Self-Timed Masking: Implementing Masked S-Boxes Without Registers

Mateus Simões[1,2], Lilian Bossuet[1], Nicolas Bruneau[2], Vincent Grosso[1], Patrick Haddad[2], and Thomas Sarno[2]

[1] Laboratoire Hubert Curien, CNRS UMR5516, Saint-Étienne, France
[2] STMicroelectronics, Rousset, France

Abstract. Masking is one of the most used side-channel protection techniques. However, a secure masking scheme requires additional implementation costs, e.g. random number, and transistor count. Furthermore, glitches and early evaluation can temporally weaken a masked implementation in hardware, creating a potential source of exploitable leakages. Registers are generally used to mitigate these threats, hence increasing the implementation's area and latency.
In this work, we show how to design glitch-free masking without registers with the help of the dual-rail encoding and asynchronous logic. This methodology is used to implement low-latency masking with arbitrary protection order. Finally, we present a side-channel evaluation of our first and second order masked AES implementations.

Keywords: Side-channel analysis · Masking · Asynchronous circuits.

1 Introduction

Different techniques exist to counter side-channel attacks; one of the most studied is the Boolean masking [7, 11, 14], which splits a sensitive variable into several shares. In this manner, a secret x is d^{th} order masked with $d+1$ shares as shown in Eq. (1), with $(x_0, x_1, \ldots, x_{d-1})$ the random shares and x_d the masked value. The $\oplus$ symbol denotes the XOR operation.

$$x_d = x_0 \oplus x_1 \oplus \ldots \oplus x_{d-1} \oplus x \tag{1}$$

Thereupon, instead of manipulating the plain data, the circuit performs computation on the shares. This results in a more complex relationship between the side-channel leakages and the sensitive data. At the appropriate moment, the shares can be recombined to uncover the secret data, that is, $x = \bigoplus_{i=0}^{d} x_i$. Note that security comes at the cost of higher implementation complexity, raising the transistor count.

The circuit needs to be transformed to perform the desired computation of the plain data while manipulating the shares. Securely masking a linear function is trivial, since each input share can be manipulated independently and in parallel. For instance, let $z = f(x, y)$ be a linear Boolean operation, its d^{th} order masking can be expressed as shown in the Eq. (2).

$$z = \bigoplus_{i=0}^{d} z_i = \bigoplus_{i=0}^{d} f_i(x_i, y_i) = f\left(\bigoplus_{i=0}^{d} x_i, \bigoplus_{i=0}^{d} y_i\right) = f(x, y) \tag{2}$$

On the other hand, the masking of a non-linear function, such as inversion in $\mathbb{F}_{2^n}$, manipulates the sharing in such a way that its intermediate terms require the recombination of several shares of a variable. Moreover, the sharing recombination may be the source of exploitable side-channel leakages, rendering masking of the non-linear operations a critical task for security engineers. In this context, techniques such as threshold implementations (TI) [24] were proposed.

TI limits the share recombination leakages. In this manner, the occurrence of glitches is an important factor to take into account when implementing the masking scheme in hardware. In fact, a glitchy function has an unexpected behavior that may be correlated to the unshared variable [18] — or more broadly to several shares. To guarantee the security of non-linear functions in the presence of glitches, register barriers can be employed to cease the spurious signal propagations [27], thus increasing the overall latency of the masked function.

Theoretical analysis can be used to strengthen confidence in the protection brought by the masked implementation. To evaluate the security of a masking scheme, several methods based on the probing model of Ishai et al. [14] exist. In this security evaluation method, the adversary can place up to d probes on different wires of the circuit in order to obtain their instantaneous logic level, providing clues about a potential dependence between the unshared value and the internal signal states. This model was enhanced to take into account physical defaults such as glitches [9] and composability [2].

Satisfying security in those security models requires additional overhead such as register layers, fresh random bits and higher silicon area. Therefore, reducing the masking costs is a pertinent branch of research in side-channel countermeasures. In this context, this work aims at reducing the number of clock cycles needed to compute masked functions. For that, we present a self-timed masking implementation built upon the Muller c-element [22] latches. We show how to replace registers with those latches, assuring data synchronization among different combinatorial layers. Furthermore, we present our locally asynchronous globally synchronous (LAGS) AES design. Finally, we evaluate the side-channel leakages based on experimental measurements up to the second-order protection.

