after each invocation of the circuit. The authors show that for stateful circuits $n \geq 4t + 1$ shares are sufficient for security against a t-limited adversary; we refer to [ISW03] for more details.

2.2 Security Model for Software Implementations

In [RP10] Rivain and Prouff describe an adaptation of the ISW model for software implementations of encryption algorithms. They consider a *randomized encryption algorithm* $\mathcal{E}$ taking as input a plaintext m and a randomly shared secret-key sk and outputting a ciphertext c, with additional access to a random number generator. More precisely the secret-key sk is assumed to be split into n shares $sk_1, \ldots, sk_n$ such that $sk = sk_1 \oplus \cdots \oplus sk_n$ and any $(n-1)$-uple of sk_i's is uniformly and independently distributed. Instead of considering the internal wires of a circuit, they consider the intermediate variables of the software implementation. This approach seems well suited for proving the security of our countermeasure; in principle one could write our countermeasure with randomized table as a stateful circuit and work in the ISW model for stateful circuits, but that would be less convenient.

In the following we describe two different models of security. In the *restricted* model the adversary provides a message m as input and receives $c = \mathcal{E}_{sk}(m)$ as output. The adversary can run $\mathcal{E}_{sk}$ several times, but she always obtain the same

set of t intermediate variables that she can freely choose before the first execution. In the *full* model, the adversary can adaptively change the set of t intermediate variables between executions. In both models the shares sk_i of the secret-key sk are initially incorporated in the memory cells of the block-cipher implementation. We say that a randomized encryption algorithm is secure against t-th order attack (in the restricted or full model) if the distribution of any t intermediate variables can be perfectly simulated without the knowledge of the secret-key sk. This implies that anything an adversary $\mathcal{A}$ can do from the knowledge of t intermediate variables, another adversary $\mathcal{A}'$ can do the same without the knowledge of those t intermediate variables. Note that since $\mathcal{A}$ initially provides the message m and receives the ciphertext c, we can consider that both m and c are public and given to the simulator.

Note that in the full model it is necessary to re-randomize in memory the shares sk_i of the key, since otherwise the adversary could recover sk by moving its probes between successive executions; obviously this re-randomization of shares must also be secure against a t-th order attack.

3 Our New Algorithm

3.1 Description

In this section we describe our new algorithm for computing $y = S(x)$ where

$$S : \{0,1\}^k \to \{0,1\}^{k'}$$

is a look-up table with k-bit input and k'-bit output. Our new algorithm takes as input $x_1, \ldots, x_n$ such that $x = x_1 \oplus \cdots \oplus x_n$ and must output $y_1, \ldots, y_n$ such that $y = S(x) = y_1 \oplus \cdots \oplus y_n$, without leaking information about x. Our algorithm uses two temporary tables T and T' in RAM; both have k-bit input and a vector of n elements of k'-bit as output, namely

$$T, T' : \{0,1\}^k \to (\{0,1\}^{k'})^n$$

Given a vector $\boldsymbol{v} = (v_1, \ldots, v_n)$ of n elements, we write $\oplus(\boldsymbol{v}) = v_1 \oplus \cdots \oplus v_n$. We denote by $T(u)[j]$ and $T'(u)[j]$ the j-th component of the vectors $T(u)$ and $T'(u)$ respectively, for $1 \leq j \leq n$. In practice the two tables can be implemented as 2-dimensional arrays of elements in $\{0,1\}^{k'}$. We use the same RefreshMasks procedure as in [RP10].

Correctness. It is easy to verify the correctness of Algorithm 1. We proceed by induction. Assume that at Line 4 for index i we have for all inputs $u \in \{0,1\}^k$:

$$\oplus\big(T(u)\big) = S(u \oplus x_1 \oplus \cdots \oplus x_{i-1}) \tag{2}$$

The assumption clearly holds for $i = 0$, since initially we have $\oplus\big(T(u)\big) = S(u)$ for all inputs $u \in \{0,1\}^k$. Assuming that (2) holds for index i at Line 4, after the shifts performed at Line 6 we have for all inputs $u \in \{0,1\}^k$,

$$\oplus\big(T'(u)\big) = \oplus\big(T(u \oplus x_i)\big) = S\big((u \oplus x_i) \oplus x_1 \oplus \cdots \oplus x_{i-1}\big) = S(u \oplus x_1 \oplus \cdots \oplus x_i)$$

