Hiding Higher-Order Univariate Leakages by Shuffling Polynomial Masking Schemes

A More Efficient, Shuffled, and Higher-Order Masked AES S-box

Fabrizio De Santis
desantis@tum.de

Tobias Bauer
tobias.bauer@tum.de

Georg Sigl
sigl@tum.de

Lehrstuhl für Sicherheit in der Informationstechnik
Technische Universität München (TUM)
Munich, Germany

ABSTRACT

Polynomial masking is a glitch-resistant and higher-order masking scheme based upon Shamir's secret sharing scheme and multi-party computation protocols. Polynomial masking was first introduced at CHES 2011, while a 1^{st}-order implementation of the AES S-box on FPGA was presented at CHES 2013. In this latter work, the authors showed a 2^{nd}-order *univariate* leakage by side-channel collision analysis on a tuned measurement setup. This negative result motivates the need to evaluate the performance, area-costs, and security margins of combined shuffled *and* higher-order polynomially masking schemes to counteract trivial univariate leakages. In this work, we provide the following contributions: first, we introduce additional principles for the selection of efficient addition chains, which allow for more compact and faster implementations of cryptographic S-boxes. Our 1^{st}-order AES S-box implementation requires approximately 27% less registers, 20% less clock cycles, and 5% less random bits than the CHES 2013 implementation. Then, we propose a lightweight shuffling countermeasure, which *inherently* applies to polynomial masking schemes and effectively enhances their univariate security at negligible area expenses. Finally, we present the design of a combined shuffled and higher-order polynomially masked AES S-box in hardware, while providing ASIC synthesis and side-channel analysis results in the Electro-Magnetic (EM) domain.

CCS Concepts

•Security and privacy → Hardware security implementation; Side-channel analysis and countermeasures; *Block and stream ciphers;*

Keywords

Shuffling; Polynomial Masking; Multi-Party Computation; Secret Sharing; Side-Channel Analysis; AES

Tis'16, October 24 2016, Vienna, Austria

© 2016 Copyright held by the owner/author(s). Publication rights licensed to ACM. ISBN 978-1-4503-4575-0/16/10...$15.00

DOI: http://dx.doi.org/10.1145/2996366.2996370

1. INTRODUCTION

With the widespread adoption of pocket-sized embedded devices which can easily fall into the adversary's hands, like banking and Pay-TV smart cards, passport ICs, and RFID transportation cards, the security of cryptographic devices does not only depend on the cryptographic strengths of mathematical algorithms, but also, and mainly, on the physical security of their implementations. In the last two decades, several side-channel analysis techniques have been developed to recover the secret keys stored in cryptographic devices. In parallel, a large body of countermeasures have been proposed to thwart side-channel attacks at different levels of abstraction (mainly at the logic [20, 16], algorithmic [7, 19, 22], and protocol levels [15, 1, 11]), but no conclusive nor satisfactory "costs vs. performance vs. security trade-off" has been achieved yet. In practice, one of the most effective ways of protecting block-cipher implementations against side-channel analysis is to employ higher-order masking schemes. The basic idea of higher-order masking schemes is to split the computation of secret-dependent intermediate values into multiple shares, as to increase the adversary's data complexity exponentially to the number of shares [6, 21]. In the last years, a number of higher-order masking schemes have been proposed in literature along with statements of provable security. Nevertheless, they turned out to be susceptible to side-channel attacks in many occasions, e.g. because of oversimplified modeling assumptions like in the case of glitches [14]. Nowadays, the two most promising approaches to glitch-resistant and higher-order masking schemes are threshold implementations (TI) [19, 4, 3, 9] and polynomial masking schemes [22, 24, 17, 10]. While TI are normally faster and more compact than polynomial masking schemes, these latter still offer some advantages over the former: (1) while TI typically require an ad-hoc re-design for each block cipher and/or masking order, polynomial masking schemes can be easily adapted to any block-cipher and inherently scaled to any masking order using regular structures e.g., hardware modules can be parametrized to any order using VHDL's GENERICS and synthesized only to that masking order required for a specific certification; (2) in a standard setting, polynomially masking schemes offer more security than TI, as they process one share per clock cycle, hence requiring the adversary to estimate the leakage distribution over more dimensions.

Related Work.

Polynomial masking was introduced at CHES 2011 [22], while a 1st-order AES S-box implementation on FPGA was presented at CHES 2013 [17]. In this latter work, the authors showed a 2nd-order univariate leakage in the AES S-box, obtained by clocking the target device at moderately high frequencies and using active probe pre-amplifiers for measurements in the power domain. It is important to notice that these type of attacks strongly depend on the measurement setup and do not undermine the security of polynomial masking schemes specifically, but represent a general threat for all masking schemes, including higher-order TI.

Contribution.

In this work, we improve the CHES 2013 implementation along three directions: (1) we introduce new principles for the selection of more efficient addition chains in the form $a_i = a_{i-1} + a_k$, which allow for more efficient S-box implementations; (2) we propose a lightweight shuffling countermeasure, which inherently applies to polynomial masking schemes, i.e. it is agnostic of the underlying block-cipher and can be deployed for both hardware and software higher-order masking implementations; (3) we present the design of a shuffled and higher-order polynomially masked AES S-box in hardware, while providing ASIC synthesis (up to the 6th-order) and side-channel analysis (up to the 2nd-order) results in the Electro-Magnetic (EM) domain. Our side-channel analysis confirms the presence of univariate 2nd-order leakage in the EM domain, i.e. the leakage is independent of the load at measuring point on the target device. Also, it shows that shuffled 1st-order and 2nd-order AES S-box implementations resist to univariate attacks up to 10,000,000 measurements in a worst-case scenario analysis, although the former is 40% faster and 69% smaller than the latter. This result highlights the practical relevance of shuffling to *improve* the resistance of 1st-order implementations against univariate attacks. Note that the possibility of easily combining shuffling and masking countermeasures at almost no additional costs (where the shuffling is applied "within" the masking itself) should be considered as a yet compelling advantage of polynomial schemes over TI.

2. BACKGROUND

In the following, background information about the AES S-box and higher-order polynomial masking is provided.

2.1 The AES S-box

The AES S-box is defined as the composition of two operations: an inversion in the binary extension field $\mathsf{GF}(2^8) \simeq \mathbf{F}_2[x]/(x^8 + x^4 + x^3 + x + 1)$ followed by an affine transformation in the binary field $\mathsf{GF}(2)$. Using the Lagrange interpolation formula, the AES S-box $\mathsf{S}(\nu)$ can be rewritten as the composition of $\mathsf{GF}(2^8)$ operations only:

$$\gamma = \mathsf{S}(\nu) = \delta_0 + \sum_{k=1}^{8} \delta_k \nu^{255-2^k} = \delta_0 + \sum_{k=1}^{8} \delta_k (\nu^{-1})^{2^k}, \quad (1)$$

where $(\delta_k)_{0\leq k\leq 8}$ are the coefficients defined by the vector $(\mathtt{0x63}, \mathtt{0x05}, \mathtt{0x09}, \mathtt{0xF9}, \mathtt{0x25}, \mathtt{0xF4}, \mathtt{0x01}, \mathtt{0xB5}, \mathtt{0x8F})$, cf. [8].

2.2 Higher-Order Polynomial Masking

A d^{th}-order masking scheme splits key-dependent intermediate values into m shares, such that no subset of the shares with cardinality $d < m$ depends on the original secrets. A d^{th}-order masking scheme can be defeated by $(d+1)^{th}$-order side-channel by exploiting, e.g. the leakage of $(d+1)$ shares combined together or the $(d+1)^{th}$ statistical moment of the leakage distribution.