2 Background

The use of asynchronous methodologies and dual-rail logic to implement low-latency masking was first introduced by Moradi and Schneider in [21]. They designed fully unrolled first-order threshold implementations of PRINCE and Midori based on WDDL gates [30]. Later, Sasdrich et al. [28] used the LUT-based Masked Dual-Rail with Pre-charge Logic (LMDPL) [17] masking scheme to implement a low-latency AES, which is limited to first-order security. More

recently, Nagpal et al. [23] presented a low-latency domain-oriented implementation [13] also built upon WDDL gates, but employing Muller c-elements as synchronization modules, whose results have shown to be higher-order secure.

Similarly to these works, we aim at the study of low-latency masking, using dual-rail encoding with pre-charge logic and Muller c-elements to implement masked S-boxes with arbitrary protection order. Relying on the domain-oriented masking (DOM) [13], we focus on presenting and evaluating a generic methodology to replace registers with self-timed latches to design effective single-cycle masked functions.

2.1 Notations

We denote binary random variables in $\mathbb{F}_2$ with lower-case letters, e.g. $\mathtt{x}$. A random variable x is Boolean masked with $d+1$ shares x_i, whose sharing is denoted with calligraphic fonts — e.g., $\mathcal{S} = (x_0, x_1, \ldots, x_d)$ — in such a manner that $x = \bigoplus_{i=0}^{d} x_i$.

We use typewriter fonts to denote binary random variables $\mathtt{x}$, vectors $\mathtt{X}$ and signals encoded in the dual-rail protocol with a pair of wires `(x.t,x.f)`. The wire `x.t` is used for signalling $\mathtt{x.t} = x$ while `x.f` signalizes the complement $\mathtt{x.f} = \overline{x}$. A dual-rail token of a variable x is then referred as $\overset{*}{\mathtt{x}} = (\mathtt{x.t}, \mathtt{x.f}) = (x, \overline{x})$.

2.2 The domain-oriented masking

This work relies on the domain-oriented masking (DOM) [13], a known secure arbitrary order masking scheme. Since their gadget has already been formally verified in the original paper, we do not present theoretical proofs of security in this work. In order to satisfy d-glitch-extended probing security [9], their gadget is divided into two register-isolated steps, which we identify, in this work, as processing and compression.

Let us take the 2-share DOM-*indep* gadget $\mathcal{Z} = \mathcal{A} \wedge \mathcal{B}$ with $\mathcal{A} = (\mathsf{a}_0, \mathsf{a}_1)$ and $\mathcal{B} = (\mathsf{b}_0, \mathsf{b}_1)$ the input shares and $\mathcal{Z} = (\mathsf{z}_0, \mathsf{z}_1)$ the output sharing. In short, assuming that the input sharings are uniform, we want to find a secure way to compute $(\mathsf{z}_0 \oplus \mathsf{z}_1) = (\mathsf{a}_0 \oplus \mathsf{a}_1) \wedge (\mathsf{b}_0 \oplus \mathsf{b}_1)$. Hence, the process step computes the product terms $\mathsf{a}_0\mathsf{b}_0$, $\mathsf{a}_0\mathsf{b}_1$, $\mathsf{a}_1\mathsf{b}_0$, $\mathsf{a}_1\mathsf{b}_1$ and adds a fresh random share r to the cross-domain ones, that is, $\mathsf{a}_0\mathsf{b}_1$ and $\mathsf{a}_1\mathsf{b}_0$. Then, to assure non-completeness, registers ($\longrightarrow$) store the resulting shares $(\mathsf{x}_0, \mathsf{x}_1, \mathsf{x}_2, \mathsf{x}_3)$, as we can see in the Eq. (3).

$$\begin{aligned} f_0(\mathsf{a}_0, \mathsf{b}_0) &= \mathsf{a}_0\mathsf{b}_0 &&\longrightarrow \mathsf{x}_0 \\ f_1(\mathsf{a}_0, \mathsf{b}_1) &= \mathsf{a}_0\mathsf{b}_1 \oplus \mathsf{r} &&\longrightarrow \mathsf{x}_1 \\ f_2(\mathsf{a}_1, \mathsf{b}_0) &= \mathsf{a}_1\mathsf{b}_0 \oplus \mathsf{r} &&\longrightarrow \mathsf{x}_2 \\ f_3(\mathsf{a}_1, \mathsf{b}_1) &= \mathsf{a}_1\mathsf{b}_1 &&\longrightarrow \mathsf{x}_3 \end{aligned} \tag{3}$$