Algorithm 1. Masked computation of $y = S(x)$

Input: $x_1, \ldots, x_n$ such that $x = x_1 \oplus \cdots \oplus x_n$
Output: $y_1, \ldots, y_n$ such that $y = S(x) = y_1 \oplus \cdots \oplus y_n$
1: **for all** $u \in \{0,1\}^k$ **do**
2: $T(u) \leftarrow \big(S(u), 0, \ldots, 0\big) \in (\{0,1\}^{k'})^n$ ▷ $\oplus\big(T(u)\big) = S(u)$
3: **end for**
4: **for** $i = 1$ to $n-1$ **do**
5: **for all** $u \in \{0,1\}^k$ **do**
6: **for** $j = 1$ **to** n **do** $T'(u)[j] \leftarrow T(u \oplus x_i)[j]$ ▷ $T'(u) \leftarrow T(u \oplus x_i)$
7: **end for**
8: **for all** $u \in \{0,1\}^k$ **do**
9: $T(u) \leftarrow \mathsf{RefreshMasks}\big(T'(u)\big)$ ▷ $\oplus\big(T(u)\big) = S(u \oplus x_1 \oplus \cdots \oplus x_i)$
10: **end for**
11: **end for** ▷ $\oplus\big(T(u)\big) = S(u \oplus x_1 \oplus \cdots \oplus x_{n-1})$ for all $u \in \{0,1\}^k$.
12: $(y_1, \ldots, y_n) \leftarrow \mathsf{RefreshMasks}\big(T(x_n)\big)$ ▷ $\oplus\big(T(x_n)\big) = S(x)$
13: **return** $y_1, \ldots, y_n$

Algorithm 2. RefreshMasks

Input: $z_1, \ldots, z_n$ such that $z = z_1 \oplus \cdots \oplus z_n$
Output: $z_1, \ldots, z_n$ such that $z = z_1 \oplus \cdots \oplus z_n$
1: **for** $j = 2$ to n **do**
2: $tmp \leftarrow \{0,1\}^{k'}$
3: $z_1 \leftarrow z_1 \oplus tmp$
4: $z_j \leftarrow z_j \oplus tmp$
5: **end for**
6: **return** $z_1, \ldots, z_n$

and therefore the assumption holds at Step $i+1$. At the end of the loop we have therefore

$$\oplus\big(T(u)\big) = S(u \oplus x_1 \oplus \cdots \oplus x_{n-1})$$

for all $u \in \{0,1\}^k$, and then $\oplus\big(T(x_n)\big) = S(x_n \oplus x_1 \oplus \cdots \oplus x_{n-1}) = S(x)$ which gives $y_1 \oplus \cdots \oplus y_n = S(x)$ as required. This proves the correctness of Algorithm 1.

Remark 1. A NAND gate can be implemented as a 2-bit input, 1-bit output look-up table; therefore Algorithm 1 can be used to protect any circuit, with the same complexity $\mathcal{O}(n^2)$ in the number of shares n as the ISW construction.

3.2 Security Proof

The following Lemma proves the security of our countermeasure against t-th order attacks, for any t such that $2t + 1 \leq n$. The proof is done in the ISW model [ISW03]. Namely we show that from any given set of t probed intermediate variables, one can define a set $I \subset [1, n]$ with $|I| < n$ such that the knowledge of the input indices $x_{|I} := (x_i)_{i \in I}$ is sufficient to perfectly simulate those t intermediate variables. Then since $|I| < n$ those input shares can be perfectly

simulated without knowing the original input variable, simply by generating independently and uniformly distributed variables.

Lemma 1. *Let $(x_i)_{1\leq i\leq n}$ be the input shares of Algorithm 1 and let t be such that $2t < n$. For any set of t intermediate variables, there exists a subset $I \subset [1, n]$ of indices such that $|I| \leq 2t < n$ and the distribution of those t variables can be perfectly simulated from the shares $x_{|I}$. The output shares $y_{|I}$ can also be perfectly simulated from $x_{|I}$.*

Proof. Given a set of t intermediate variables $v_1, \ldots, v_t$ probed by the adversary, we construct a subset $I \subset [1, n]$ of indices such that the distribution of those t variables can be perfectly simulated from $x_{|I}$. We call *Part* i the computation performed within the main for loop for index i for $1 \leq i \leq n-1$, that is from Line 5 to Line 10 of Algorithm 1; similarly we call *Part* n the computation performed at Line 12. We do not consider the intermediate variables from Line 2, as they can be perfectly simulated without the knowledge of x.