Polynomial masking is a d^{th}-order masking scheme based upon the Shamir's secret sharing scheme and Multi-Party Computation (MPC) protocols [25, 2, 22, 24]. Polynomial masking schemes are particularly suitable to protect the implementation of cryptographic algorithms defined over finite fields, e.g. $\mathsf{GF}(2^8)$. Let $x \in \mathsf{GF}(2^n)$ be a secret intermediate value, $P_x(Y) = x + \sum_{j=1}^{d} r_j Y^j$ be its associated polynomial obtained by drawing d coefficients $(r_j)_{1\leq j\leq d} \xleftarrow{\$} \mathsf{GF}(2^n)$ at random, and $(\alpha_i)_{1\leq i\leq m} \xleftarrow{\$} \mathsf{GF}(2^n)^*$ be m public random elements such that $\forall i, j \in [1, m]$, $\alpha_i \neq \alpha_j$. The secret shares $(x_i)_{1\leq i\leq m}$ of x are obtained by evaluating P_x at $(\alpha_i)_{1\leq i\leq m}$:

$$x_i = x + \sum_{j=1}^{d} r_j \alpha_i^j,\ i \in [1, m]. \quad (2)$$

Once the shares are created, then the addition of two secrets $z = x+y$ can be performed independently on the single shares without the need of fresh randomness, such that:

$$z_i = (x_i + y_i) + \sum_{j=1}^{d} (r_j + r'_j)\alpha_i^j, \quad i \in [1, m], \quad (3)$$

where $P_x(Y) = x + \sum_{j=1}^{d} r_j Y^j$, $P_y(Z) = y + \sum_{j=1}^{d} r'_j Z^j$. Similarly, the multiplication of a secret by a constant value $z = cx$, $c \neq 0$ can be applied to the single shares without requiring fresh randomness:

$$z_i = (cx_i) + \sum_{j=1}^{d} (cr_j)\alpha_i^j, \quad i \in [1, m], \quad (4)$$

On the contrary, the multiplication of two secrets $z = xy$ can not be performed in a straightforward way for two reasons: (1) the result of the multiplication of two polynomials is a polynomial of degree $2d$; (2) its coefficients are not randomly distributed. The Ben-Or, Goldwasser and Wigderson multiplication method solves these two issues by performing the multiplication of two secrets in three steps [2]:

$$\begin{cases} \texttt{Step 1)} & t_i = x_i y_i \\ \texttt{Step 2)} & q_{i,k} = t_i + \sum_{j=1}^{d} s_j \alpha_k^j \ ,\ i,k \in [1,m]. \\ \texttt{Step 3)} & z_i = \sum_{w=1}^{m} q_{w,i} \lambda_w \end{cases} \quad (5)$$

First, the initial shares are multiplied in a straightforward way (`Step 1`). Then, the resulting shares $(t_i)_{1\leq i\leq m}$ are re-shared using d fresh randomly generated masks $(s_j)_{1\leq j\leq d}$ (`Step 2`). Finally, a correct sharing of $z = xy$ is obtained from the $(\lambda_i)_{1\leq i\leq m}$ coefficients laying on the first row of the inverse Vandermonde matrix $(\alpha_i^j)_{1\leq i,j\leq m}$ (`Step 3`):

$$\lambda_i = \prod_{k=1,k\neq i}^{m} \frac{-\alpha_k}{\alpha_i - \alpha_k}. \quad (6)$$

Similarly, the reconstruction of a secret x from its shares $(x_i)_{1\leq i\leq m}$ is obtained using Lagrange's interpolation formulas, as follows:

$$x = \sum_{i=1}^{m} x_i \lambda_i. \quad (7)$$

3. MORE EFFICIENT ADDITION CHAINS

This section introduces new principles for the selection of more efficient addition chains, which lead to more compact and faster implementations of cryptographic S-boxes.

An addition chain for a positive integer q is a sequence of positive integers $(a_0 = 1, \ldots, a_\ell = q)$, such that for every $1 \le i \le \ell$, there exist $0 \le j, k < i$ and $a_i = a_j + a_k$, where ℓ is the length of the chain. Addition chains can be used to implement fast exponentiations x^q, for large q. In this case, every addition corresponds to a multiplication and every doubling corresponds to a squaring operation. Addition chains in the form $a_i = a_{i-1} + a_k$ are particularly suitable for compact *hardware* implementations, as the operand a_{i-1} is directly available from the previous computation step and it does not have to be stored. These type of addition chains are called *star chains* [5].

In order to enable efficient computations, the shortest addition chains for q which maximize the number of doublings, i.e. $a_i = 2a_{i-1}$, are typically used. In recent works, the AES inversion $x^{-1} = x^{254} \in \mathsf{GF}(2^8)$ was implemented using the addition chain $\mathcal{C}^{254}_{\mathsf{orig}} = (1, 2, 3, 6, 12, 15, 30, 60, 120, 240, 252, 254)$ [23, 13, 17], i.e.:

$$x \xrightarrow{S} x^2 \xrightarrow{M} x^3 \xrightarrow{2S} x^{12} \xrightarrow{M} x^{15} \xrightarrow{4S} x^{240} \xrightarrow{M} x^{252} \xrightarrow{M} x^{254}.$$

The addition chain $\mathcal{C}^{254}_{\mathsf{orig}}$ requires a total of 11 steps, of which 7 are squaring operations and 4 are multiplications. However, it can be noted that using $\mathcal{C}^{254}_{\mathsf{orig}}$ requires the storage of at least 3 intermediate values, namely x^2, x^3, and x^{12}, and does not generate any intermediate value that can be useful for subsequent computations, e.g. the affine transformation in the AES algorithm. Hence, we introduce the following two principles to improve the selection of addition chains along two directions:

PRINCIPLE 1. *Select those addition chains which generate intermediate values that can be useful for subsequent operations within the considered cryptographic algorithm.*

PRINCIPLE 2. *Select those addition chains which minimize the number of intermediate values that must be saved for exponentiation.*

Clearly, those addition chains fulfilling Principle 1 and Principle 2 provide two additional benefits, which are advantageous for efficient implementations: they allow for an overall speed-up of the cryptographic implementation, by computing values which are needed in successive operations (Principle 1), and for a minimization of the storage required to save intermediate values of exponentiations (Principle 2). Note that, while Principle 1 is algorithm specific, Principle 2 is generally valid and of independent interest.

In order to find star chains for AES, which fulfill Principle 1 and Principle 2, we used a naïve exhaustive search algorithm to test all the addition chains in the form $a_i = a_{i-1} + a_k$ up to a given length ℓ (cf. Algorithm 1). The algorithm runs in $\mathcal{O}(\ell!)$ and proceeds by constructing a tree of all possible addition chains of length ℓ, recursively.