The processing step produces four shares. To reduce the number of output shares, there exists the compression step, as shown in the Eq. (4). Thanks to the

register barrier between both steps and the fresh randomness, 1-glitch-extended security is satisfied.

$$\begin{aligned} z_0 &= x_0 \oplus x_1 \\ z_1 &= x_2 \oplus x_3 \end{aligned} \tag{4}$$

Based on the domain-oriented scheme, Gross et al. proposed the first generic low-latency masking (GLM) in [12]. In their work, they skip the compression step after the non-linear operations, eliminating the registers after the shared processing. However, the number of shares grows quadratically after each masked multiplication. In consequence, the area and randomness costs increase substantially. In our work, we maintain the compression step and use a dual-rail synchronization element to obtain a generic low-latency masking.

2.3 The dual-rail encoding

The dual-rail protocol encodes a bit using two signal wires: a wire `x.t` carries the logic value of a variable x while a second wire `x.f` transports its complement [8]. In this configuration, a valid token is obtained when one, and only one, signal wire is active (i.e. in a high logic state), although the null token is encoded when both wires are deactivated, that is, $\varnothing = (0, 0)$. The encoding $(1, 1)$ is never used, and the behavior of our design after the injection of this invalid token is out of scope of this work. Table 1 summarizes the dual-rail encoding.

Table 1. The dual-rail encoding.

Data	Token	
	`x.t`	`x.f`
null	0	0
$x = 0$	0	1
$x = 1$	1	0
not used	1	1

Moradi and Schneider presented the first work that borrowed asynchronous dual-rail techniques with the purpose of implementing low-latency masking [21]. Different from our choice of design, they designed a fully unrolled threshold implementation of two lightweight block-ciphers PRINCE and Midori using WDDL cells.

In contrast, we opted for an LAGS design, creating a single-cycle S-box within a synchronous AES architecture. The dual-rail encoding was also present in the implementation proposed by Sasdrich et al. [28] to create a first-order secure low-latency AES based on the LMDPL masking scheme [17]. Similar to the LMDPL, we employ the pre-charge / evaluation logic with monotonic functions to obtain a glitch-free circuit [25]. To eliminate the glitches, only regular AND and OR

gates are used to construct our dual-rail functions, due to their monotonic behavior [15].

We refer to Eqs. (5) and (6) for the AND and XOR functions, respectively, used in our work. We use the DPL_noEE AND gate [4], shown in Eq. (5), instead of the WDDL AND [30] to avoid the early propagation in the evaluation phase [16, 19].

$$\begin{aligned} \mathtt{z} = \mathtt{a} \wedge \mathtt{b} \iff \mathtt{z.t} &= \mathtt{a.t} \wedge \mathtt{b.t} \\ \mathtt{z.f} &= (\mathtt{a.t} \wedge \mathtt{b.f}) \vee (\mathtt{a.f} \wedge \mathtt{b.t}) \vee (\mathtt{a.f} \wedge \mathtt{b.f}) \end{aligned} \tag{5}$$

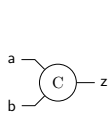
$$\begin{aligned} \mathtt{z} = \mathtt{a} \oplus \mathtt{b} \iff \mathtt{z.t} &= (\mathtt{a.t} \wedge \mathtt{b.f}) \vee (\mathtt{a.f} \wedge \mathtt{b.t}) \\ \mathtt{z.f} &= (\mathtt{a.f} \wedge \mathtt{b.f}) \vee (\mathtt{a.t} \wedge \mathtt{b.t}) \end{aligned} \tag{6}$$

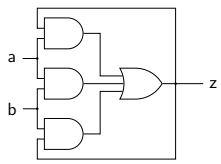
Encoded as dual-rail tokens, the information in the communication channel carries the data itself and the validity signal. For instance, the output token of a dual-rail logic gate is valid when $\mathtt{z.f} \vee \mathtt{z.t} = 1$, supposing $\mathtt{z.f} \neq \mathtt{z.t}$ to avoid the illegal case. Thus, the validity signal can be used to control the flow of tokens. Based on this idea, the data synchronization is managed by the tokens, eliminating the need of register layers, as we will explain in the next subsection.