The proof intuition is as follows. Every intermediate variable v_h is identified by its "line" index i corresponding to the Part in which it appears, with $1 \leq i \leq n$, and by its "column" index j corresponding to the j-th component of the vector in which it appears; for any such intermediate variable v_h both indices i and j are added to the subset I (except for x_i and the *tmp* variables within RefreshMasks for which only i is added). The crucial observation is the following: if $i \notin I$, then no intermediate variable was probed within Part i of Algorithm 1; in particular the *tmp* variables within the corresponding RefreshMasks were not probed. Therefore we can perfectly simulate the outputs of the RefreshMasks function which have "column" index $j \in I$, by generating uniform and independent elements in $\{0,1\}^{k'}$, as long as $|I| < n$. This means that for $i \notin I$ we can perfectly simulate all variables $T(u)[j]$ for $j \in I$ in Line 9. Considering now Part i for which $i \in I$, since we know x_i we can still perfectly simulate all intermediate variables with "column" index $j \in I$ (including also the *tmp* variables within RefreshMasks), which includes by definition of I all the intermediates variables v_h. Therefore all intermediate variables v_h can be perfectly simulated as long as $|I| < n$, which gives the condition $2t < n$.

Formally the procedure for constructing the set I is as follows:

1. We start with $I = \emptyset$.
2. For any intermediate variable v_h:
   (a) If $v_h = x_i$ or $v_h = u \oplus x_i$ at Line 6, then add i to I.
   (b) If $v_h = T(u \oplus x_i)[j]$ or $v_h = T'(u)[j]$ at Line 6 in Part i, then add both i and j to I.
   (c) If $v_h = T'(u)[j]$ or $v_h = T(u)[j]$ at Line 9 in Part i, then add both i and j to I.
   (d) If $v_h = tmp$ for any *tmp* within RefreshMasks in Part i (either at Line 9 or 12), then add i to I.
   (e) If $v_h = x_n$ at Line 12, then add n to I.
   (f) If $v_h = T(x_n)[j]$ or $v_h = y_j$ at Line 12, then add both n and j to I.

This terminates the description of the procedure for constructing the set I. Since any intermediate variable v_h adds at most two indices in I, we must have $|I| \leq 2t < n$.

We now show how to complete a perfect simulation of all intermediate variables v_h using only the values $x_{|I}$. We proceed by induction. Assume that at the beginning of Part i we can perfectly simulate all variables $T(u)[j]$ for all $j \in I$ and all $u \in \{0,1\}^k$. This holds for $i = 1$ since initially we have $T(u) = (S(u), 0, \ldots, 0)$ which does not depend on x.

We distinguish two cases. If $i \notin I$ then no *tmp* variable within the RefreshMasks in Part i has been probed. Therefore we can perfectly simulate all intermediate variables $T(u)[j]$ for $j \in I$ at the output of RefreshMasks at Line 9, or similarly all y_j for $j \in I$ at the output of RefreshMasks at Line 12 when $i = n$, as long as $|I| < n$. Formally this can be proven as follows. Let j^* be such that $j^* \notin I$. Since the internal variables of the RefreshMasks are not probed, we can redefine RefreshMasks where the randoms *tmp* are accumulated inside z_{j^*} instead of z_1. Since $j^* \notin I$ we have that z_{j^*} is never used in the computation of any variable v_h, and therefore every variables z_j for $j \in I$ is masked by a random *tmp* which is used only once. Therefore at the output of RefreshMasks the variables $T(u)[j]$ for $j \in I$ can be perfectly simulated for all $u \in \{0,1\}^k$, simply by generating uniform and independent values.

If $i \in I$ then knowing x_i we can perfectly simulate all intermediate variables with column index $j \in I$ in Part i. Namely our induction hypothesis states that at the beginning of Part i the variables $T(u)[j]$ for all $j \in J$ can already be perfectly simulated. Knowing x_i we can therefore propagate the simulation for all variables with column index j and perfectly simulate $T(u \oplus x_i)[j]$, $T'(u)[j]$ and the resulting $T(u)[j]$ at Line 9, and similarly the variables y_j at Line 12 if $i = n$; in particular the *tmp* variables within RefreshMasks are simulated exactly as in the RefreshMasks procedure.