By running Algorithm 1, we found $|\mathcal{S}| = 6966$ addition chains for $(q = 254, \ell = 11)$, of which 754 were using 7 squaring and 4 multiplications, of which 106 required storage of only one single intermediate value, of which 55 also generated the value 127, that can be used during the AES affine transformation. Out of $\mathcal{S}$, we choose (arbitrarily) the addition chain $\mathcal{C}^{254}_{\mathsf{new}} = (1, 2, 4, 8, 9, 18, 36, 54, 108, 126, 127, 254)$, i.e.:

Algorithm 1 Exhaustive Search for Star Chains.

```
Input: $q > 0$, $\ell \ge 0$, $\mathcal{C} = (a_0 = 1, \ldots, a_r)$.
Output: $\mathcal{S} = \{(a_0 = 1, \ldots, a_r) : \forall\, 0 \le r \le \ell,\ a_r = q\}$.

1: procedure FindChain($q, \ell, \mathcal{C}$)
2:     if $a_r = q$ then
3:         Save($\mathcal{C}$)          $\{\mathcal{C} = (a_0 = 1, \ldots, a_{r \le \ell} = q)\}$
4:     else if $r < \ell$ then
5:         for $k = 0$ to $r$ do
6:             $\mathcal{C} \leftarrow (a_0, \ldots, a_r, a_{r+1} = a_r + a_k)$
7:             FindChain($q, \ell, \mathcal{C}$)
8:         end for
9:     end if
10: end procedure
```

$$x \xrightarrow{3S} x^8 \xrightarrow{M} x^9 \xrightarrow{2S} x^{36} \xrightarrow{M} x^{54} \xrightarrow{S} x^{108} \xrightarrow{M} x^{126} \xrightarrow{M} x^{127} \xrightarrow{S} x^{254}.$$

The addition chain $\mathcal{C}^{254}_{\mathsf{new}}$ has the following properties: (1) it consists of 11 steps, of which 7 are squaring operations and 4 are multiplications; (2) it produces the value x^{127}, which can be used by the AES affine transformation, and it requires only one intermediate value to be stored, namely x^{18}. Hence, using $\mathcal{C}^{254}_{\mathsf{new}}$ in place of $\mathcal{C}^{254}_{\mathsf{orig}}$ leads to faster and more compact AES S-box implementations: (1) the affine transformation results 1/7 faster, as one squaring operation can be skipped; (2) the inversion results 1/3 smaller, as only one intermediate value has to be stored (instead of three[1]).

Similarly, we searched for star chains for $(q = 14, \ell = 5)$, which can be useful for efficient implementations of AES in tower fields, i.e. $\mathsf{GF}((2^4)^2)$, where the inversion $x^{-1} = x^{14} \in \mathsf{GF}(2^4)$ is used. In previous works, the following addition chain $\mathcal{C}^{14}_{\mathsf{orig}} = (1, 2, 3, 6, 12, 14)$ was used [13]:

$$x \xrightarrow{S} x^2 \xrightarrow{M} x^3 \xrightarrow{2S} x^{12} \xrightarrow{M} x^{14}.$$

The addition chain $\mathcal{C}^{14}_{\mathsf{orig}}$ requires 5 steps consisting of 3 squaring and 2 multiplication operations. However, it requires to store the intermediate value x^2 to compute x^{14}.

By running Algorithm 1, we found 14 star chains for $(q = 14, \ell = 5)$, of which 8 were using 3 squaring and 2 multiplications, of which only 1 addition chain did not require the storage of any intermediate value, namely $\mathcal{C}^{14}_{\mathsf{new}} = (1, 2, 3, 6, 7, 14)$:

$$x \xrightarrow{S} x^2 \xrightarrow{M} x^3 \xrightarrow{S} x^6 \xrightarrow{M} x^7 \xrightarrow{S} x^{14}.$$

4. HARDWARE DESIGN

This section details the hardware design to shuffle a polynomially masked AES S-box for any order $d = (m-1)/2$. The design has an 8-bit interface, implements a single S-box instance, and consists essentially of three modules: (1) the shared multiplication module (`shamul`), which performs the

[1]Note that, despite this rather small improvement, the area saving of hardware implementations might be significant when addition chains are instantiated multiple times, e.g. this is the case for higher-order polynomial masking schemes.

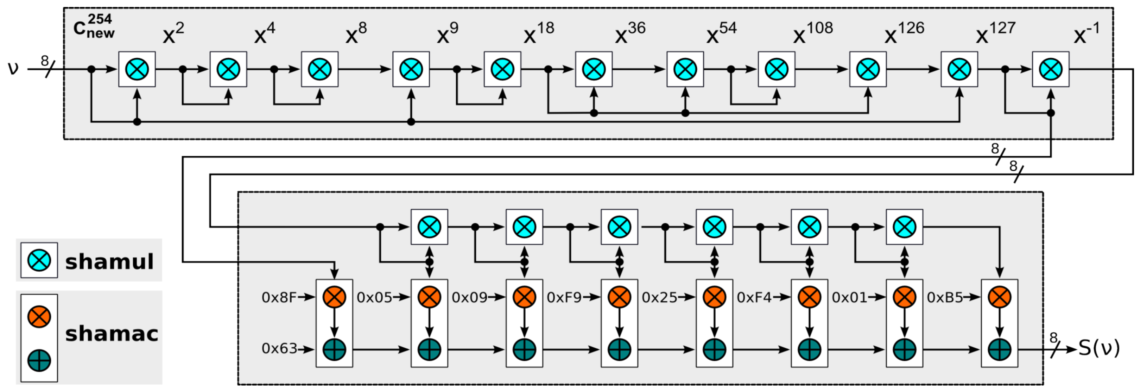

Figure 1: Shared AES S-box with $\mathcal{C}_{\text{new}}^{254}$.

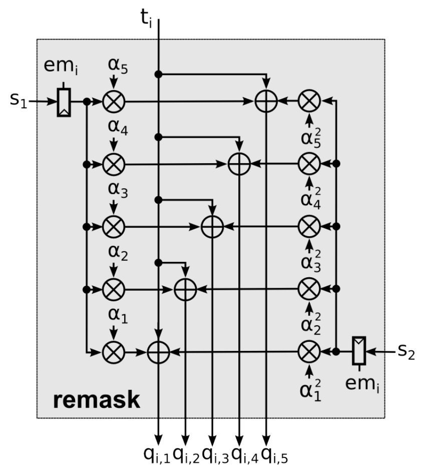

Figure 2: 2^{nd}-Order `remask` Module (i^{th} Share).

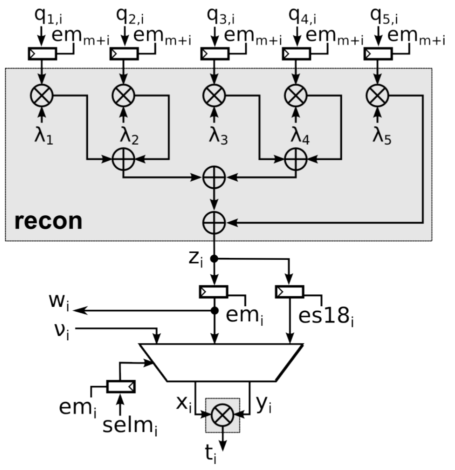

Figure 3: 2^{nd}-Order `recon` Module (i^{th} Share).

multiplication of two shared secrets; (2) the shared multiplication by a constant and addition module (**`shamac`**), which performs the multiplication of a secret by a constant value followed by a secret addition; (3) the shuffling module (**`shuffle`**), which generates random permutations from the symmetric group S_m to shuffle the activation order of the m secret shares within every other shared module. The processing of the shared S-box using the more efficient addition chain $\mathcal{C}_{\text{new}}^{254}$ is illustrated in Figure 1.