2.4 Data synchronization with the Muller c-elements

Registers are important components in hardware masking due to their role in synchronizing the boundaries of different combinatorial blocks [27, 9]. However, although limiting the combinatorial data path, registers increase the overall latency by requiring additional clock cycles to process the whole circuit.

In this work, we use an alternative state-holding element to obtain single-cycle S-box implementations. The state-holding module used in this work is built upon the Muller c-element [22], whose symbol is shown in Figure 1 alongside with a gate-level implementation and a summary of its logical behavior. The Muller c-element operates as a Boolean function $f(\mathtt{a}, \mathtt{b}) = \mathtt{a} \circ \mathtt{b}$ that outputs 0 when all inputs have a low logic level, and when all inputs have a high logic level it outputs 1. Otherwise, the Muller c-element maintains its current steady state if at least one input is different from the others.





a	b	$\mathtt{z} = \mathtt{a} \circ \mathtt{b}$
0	0	0
0	1	$\mathtt{z}_{-1}$
1	0	$\mathtt{z}_{-1}$
1	1	1

Fig. 1. A Muller c-element symbol (left), a gate-level design (middle) and its truth table (right).

Recently, Nagpal et al. [23] presented a domain-oriented gadget built upon Muller c-elements and WDDL gates to create a generic single-cycle masking in hardware that is higher-order secure. We also use Muller cells, but with the purpose of designing self-timed latches to replace the register layers in any masking implementation, easing the application of our method in different scenarios. Moreover, we do not employ WDDL gates as they are vulnerable due to early propagation leakages [16].

Figures 2 and 3 show the 2-bit wide self-timed latches used in our designs. The dual-rail latches can be characterized as either strongly indicating or weakly indicating, depending on how their acknowledgement signal is computed. Figure 2, waits for all of its inputs to become valid, or null, before sending the respective acknowledgement. In contrast, a weakly indicating latch, Figure 3, waits for only one specific input token to become valid or null before authenticating its current state [29].

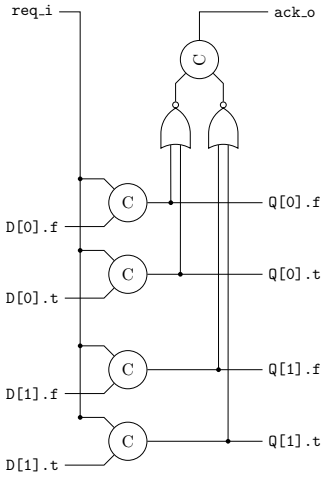


Fig. 2. A 2-bit wide strongly indicating asynchronous latch.

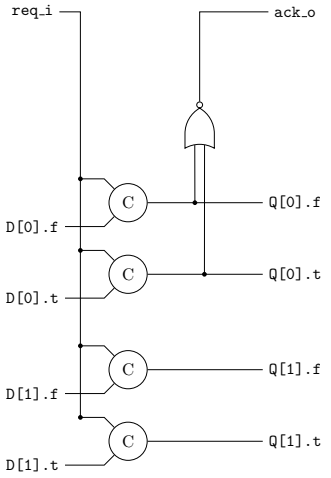


Fig. 3. A 2-bit wide weakly indicating asynchronous latch.

In both cases, weakly or strongly indicating, n pairs of Muller c-elements store a n-bit token $\overset{*}{\mathtt{x}}$ and a regular 2-input NOR gate computes the validity signal for each pair — or a single pair for the weakly indicating version — to obtain the correspondent acknowledgement signal state.

The handshake logic contains two signals: a request input, denoted `req_i`, and an acknowledgement output, identified as `ack_o`. In fact, the acknowledgement signal indicates when the latch stores a valid (`ack_o` $= 0$) or a null (`ack_o` $= 1$) token. The request signal paces the data flow and is connected to one of the Muller c-element inputs. Thus, `req_i` $= 0$ requests a null token (i.e., the pre-charge), while `req_i` $= 1$ means that the combinatorial block fol-

lowing the latch is ready to evaluate a new valid token. Figure 4 shows the functioning of the handshake logic.