Since in both cases we can perfectly simulate all intermediate variables $T(u)[j]$ for $j \in I$ at the end of Part i, the induction hypothesis holds for $i + 1$; therefore it holds for all $1 \leq i \leq n$. From the reasoning above we can therefore simulate all intermediate variables in Part i with column index j such that $i, j \in I$; by definition of I this includes all intermediate variables v_h, and all output shares $y_{|I}$; this proves Lemma 1. □

Remark 2. Although $n \geq 2t + 1$ shares are required for the security proof, our countermeasure seems heuristically secure with $n \geq t + 1$ shares only in the restricted model.

3.3 A Variant for Processors with Large Registers

With respect to the number n of shares, our new countermeasure has the same time complexity $\mathcal{O}(n^2)$ as the Rivain-Prouff and Carlet *et al.* countermeasures. However for a k-bit input table, our algorithm has complexity $\mathcal{O}(2^k \cdot n^2)$ whereas the Carlet *et al.* countermeasure has complexity $\mathcal{O}(2^{k/2} \cdot n^2)$ only.

In this section we describe a variant of our countermeasure with the same complexity as Carlet *et al.*, but for processors with large enough register size

ω bits, using a similar approach as in [RDP08, Section 3.3]. We assume that a read/write operation on such register takes unit time. As previously the goal is to compute $y = S(x)$ where

$$S : \{0,1\}^k \to \{0,1\}^{k'}$$

is a look-up table with k-bit input and k'-bit output.

Under the variant the k'-bit outputs of S are first packed into register words of $\omega = \ell \cdot k'$ bits, where ℓ is assumed to be a power of two. For example, for a DES Sbox with $k = 6$ input bits and $k' = 4$ output bits, on a $\omega = 32$ bits architecture we can pack $\ell = 8$ output 4-bit nibbles into a 32-bit word. Formally, we define a new Sbox S' with k_1-bit input and $\omega = \ell \cdot k'$ bits output with

$$S'(a) = S(a \,\|\, 0^{k_2}) \,\|\, \cdots \,\|\, S(a \,\|\, 1^{k_2})$$

for all $a \in \{0,1\}^{k_1}$, where $k = k_1 + k_2$ and $k_2 = \log_2 \ell$. To compute $S(x)$ for $x \in \{0,1\}^k$, we proceed in two steps:

1. Write $x = a\|b$ for $a \in \{0,1\}^{k_1}$ and $b \in \{0,1\}^{k_2}$, and compute $z = S'(a) = S(a\|0^{k_2})\| \cdots \|S(a\|1^{k_2})$
2. Viewing z as a k_2-bit input and k'-bit output table, compute $y = z(b) = S(x)$.

We must show how to compute $y = S(x)$ with the two steps above when the input x is shared with n shares x_i. In the first step we proceed as in Algorithm 1, except that the new table S' has a k_1-bit input instead of a k-bit input, and $\omega = \ell \cdot k'$-bit output instead of k'-bit output. Note that the table S' contains $2^{k_1} = 2^{k-k_2} = 2^k/\ell$ elements instead of 2^k for the original S. Since we assume that a read/write operation on a ω-bit register takes unit time, the complexity of the first step is now $\mathcal{O}(2^k/\ell \cdot n^2)$. Note that S and S' take the same amount of memory in RAM; in the first step of our countermeasure we can achieve a speed-up by a factor ℓ because we are moving ℓ blocks of k' bits at a time inside registers of size $\omega = \ell \cdot k'$ bits.

The second step requires a slight modification of Algorithm 1. Namely we must view the output z from Step 1 as a look-up table with k_2-bit input and k'-bit output. However this output z is now obtained in shared form, namely we get shares $z_1, \ldots, z_n$ such that $z = z_1 \oplus \cdots \oplus z_n$, whereas in Algorithm 1 the look-up table $S(x)$ is a public table. This is not a problem, as we can simply keep this table in shared form when initializing the $T(u)$ table at Line 2 of Algorithm 1. More precisely in the second step we can initialize the table $T(u)$ with:

$$T(u) = (z_1(u), \ldots, z_n(u)) \in (\{0,1\}^{k'})^n$$

for all $u \in \{0,1\}^{k_2}$, and we still have $\oplus\big(T(u)\big) = z(u)$ for all u as required. The rest of Algorithm 1 is the same. Since the second step uses a table of size $2^{k_2} = \ell$ elements, its complexity is $\mathcal{O}(\ell \cdot n^2)$.