4.1 The `shamul` Module

The shared multiplication module (**`shamul`**) is used to perform the inversion defined by $\mathcal{C}_{\text{new}}^{254}$ and to iterate the squaring operations as needed by the affine transformation in $\mathsf{GF}(2^8)$, cf. Equation (1). It basically implements the multiplication of two shared secrets according to Equation (5) using three submodules: the re-masking module (**`remask`**), as illustrated in Figure 2 for the case $(d, m) = (2, 5)$, the reconstruction module (**`recon`**) and a $\mathsf{GF}(2^8)$ multiplier, as illustrated in Figure 3 for the case $(d, m) = (2, 5)$. These modules are implemented m times within the **`shamul`** module, one for each share $1 \leq i \leq m$. The first step of the shared multiplication is performed by the $m \times \mathsf{GF}(2^8)$ multipliers instantiated in the **`shamul`** module. The second step is implemented by the m **`remask`** modules, each one instantiating $m \times d$ $\mathsf{GF}(2^8)$ multipliers and $m \times d$ $\mathsf{GF}(2^8)$ adders. This step requires $d \times$ 8-bit of freshly generated random masks $(s_j)_{1 \leq j \leq d}$ for each share. Finally, the third step is implemented by the $m\times$ **`recon`** modules, each one instantiating $m \times \mathsf{GF}(2^8)$ multipliers and $(m-1) \times \mathsf{GF}(2^8)$ adders. Note that each one of the **`recon`** modules are additionally equipped with $m \times$ 8-bit DFF registers to block the glitches, arising from the different **`remask`** modules, joining together in the combinational paths, as well as $2 \times$ 8-bit DFF registers for storing the intermediate results of the inversion. Due to the usage of our selected addition chain $\mathcal{C}_{\text{new}}^{254}$, the total number of DFFs in the circuit is reduced from $m \times 40$ to $m \times 16$, when compared to [17]. For the same reason, also the size of the multiplexers (MUXes) selecting the input shares (x_i, y_i) to the $\mathsf{GF}(2^8)$ multipliers results significantly reduced in practice. Finally, note that the switching activity of all MUXes in the design is clocked using "DFFs with Enable" as to avoid glitches on the select lines [17].

The **`shamul`** takes a total of $2 \times m$ clock cycles to perform the multiplication of two secrets and works as follows: in the first m clock cycles, the signals **`selm`**$_i$ for $i \in [1, m]$ select the proper inputs (where ν_i represents the i^{th} input share of the S-box) to the $\mathsf{GF}(2^8)$ multipliers performing $t_i = x_i y_i$ and the enable signals **`em`**$_i$ for $i \in [1, m]$ are asserted to perform the first two steps of the shared multiplication. The results

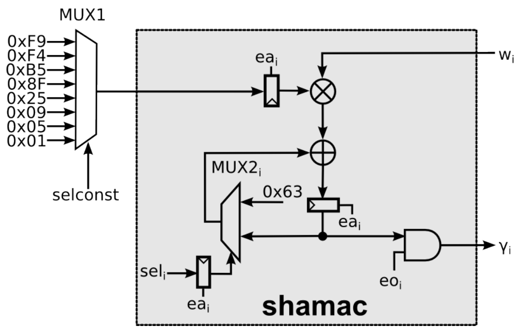

Figure 4: Any-Order shamac Module (i^{th} Share).

are stored in the registers $\mathtt{q_{j,i}}$ for $i, j \in [1, m]$. Then, during the next m clock cycles, the enable signals $\mathtt{em_{m+i}}$ are asserted to finalize the shared multiplication. Note that the enable signals $\mathtt{es18_i}$ are activated only in the step in which the shared x^{18} value is computed. The output of the shamul are the shares $\mathtt{w_i}$ for $i \in [1, m]$ which are sent to the shamac module.

4.2 The shamac Module

The shamac module performs the shared multiplication by a constant followed by a shared addition according to Equation (4) and Equation (3), respectively. These operations do not require fresh randomness and can be performed independently on the individual shares, thus requiring only m clock cycles to be performed. The shamac is used during the affine transformation, but it can be extended straightforwardly to support e.g. the initial sharing of the plaintext and key bytes, the initial AddRoundkey, and the MixColumns+AddRoundkey operations. One share of the shamac module is illustrated in Figure 4. Each share deploys $1 \times \mathsf{GF}(2^8)$ multiplier, $1 \times \mathsf{GF}(2^8)$ adder and $2 \times$ 8-bit DFF registers. The affine transformation uses the shamac module as follows: at the beginning the $\mathtt{sel_i}$ selects the constant 0x63 from the MUX2$_i$, while the signal selconst selects the constant 0x8F from the MUX1 (common to all shares). Hence, the enable signal $\mathtt{ea_i}$ is asserted and the result is stored in the register right below the adder. Finally, the MUX2$_i$ chooses the loop back input to continue the affine transformation using the constants (0x05, 0x09, 0xF9, 0x25, 0xF4, 0x01, 0xB5) selected by the signal selconst on the MUX1. Note that, in contrast to [17], we allow both the shamul and the shamac to be active at the same time during the affine transformation. This allows to save $7 \times m$ clock cycles per S-box computation, while not having any apparent impact on the side-channel security of the implementation (cf. Section 5).

4.3 The shuffle Module

Polynomially masking works by activating the secret shares individually, one after the other, such that only one secret share is active per clock cycle. For instance, the i^{th} share of the shamac module is activated when the m-bit enable signal $\mathtt{ea_i}$ is set to logic '1'. This value is generated e.g. by a $\lceil \log_2 m \rceil$-bit counter, which is incremented every clock cycle and set to zero every m clock cycles. However, since there are no strict requirements on the activation order of the secret shares, they can be activated in a random order at each secret computation, e.g. generating a random permutation from the symmetric group S_m every m clock cycles. This acts as a shuffling countermeasure (within the masking scheme), which can be used to enhance the side-channel security of polynomial masking without penalties on the running time.

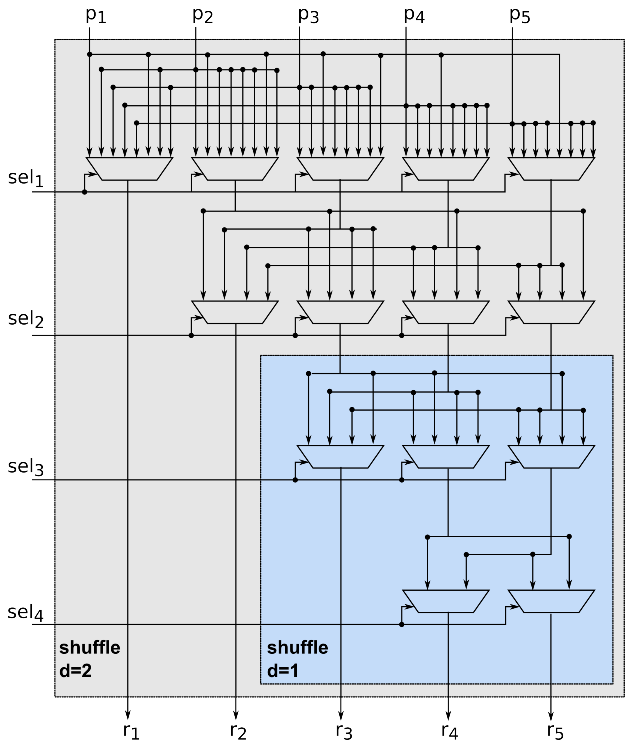

Figure 5: 1^{st}-Order (blue box) and 2^{nd}-Order (gray box) shuffle Module.