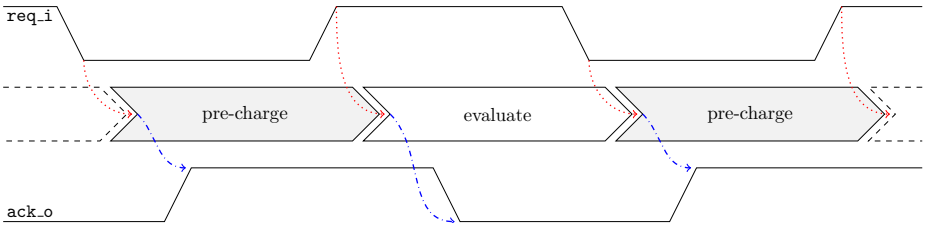


Fig. 4. Self-timed handshake in a pre-charge / evaluate logic.

In our designs, we favor the weakly indicating version based on three aspects.

1. *Speed*: since a single bit triggers the acknowledgement signal, the handshake logic depth is lower.
2. *Area*: a single 2-input NOR gate is used to compute the acknowledgement signal, reducing the total silicon area.
3. *Security*: a single bit triggers the acknowledgement, instead of the whole word, mitigating data dependent evaluation time leakages.

The latch is "self-timed" due to its handshake logic, that is managed by the data itself, excluding the need of a clock signal to pace the token flow. In this context, the data streams like a wave, with the intermediate states oscillating between null and valid tokens, configuring what is known as pre-charge logic [25].

To illustrate the operation of a self-timed circuit, consider the following two-stage pipeline, Figure 5, in which C denotes a combinatorial circuit. For ease of visualization, ★ represents a random valid token and ∅ denotes the null token. There are two latches (A) and (B) in this example, whose initial states are, respectively, ∅ and ★. The `req_i` of the (A) is connected to the `ack_o` of (B). This wire is identified as `ack_s`.

Considering that C_A is pre-charged, the valid token is processed once it arrives at the pipeline input. The latch (A) keeps its logic state since its `ack_s` $= 0$. When `req_i` switches to 0, (B) absorbs the ∅ from (A) and sets `ack_s` $= 1$; The pre-charge phase of C_B is complete, which is signalized by (B) setting `ack_s` $= 1$. In consequence, (A) absorbs the valid token $\mathsf{C}_A(★)$.

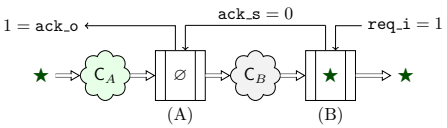


Fig. 5. Initial state.

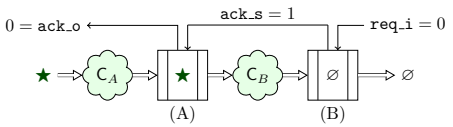


Fig. 6. `req_i` switches to 0.

The external circuit issues a null token, pre-charging the combinatorial circuit C_A; the latches (A) and (B) stand by, since the request input remains constant at `req_i` $= 0$. Next, the `req_i` switches to 1, triggering the absorption of the output valid token $\mathsf{C}_B(\mathsf{C}_A(\bigstar))$, completing one self-timed processing cycle. This absorption sets the `ack_s` $= 0$.

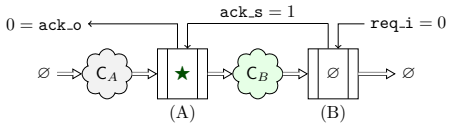


Fig. 7. The pre-charge of C_A.

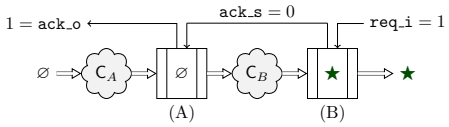


Fig. 8. `req_i` switches to 1.

Although being a generic example, we aim at obtaining the same behavior for the AES S-box. Thus, in the following sections, we show how this can be implemented and discuss the implementation results for this solution.

3 Self-timed masking implementation

This section presents the design of a self-timed AES S-box based on the *simple* variant of the domain-oriented masking (DOM) proposed by Gross et al. in [13]. We start from a known implementation whose security performance has been already assured in the original work. Thus, we can study the resulting overheads when enabling self-timed features and evaluate first and higher-order security performance against side-channel analysis.