The full complexity of our variant countermeasure is therefore $\mathcal{O}((2^k/\ell + \ell) \cdot n^2)$. This is minimized for $\ell = 2^{k/2}$. If we have large enough register size ω so that

we can take $\ell = \omega/k' = 2^{k/2}$, then the complexity of our variant countermeasure becomes $\mathcal{O}(2^{k/2} \cdot n^2)$, the same complexity as the Carlet *et al.* countermeasure.[6]

The following Lemma shows that our variant countermeasure achieves the same level of security as Algorithm 1; the proof is essentially the same as the proof of Lemma 1 and is therefore omitted.

Lemma 2. *Let $(x_i)_{1 \leq i \leq n}$ be the input shares of the above countermeasure for large register size, and let t be such that $2t < n$. For any set of t intermediate variables, there exists a subset $I \subset [1, n]$ of indices such that $|I| \leq 2t < n$ and the distribution of those t variables can be perfectly simulated from the x_i's with $i \in I$. The output shares $y_{|I}$ can also be perfectly simulated from $x_{|I}$.*

4 Higher Order Masking of a Full Block-Cipher

In this section we show how to integrate our countermeasure into a full block-cipher. We consider a block-cipher with the following operations: Xor operation $z = x \oplus y$, linear (or affine) transform $y = f(x)$, and look-up table $y = S(x)$. This covers both AES and DES block-ciphers. We show how to apply high-order masking to these operations, in order to protect a full block-cipher against t-th order attacks.[7]

Xor Operation. We consider a Xor operation $z = x \oplus y$. Taking as input the shares x_i and y_i such that $x = x_1 \oplus \cdots \oplus x_n$ and $y = y_1 \oplus \cdots \oplus y_n$, it suffices to compute the shares $z_i = x_i \oplus y_i$.

Linear Operation. We consider a linear operation $y = f(x)$. Taking as input the shares x_i such that $x = x_1 \oplus \cdots \oplus x_n$, it suffices to compute the shares $y_i = f(x_i)$ separately.

Table Look-Up. A table look-up $y = S(x)$ is computed using our previous Algorithm 1.

Input Encoding. Given x as input, we first encode x as $x_1 = x$ and $x_i = 0$ for $2 \leq i \leq n$. Secondly we let $(x_1, \ldots, x_n) \leftarrow \mathsf{RefreshMasks}(x_1, \ldots, x_n)$.

Output Decoding. Given $y_1, \ldots, y_n$ as input, we compute $y = y_1 \oplus \cdots \oplus y_n$ using Algorithm 3 below.

Key Shares Refreshing. As mentioned in Section 2.2 we must re-randomize the key shares between successive executions of the block-cipher in order to achieve security in the full model. Using Algorithm 4 below we perform both an initial Key Shares Refreshing (before the shares sk_i are used to evaluate the block-cipher), and a final Key Shares Refreshing (before the key shares sk_i are stored for the next execution).[8]

[6] Note that for DES with 32-bit registers we can take the optimum $\ell = 2^{6/2} = 8$. However for AES the optimum $\ell = 2^{8/2} = 16$ would require 128-bit registers.

[7] Xor is a linear operation, so one could consider the linear operation $y = f(x)$ only, but it seems more convenient to consider the Xor operation separately.

[8] Note that for both algorithms 3 and 4 the RefreshMasks procedure must be applied with the *tmp* randoms generated with the appropriate bit-size (instead of k').