One simple way to create random permutations is to use the Fisher-Yates-Durstenfeld (FYD) algorithm, e.g. in forward direction [12]. However, as the number of shares $m = 2d + 1$ is not a power of two, random permutations can be approximated using a slightly modified FYD algorithm, as shown in Algorithm 2. This variant basically generates only the values $[0, m - \omega]$ for $\omega \in \{1, \ldots, m - 1\}$ using a modulo operation. The shuffle modules implements Algorithm 2 cascading a bunch of multiplexers, as illustrated in Figure 5. The shuffle module requires $m - 1$ stages, in which $(m+1-\omega)$ multiplexers with $2^{\lceil \log_2(m+1-\omega) \rceil}$ inputs are deployed. In order to generate a random permutation, the select signals $\mathtt{sel}_\omega$ takes a uniformly random value, which permutes the current input $(p_i)_{1 \leq i \leq m}$ into a new permutation $(r_i)_{1 \leq i \leq m}$ that gets stored into a register array consisting of $m \times \lceil \log_2(m) \rceil$-bit registers. The shuffle module generates a new permutation (from the previous one) every m clock cycles. The new permutation is then used to shuffle the activation order of the shares within any other shared module[2] during the next shared operation.

4.4 Performance Evaluation

We synthesized the design using Synopsys Design Compiler J-2014.09-SP3 targeting a TSMC 45 nm standard technology library (tcbn45gsbwptc) at 1 MHz. Our gate equiva-

[2]Note that the shuffle module can be used "as it is" to shuffle also the activation order of the shared key and state arrays within the AES.

Algorithm 2 Modified FYD in Forward Direction.

```
Input: Input Permutation $(p_1, ..., p_m)$
Output: Output Permutation $(r_1, ..., r_m)$
 1: for $\omega = 1$ to $m$ do
 2:     $a_\omega \leftarrow p_\omega$
 3: end for
 4: for $\omega = 1$ to $m - 1$ do
 5:     $sel_\omega \xleftarrow{\$} [0, 2^{\lceil \log_2(m+1-\omega) \rceil} - 1]$
 6:     $j \leftarrow \mathrm{mod}(sel_\omega, m + 1 - \omega)$
 7:     $t \leftarrow a_\omega$
 8:     $a_\omega \leftarrow a_{j+\omega}$
 9:     $a_{j+\omega} \leftarrow t$
10: end for
11: for $\omega = 1$ to $m$ do
12:     $r_\omega \leftarrow a_\omega$
13: end for
```

lent estimations are obtained using the smallest 2-to-1 NAND gate available in the library (`ND2D0BWP`), whose area accounts for 0.705 600 µm^2. We used the `compile` command *without* the `-ultra` parameter to allow for a fair comparison of the results with the literature (where not otherwise specified). In this library, the GF(2^8) multiplier and adder cost 280 GE and 20 GE, respectively. Table 1 summarizes the synthesis and performance results of our shuffled polynomially masked AES S-box implementation for $d \in [1, 6]$. The number of clock cycles and random bit have been obtained from the formulas $35 \times m$ and $17 \times m \times d$, respectively. Note that, while the number of clock cycles grows linearly in the masking order d, the number of random bytes as well as the area are $\mathcal{O}(d^2)$, being the size of the AES S-box dominated by the `shamul` module, whose area and randomness requirements grow quadratically in the masking order d.

5. EM FIELD SIDE-CHANNEL ANALYSIS

Electro-Magnetic (EM) field side-channel analysis is particularly relevant in those cases where modifying the target device to insert a shunt resistor might permanently damage it, or it would require time and expertise, e.g. unaccessible contacts due to multi-layers PCB boards or ICs with multiple power and ground pins. In the context of polynomial masking schemes, EM analysis becomes also particularly interesting in order to verify whether the 2nd-order univariate leakage exhibited in [17] can also be exploited in the EM domain, i.e. it is not caused by the load at the measurement point on the target device. We synthesized the design using Xilinx ISE Webpack version 14.7 targeting a Xilinx Spartan-6 FPGA (`xc6slx9-2-ftg256`). We used an Agilent DSO9254A 2.5 GHz digital oscilloscope with a 10 mm diameter magnetic field probe connected to a 30 dB pre-amplifier (namely, `RF-R 50-1` and `PA303` from Langer EMV-Technik) to emulate the worst-case scenario analysis presented in [17]. However, differently from their work, we did not employ any DC blocker, as our EM probe is active in the frequency range 30 MHz−3 GHz. The oscilloscope sampled at 100 MSa/s[3] and we clocked our design at 4 MHz, resulting in 25 Sa/cycle for all our experiments (where not otherwise specified). We focused our analysis on the first two S-box operations and applied 1^{st}- and 2^{nd}-order side-channel correlation-collision

[3]We did not observe more leakages when sampling at higher sampling rates.

Table 1: AES S-box: Speed [Clock Cycles] vs. Random [Bytes] vs. Area [kGE]

	$d=1$	$d=2$	$d=3$	$d=4$	$d=5$	$d=6$
Speed	105	175	245	315	385	455
Random	51	170	357	612	935	1326
Area	10.2	33.4	79.7	157.8	276	442.7

attacks [18], while fixing the security level at $10,000,000$ measurements (where not otherwise specified).

For the sake of evaluation, we fixed the public points $(\alpha_i)_{1\leq i\leq m}$ and $(\lambda_i)_{1\leq i\leq m}$ to (`0x02`, `0x03`, `0x04`) and (`0x02`, `0xD1`, `0xD2`) for $d = 1$ and to (`0x02`, `0x03`, `0x04`, `0x0A`, `0x0B`) and (`0x64`, `0x94`, `0xAE`, `0x9F`, `0xC0`) for $d = 2$, respectively. Note that, this is again a worst-case assumption, as the public points are not required to be fixed in practice. Assuming that the points are uniformly and independently drawn at random, the probability that the adversary can observe the same public points is $\prod_{i=0}^{m-1} \frac{1}{(2^8-1-i)}$. This means that it would be much harder to mount successful attacks in practice, especially for $d > 1$, as the probability decreases quickly with the masking order d.

For our side-channel investigations, we proceeded as follows: (1) we considered a 1^{st}-order implementation and verified the existence of 1^{st}-order univariate leakage, when the masks are off; (2) we activated the PRNG to generate the masks and confirm the existence of 2^{nd}-order univariate leakage in the EM domain; (3) we activated the shuffling countermeasure to verify its effectiveness in 1^{st}-order implementations; (4) we verified the security of a 2^{nd}-order implementation without shuffling.

We conclude that our 1^{st}-order implementation with shuffling and our 2^{nd}-order implementation achieve univariate 1^{st}- and 2^{nd}-order security up to $10,000,000$ traces in a worst-case scenario analysis.

1^{st}-*Order Masking and PRNG Off.*

As a sanity check, we verified the existence of 1^{st}-order univariate leakage in our 1^{st}-order implementation, when all masks are set to zero. The results are reported in Figure 8, which clearly shows how the EM activity of the two S-boxes perfectly correlates over time, when the masks are off. This analysis was conducted with only $1,000,000$ measurements.