The d^{th} order DOM gadget has $(d+1)^2$ registers, which will be replaced by dual-rail latches with the purpose of enabling the aforementioned self-timed features. To illustrate, consider the resulting 1^{st} order DOM gadget shown in Figure 9, in which all logic gate symbols represent their dual-rail variant.

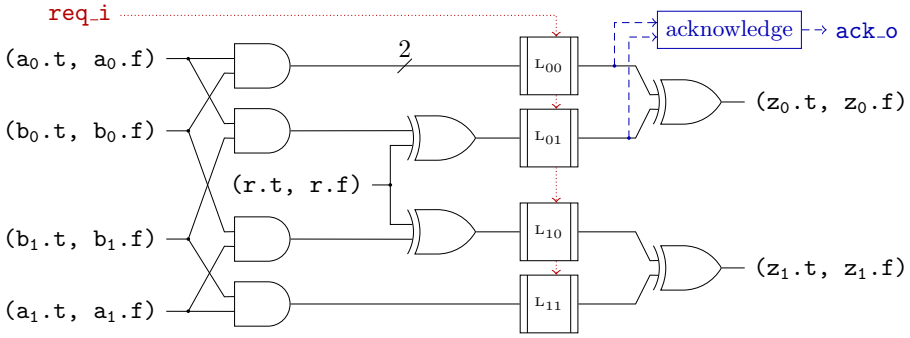


Fig. 9. The 1^{st} order self-timed DOM gadget.

The request input is common to all latches and the Eq. 7 shows the complete detection logic for the general case, with the symbol $\circ$ denoting the Muller c-element operation. In this manner, the acknowledgement signal indicates the token state stored in the current latch.

$$\texttt{ack\_o} = \neg((\texttt{x}_0\texttt{.f} \vee \texttt{x}_0\texttt{.t}) \circ (\texttt{x}_1\texttt{.f} \vee \texttt{x}_1\texttt{.t}) \circ \cdots \circ (\texttt{x}_\texttt{d}\texttt{.f} \vee \texttt{x}_\texttt{d}\texttt{.t})) \tag{7}$$

Hence, for any protection order, only the first output share is used to compute the acknowledgement signal. By limiting the number of shares used to compute the validity signal, we aim at avoiding data dependent time of evaluation. Moreover, note that the more validity bits we use, the later the acknowledgement signal will be obtained, reducing the handshake speed.

Our S-box implementation, shown in the Figure 10, is based on the Canright's design [6]. Basically, we implemented the *simple* variant shown in [13] using self-timed latches instead of registers. Each multiplier outputs a single acknowledgement signal and Muller c-elements are used to link the handshake of the two last multiplication stages. Indeed, the Muller c-elements are also employed in our design to join different acknowledgement signals.

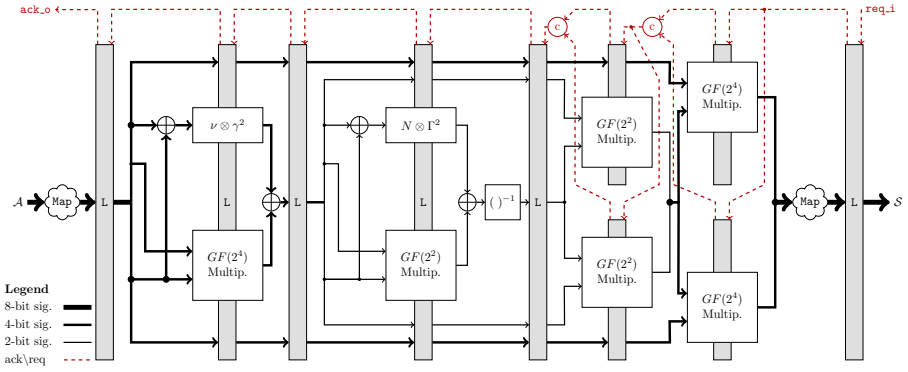


Fig. 10. Self-timed AES S-box based on the Canright's design.

We opt for the Canright's AES S-box design to evaluate our solution. However, we highlight that replacing register by self-timed latches is a generic solution, that can be applied in different implementations.