Algorithm 3. Shares recombination

Input: $y_1, \ldots, y_n$
Output: y such that $y = y_1 \oplus \cdots \oplus y_n$
1: **for** $i = 1$ **to** n **do** $(y_1, \ldots, y_n) \leftarrow \mathsf{RefreshMasks}(y_1, \ldots, y_n)$
2: $c \leftarrow y_1$
3: **for** $i = 2$ **to** n **do** $c \leftarrow c \oplus y_i$
4: **return** c

Algorithm 4. Key Shares Refreshing

Input: $sk_1, \ldots, sk_n$ such that $sk = sk_1 \oplus \cdots \oplus sk_n$
Output: $sk_1, \ldots, sk_n$ such that $sk = sk_1 \oplus \cdots \oplus sk_n$
1: **for** $i = 1$ **to** n **do** $(sk_1, \ldots, sk_n) \leftarrow \mathsf{RefreshMasks}(sk_1, \ldots, sk_n)$
2: **return** $sk_1, \ldots, sk_n$

This terminates the description of our randomized encryption algorithm. The following theorem proves the security of the randomized encryption scheme defined above in the full model, under the condition $n \geq 2t + 1$; we give the proof in the full version of this paper [Cor13a]. This improves the bound $n \geq 4t + 1$ from [ISW03] for stateful circuits. We stress that any set of t intermediate variables can be probed by the adversary, including variables in the input encoding, output decoding, and key shares refreshing; that is, no operation is assumed to be leak-free.

Theorem 1. *The randomized encryption scheme defined above achieves t-th order security in the full model for $n \geq 2t + 1$.*

Remark 3. The input encoding operation need not be randomized by RefreshMasks; this is because the input x is public and given to the simulator, who can therefore perfectly simulate the initial shares $x_{|I}$ for any subset $I \subset [1, n]$. Moreover in the restricted model the key shares refreshing is not necessary. In practice we can keep both operations as their time complexity is only $\mathcal{O}(n)$ and $\mathcal{O}(n^2)$ respectively.

Remark 4. We stress that the secret key sk must be initially provided with randomized shares, since sk is secret and not given to the simulator; in other words it would be insecure for the randomized block-cipher to receive sk as input and perform the initial input encoding on sk by himself.

Remark 5. In the output decoding operation we perform a series of n mask refreshing before computing y. This is to enable a correct simulation of the intermediate variables c at Line 3 in case they are probed by the adversary.

5 Practical Implementation

We have performed a practical implementation of our new countermeasure for both AES and DES, using a 32-bit architecture so that we could apply our

large register variant. More precisely we could pack $\ell = 4$ output bytes for AES, and $\ell = 8$ output 4-bit nibbles for DES. For comparison we have also implemented the Rivain-Prouff countermeasure [RP10] for AES and the Carlet *et al.* countermeasure [CGP+12] for DES; for the latter we have used the technique from [RV13], in which the evaluation of a DES Sbox requires only 7 non-linear multiplications. The performances of our implementations are summarized in Table 2. We use the bound $n = 2t + 1$ for the full model of security (which implies security in the restricted model).

Table 2. Comparison of secure AES and DES implementations, for the Rivain-Prouff (RP) countermeasure, and our Table Recomputation (TR) countermeasure. The implementation was done in C on a MacBook Air running on a 1.86 GHz Intel processor. We denote the number of calls to the random number generator (times 10^3), the required memory in bytes (only for the Sbox computation part), the total running time in ms, and the Penalty Factor (PF) compared the the unmasked implementation.

	t	n	Rand	Mem	Time	PF
AES					0.0018	1
AES, RP	1	3	2.1	20	0.092	50
AES, TR	1	3	44	1579	0.80	439
AES, RP	2	5	6.8	30	0.18	96
AES, TR	2	5	176	2615	2.2	1205
AES, RP	3	7	14	40	0.31	171
AES, TR	3	7	394	3651	4.4	2411
AES, RP	4	9	24	50	0.51	276
AES, TR	4	9	700	4687	7.3	4003

	t	n	Rand	Mem	Time	PF
DES					0.010	1
DES, RP	1	3	2.8	72	0.47	47
DES, TR	1	3	8.5	423	0.31	31
DES, RP	2	5	9.2	118	0.78	79
DES, TR	2	5	33	691	0.59	59
DES, RP	3	7	19	164	1.3	129
DES, TR	3	7	75	959	0.90	91
DES, RP	4	9	33	210	1.9	189
DES, TR	4	9	133	1227	1.4	142

We obtain that in practice for AES our algorithm is an order of magnitude less efficient than Rivain-Prouff, which can take advantage of the special algebraic structure of the AES Sbox; however for DES our algorithm performs slightly better than the Carlet *et al.* countermeasure. Note that this holds for a 32-bit architecture; on a 8-bit architecture the comparison could be less favorable. The source code of our implementations is publicly available [Cor13b].