1^{st}-*Order Masking, PRNG On and Shuffling Off.*

The results of the analysis of the 1^{st}- and 2^{nd}-order univariate leakage, when the shuffling countermeasure is deactivated, is reported in Figure 6. Even though the `shamul` and `shamac` module are active at the same time, no leakage can be observed in the 1^{st}-order moment. However, a significant leakage can be observed in the 2^{nd}-order moment during both the inversion and the affine transformation. Interestingly, the leakage captured in the 2^{nd}-moment corresponds to the whole leakage available in the traces, as confirmed by mutual information analysis[4], i.e. showing that no other source of leakage is present in the traces. We repeated the same attacks by clocking the target device with increasing clock frequencies. The results are illustrated in

[4]Estimated using histograms.

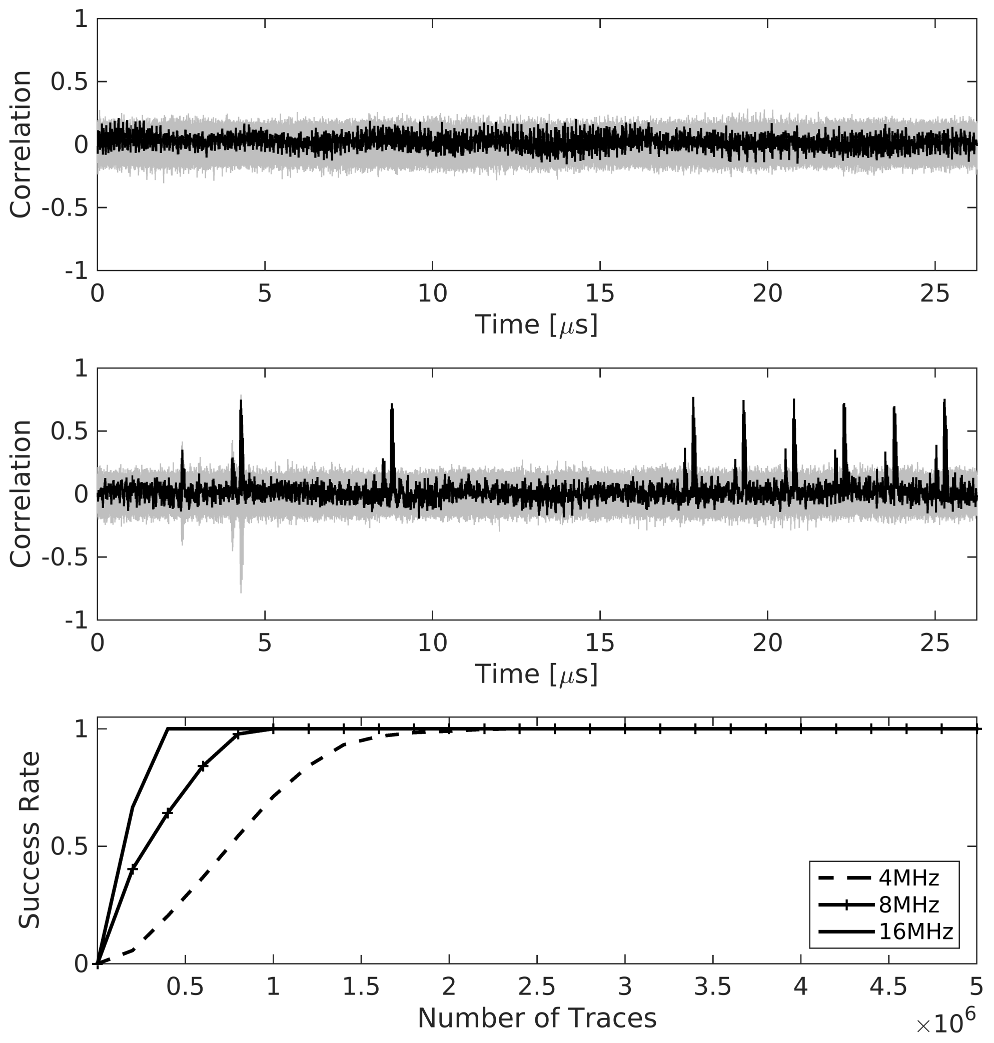

Figure 6: 1^{st}-Order Masking, PRNG On, Shuffling Off, 1^{st}-Order (top) and 2^{nd}-Order Univariate Attacks (middle), Success Rate for various clock frequencies (bottom), $10,000,000$ Traces.

Figure 6 (bottom) and show that the efficiency of 2^{nd}-order attacks increases with the clock frequency, confirming the claims of [17] also in the EM domain.

1^{st}-*Order Masking, PRNG On and Shuffling On.*

The results of the analysis of the 1^{st}- and 2^{nd}-order univariate leakage, when the shuffling countermeasure is on, is reported in Figure 7. Interestingly, no univariate leakage can be observed in any of the first four moments at the considered security level. Also, mutual information analysis reveals no any other leakage source in the traces. It is worth noting that shuffling the activation order of the shares is particularly effective against side-channel collision attacks, as each operation is independently shuffled, thus making collisions much harder to match and detect in practice. However, it must be also noted that the leakages are only hidden by the shuffling countermeasures, thus some leakage is expected to emerge either by increasing the number of measurements or by using more sophisticated combined attacks. Nevertheless, the univariate security is guaranteed by our experiments up to $10,000,000$ traces in a worst-case scenario analysis with fixed public points, thus providing a *significant* security improvement at negligible area costs.

2^{nd}-*Order Masking, PRNG On and Shuffling Off.*

Finally, the results of the analysis of the 1^{st}- and 2^{nd}-order univariate leakage, when the shuffling countermeasure is off, for a 2^{nd}-order implementation is reported in Figure 9. As for the previous case, it is not possible to observe any leakage in the 1^{st} and 2^{nd} order, but also no other leakage is detected by either higher-moments (up to the fourth) nor using mutual information, for the considered security level.

6. CONCLUSION

In this work, we provided new principles for the selection of more efficient addition chains, which lead to faster and more compact S-box implementations. We proposed a lightweight shuffling countermeasure, which randomly permutes the activation order of the shares within polynomial masking schemes, and that effectively *increases* their univariate security in practice. Finally, we provided the design

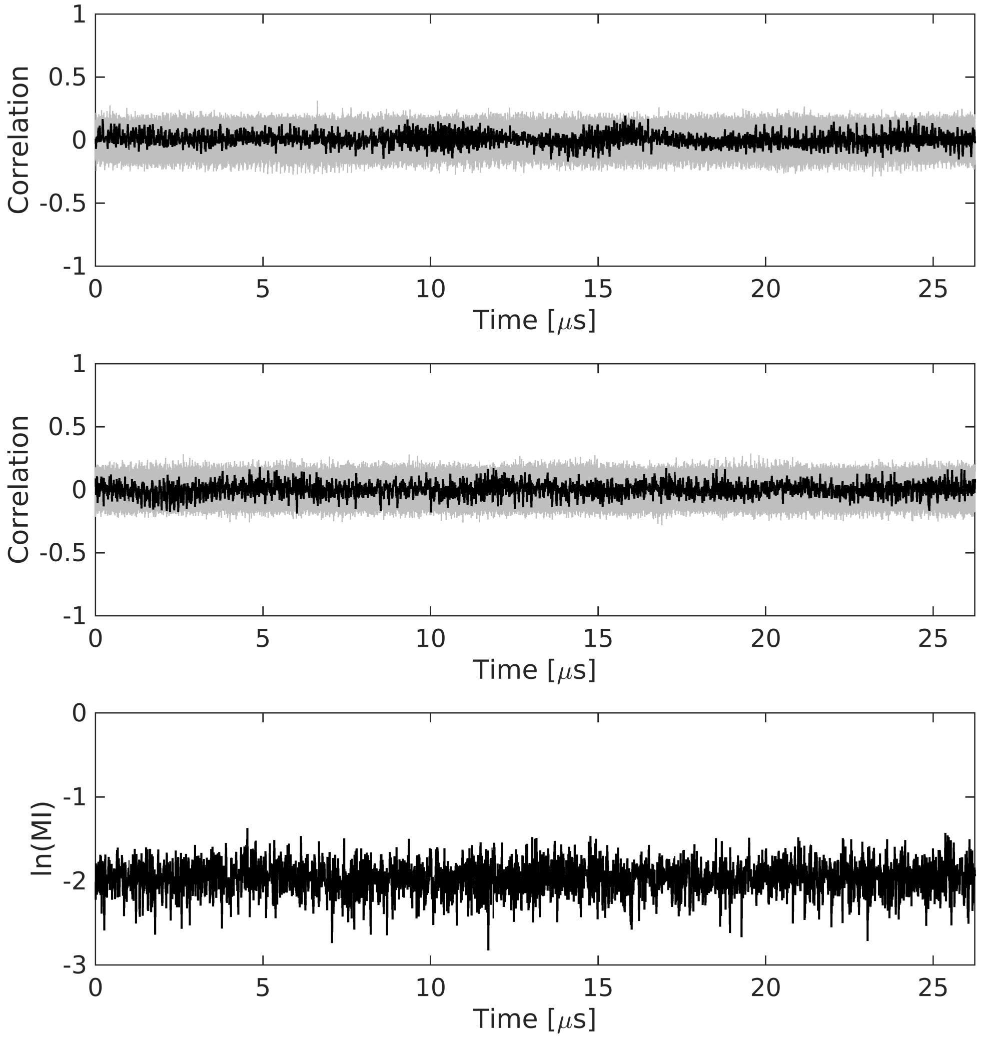

Figure 7: 1^{st}-Order Masking, PRNG On, Shuffling On, 1^{st}-Order (top) and 2^{nd}-Order Univariate Attacks (middle), Mutual Information (bottom), $10,000,000$ Traces.

and synthesis results of a more efficient, shuffled and higher-order polynomial masked AES S-box, showing that univariate security up to $10,000,000$ traces can be achieved either by a shuffled 1^{st}-order implementation, or by a 2^{nd}-order implementation, in a pessimistic worst-case scenario analysis. In particular, our 1^{st}-order implementation requires 105 clock cycles and costs 10.2 kGE, while our 2^{nd}-order implementation requires 175 clock cycles and costs 33.4 kGE.

7. ACKNOWLEDGMENTS

This work was partly funded by the German Federal Ministry of Education and Research (BMBF) in the project SIBASE under grant number 01IS13020A.

8. REFERENCES

[1] S. Belaid, F. De Santis, J. Heyszl, S. Mangard, M. Medwed, J.-M. Schmidt, F.-X. Standaert, and S. Tillich. Towards fresh re-keying with leakage-resilient PRFs: cipher design principles and analysis. *JCEN*, pages 1–15, 2014.

[2] M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, STOC '88, pages 1–10, New York, NY, USA, 1988. ACM.

[3] B. Bilgin, B. Gierlichs, S. Nikova, V. Nikov, and V. Rijmen. *Advances in Cryptology – ASIACRYPT 2014: 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, R.O.C., December 7-11, 2014, Proceedings, Part II*, chapter Higher-Order Threshold Implementations, pages 326–343. Springer Berlin Heidelberg, Berlin, Heidelberg, 2014.

[4] B. Bilgin, B. Gierlichs, S. Nikova, V. Nikov, and V. Rijmen. A more efficient aes threshold implementation. In D. Pointcheval and D. Vergnaud, editors, *Progress in Cryptology - AFRICACRYPT 2014*, volume 8469 of *Lecture Notes in Computer Science*, pages 267–284. Springer International

Publishing, 2014.

[5] A. Brauer. On addition chains. *Bull. Amer. Math. Soc.*, 45(10):736–739, 10 1939.

[6] S. Chari, C. Jutla, J. R. Rao, and P. Rohatgi. A cautionary note regarding evaluation of aes candidates on smart-cards. In *In Second Advanced Encryption Standard (AES) Candidate Conference*, 1999.

[7] S. Chari, C. S. Jutla, J. R. Rao, and P. Rohatgi. Towards sound approaches to counteract power-analysis attacks. In M. J. Wiener, editor, *CRYPTO*, volume 1666 of *Lecture Notes in Computer Science*, pages 398–412. Springer, 1999.

[8] J. Daemen and V. Rijmen. *The Design of Rijndael*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2002.

[9] T. De Cnudde, B. Bilgin, O. Reparaz, V. Nikov, and S. Nikova. *Smart Card Research and Advanced Applications: 14th International Conference, CARDIS 2015, Bochum, Germany, November 4-6, 2015. Revised Selected Papers*, chapter Higher-Order Threshold Implementation of the AES S-Box, pages 259–272. Springer International Publishing, Cham, 2016.

[10] T. De Cnudde, B. Bilgin, O. Reparaz, and S. Nikova. *Cryptography and Information Security in the Balkans: First International Conference, BalkanCryptSec 2014, Istanbul, Turkey, October 16-17, 2014, Revised Selected Papers*, chapter Higher-Order Glitch Resistant Implementation of the PRESENT S-Box, pages 75–93. Springer International Publishing, Cham, 2015.

[11] C. Dobraunig, F. Koeune, S. Mangard, F. Mendel, and F.-X. Standaert. *Towards Fresh and Hybrid Re-Keying Schemes with Beyond Birthday Security*, pages 225–241. LNCS. Springer, 2015.

[12] R. Durstenfeld. *Algorithm 235: Random Permutation*, page 420. ACM, 1964.

[13] H. Kim, S. Hong, and J. Lim. A fast and provably secure higher-order masking of aes s-box. In *Proceedings of the 13th International Conference on Cryptographic Hardware and Embedded Systems*, CHES'11, pages 95–107, Berlin, Heidelberg, 2011. Springer-Verlag.

[14] S. Mangard, T. Popp, and B. M. Gammel. Side-channel leakage of masked CMOS gates. In A. Menezes, editor, *Topics in Cryptology - CT-RSA 2005, The Cryptographers' Track at the RSA Conference 2005, San Francisco, CA, USA, February 14-18, 2005, Proceedings*, volume 3376 of *Lecture Notes in Computer Science*, pages 351–365. Springer, 2005.

[15] M. Medwed, F.-X. Standaert, J. Großschädl, and F. Regazzoni. Fresh Re-keying: Security Against Side-channel and Fault Attacks for Low-cost Devices. In *AFRICACRYPT'10*, pages 279–296, Berlin, Heidelberg, 2010. Springer-Verlag.

[16] A. Moradi, M. Kirschbaum, T. Eisenbarth, and C. Paar. Masked dual-rail precharge logic encounters state-of-the-art power analysis methods. *IEEE transactions on VLSI systems*, 20:1578 – 1589, 2012.

[17] A. Moradi and O. Mischke. On the simplicity of converting leakages from multivariate to univariate. In G. Bertoni and J.-S. Coron, editors, *Cryptographic Hardware and Embedded Systems - CHES 2013*, volume 8086 of *Lecture Notes in Computer Science*, pages 1–20. Springer Berlin Heidelberg, 2013.