One could think that because of the large penalty factors the countermeasures above are unpractical. However in some applications the block-cipher evaluation can be only a small fraction of the full protocol (for example in a challenge-response authentication protocol), and in that case a penalty factor of say 100 for a single block-cipher evaluation may be acceptable.

Acknowledgments. We would like to thank the Eurocrypt 2014 referees for their helpful comments.

References

CGP+12. Carlet, C., Goubin, L., Prouff, E., Quisquater, M., Rivain, M.: Higher-order masking schemes for s-boxes. In: Canteaut, A. (ed.) FSE 2012. LNCS, vol. 7549, pp. 366–384. Springer, Heidelberg (2012)

CJRR99. Chari, S., Jutla, C.S., Rao, J.R., Rohatgi, P.: Towards sound approaches to counteract power-analysis attacks. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 398–412. Springer, Heidelberg (1999)

Cor13a. Coron, J.-S.: Higher order masking of look-up tables. Cryptology ePrint Archive, Report 2013/700. Full version of this paper (2013), `http://eprint.iacr.org/`

Cor13b. Coron, J.-S.: Implementation of higher-order countermeasures (2013), `https://github.com/coron/htable/`

CPR07. Coron, J.-S., Prouff, E., Rivain, M.: Side channel cryptanalysis of a higher order masking scheme. In: Paillier, P., Verbauwhede, I. (eds.) CHES 2007. LNCS, vol. 4727, pp. 28–44. Springer, Heidelberg (2007)

CRR02. Chari, S., Rao, J.R., Rohatgi, P.: Template attacks. In: Kaliski Jr., B.S., Koç, Ç.K., Paar, C. (eds.) CHES 2002. LNCS, vol. 2523, pp. 13–28. Springer, Heidelberg (2003)

HOM06. Herbst, C., Oswald, E., Mangard, S.: An AES smart card implementation resistant to power analysis attacks. In: Zhou, J., Yung, M., Bao, F. (eds.) ACNS 2006. LNCS, vol. 3989, pp. 239–252. Springer, Heidelberg (2006)

ISW03. Ishai, Y., Sahai, A., Wagner, D.: Private circuits: Securing hardware against probing attacks. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 463–481. Springer, Heidelberg (2003)

KJJ99. Kocher, P.C., Jaffe, J., Jun, B.: Differential power analysis. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 388–397. Springer, Heidelberg (1999)

Mes00. Messerges, T.S.: Using second-order power analysis to attack dpa resistant software. In: Paar, C., Koç, Ç.K. (eds.) CHES 2000. LNCS, vol. 1965, pp. 238–251. Springer, Heidelberg (2000)

PR13. Prouff, E., Rivain, M.: Masking against side-channel attacks: A formal security proof. In: Johansson, T., Nguyen, P.Q. (eds.) EUROCRYPT 2013. LNCS, vol. 7881, pp. 142–159. Springer, Heidelberg (2013)

RDP08. Rivain, M., Dottax, E., Prouff, E.: Block ciphers implementations provably secure against second order side channel analysis. In: Nyberg, K. (ed.) FSE 2008. LNCS, vol. 5086, pp. 127–143. Springer, Heidelberg (2008)

RP10. Rivain, M., Prouff, E.: Provably secure higher-order masking of AES. In: Mangard, S., Standaert, F.-X. (eds.) CHES 2010. LNCS, vol. 6225, pp. 413–427. Springer, Heidelberg (2010)

RV13. Roy, A., Vivek, S.: Analysis and improvement of the generic higher-order masking scheme of FSE 2012. In: Bertoni, G., Coron, J.-S. (eds.) CHES 2013. LNCS, vol. 8086, pp. 417–434. Springer, Heidelberg (2013)

Sha79. Shamir, A.: How to share a secret. Commun. ACM 22(11), 612–613 (1979)

SP06. Schramm, K., Paar, C.: Higher order masking of the AES. In: Pointcheval, D. (ed.) CT-RSA 2006. LNCS, vol. 3860, pp. 208–225. Springer, Heidelberg (2006)