[18] A. Moradi, O. Mischke, and T. Eisenbarth. *Cryptographic Hardware and Embedded Systems, CHES 2010: 12th International Workshop, Santa Barbara, USA, August 17-20, 2010. Proceedings*, chapter Correlation-Enhanced Power Analysis Collision Attack, pages 125–139. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[19] A. Moradi, A. Poschmann, S. Ling, C. Paar, and H. Wang. Pushing the limits: A very compact and a threshold implementation of aes. In *Advances in Cryptology - EUROCRYPT 2011 - 30th Annual International Conference on the Theory and Applications of Cryptographic Techniques*, volume 6632 of *Lecture Notes in Computer Science*, page 69. Springer, 2011.

[20] T. Popp, M. Kirschbaum, T. Zefferer, and S. Mangard. Evaluation of the masked logic style mdpl on a prototype chip. In *Cryptographic Hardware and Embedded Systems - CHES 2007*, volume 4727 of *LNCS*, pages 81 – 94. Springer, 2007.

[21] E. Prouff and M. Rivain. *Advances in Cryptology – EUROCRYPT 2013: 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings*, chapter Masking against Side-Channel Attacks: A Formal Security Proof, pages 142–159. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[22] 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, *Cryptographic Hardware and Embedded Systems - CHES 2011 - 13th International Workshop, Nara, Japan, September 28 - October 1, 2011. Proceedings*, volume 6917 of *Lecture Notes in Computer Science*, pages 63–78. Springer, 2011.

[23] M. Rivain and E. Prouff. Provably secure higher-order masking of AES. In S. Mangard and F. Standaert, editors, *Cryptographic Hardware and Embedded Systems, CHES 2010, 12th International Workshop, Santa Barbara, CA, USA, August 17-20, 2010. Proceedings*, pages 413–427. Springer, 2010.

[24] T. Roche and E. Prouff. Higher-order glitch free implementation of the AES using secure multi-party computation protocols - extended version. *J. Cryptographic Engineering*, 2(2):111–127, 2012.

[25] A. Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, Nov. 1979.

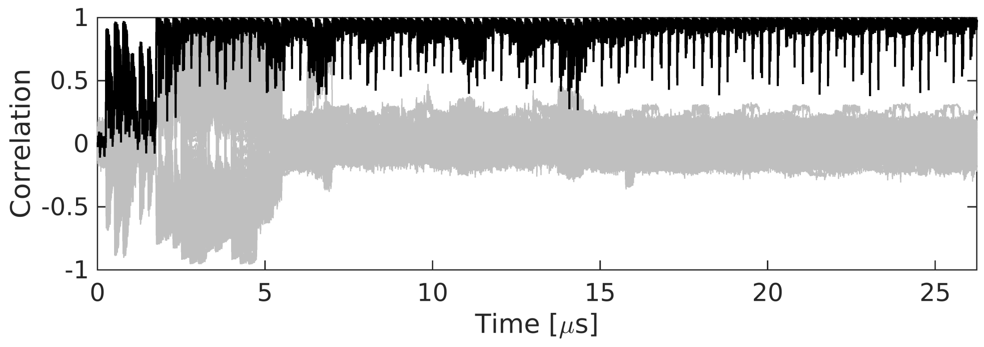

Figure 8: 1^{st}**-Order Masking, PRNG Off, Shuffling Off,** 1^{st}**-Order Univariate Attacks,** $1,000,000$ **Traces.**

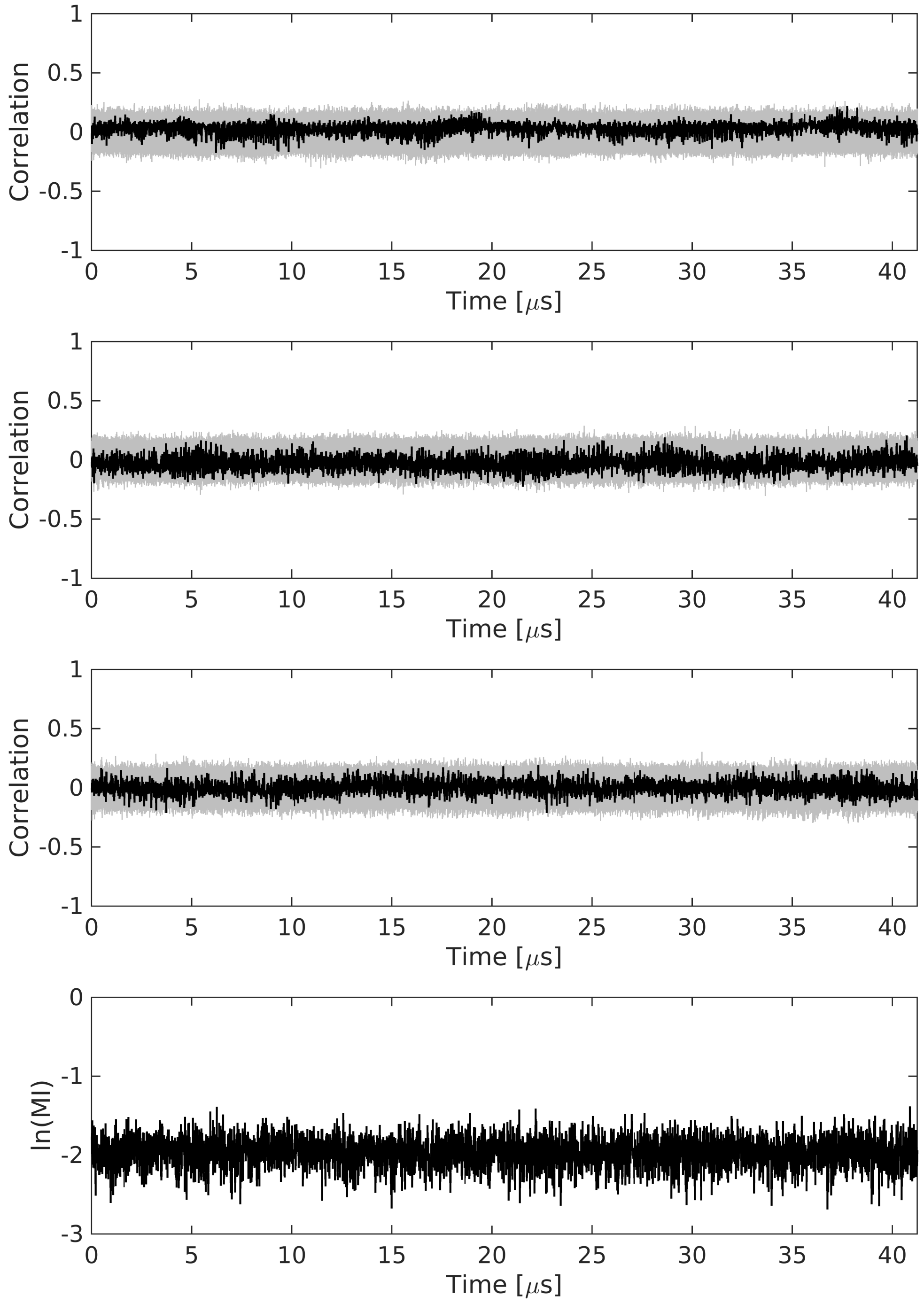

Figure 9: 2^{nd}**-Order Masking, PRNG On, Shuffling Off,** 1^{st}**-Order (top),** 2^{nd}**-Order and** 3^{rd}**-Order Univariate Attacks (middle), Mutual Information (bottom),** $10,000,000$ **Traces.**