Our AES128 architecture is locally asynchronous and globally synchronous (LAGS), as we are interested in employing asynchronous techniques to obtain a single-cycle S-box. Nevertheless, a fully unrolled design can be obtained using self-timed latches.

Moreover, as the asynchronous pipeline can process several valid tokens, our AES architecture has a 32-bit single-rail data path with a single S-box. Thus, four substitution bytes are computed per clock cycle. In this context, a positive clock edge triggers the domino logic, allowing us to synchronize the computation

of the correct tokens. If the clock period is adequate, the four SubBytes are ready before the next positive edge using only one S-box.

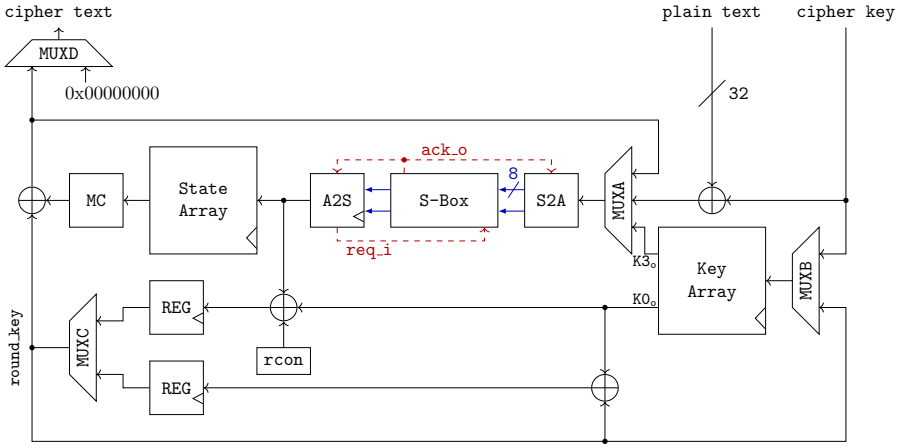


Fig. 11. The LAGS AES128 architecture with a 32-bit data path.

The interface between the synchronous and asynchronous worlds are managed by two modules: the synchronous to asynchronous (S2A) and the asynchronous to synchronous (A2S) blocks shown in the Figure 11.

The S2A block converts the 32-bit single-rail data into four 8-bit dual-rail tokens. This block issues the tokens to the S-box input with the help of a multiplexer. The S2A is also able to identify whenever the circuit requests a pre-charge token. In other words, the acknowledgement output signal of the S-box block indicates when a token has been absorbed, triggering the pre-charge or the evaluation phase, depending on the acknowledgement signal state.

In parallel, the A2S module manages the S-box output — converting the valid token back to single-rail logic — and its request input signal. We count the number of occurrences of the positive acknowledgement edge in order to track the desired token progression in the pipeline. This block contains a 32-bit wide self-timed latch to store each output token. However, this latch has four acknowledgement signals, one for each output token, in order to identify the validity of each computation. The request signals are demultiplexed to obtain a single request output. At the end of the S-box computation, the four substitution bytes are available at the S-box output and the A2S module waits for the next positive edge of the clock signal to trigger the next computation. Since the A2S module stands by until the next positive edge of the clock signal, the request signal remains constant as well as the internal states of the S-box.

Different from previous low-latency AES128 implementations, such as [28, 23], we compute an AES128 round in five cycles. This is a choice of design. As we will show in the implementation results, this design allows us to obtain a

relatively small AES architecture at the cost of higher encryption latency. Thus, to compute the AES128 encryption we need $10 \times 5 + 4 = 54$ clock cycles. Our AES architecture is based on the one by Moradi et al. [20], which relies on state and key arrays built upon shift registers.

Table 2 summarizes the control used during one encryption round of the AES, with $\texttt{SB}_\circ$, $\texttt{MC}_\circ$ and $\texttt{MUXC}_\circ$ denoting the S-box, mix columns and MUX C outputs, respectively. Also, the $\texttt{K0}_\circ$ and $\texttt{K3}_\circ$ represent the output and input columns of the key array. The rotated $\texttt{K3}_\circ$, for example, is used as the S-box input for the key scheduling. One clock cycle is used to perform the shift rows in the state array. The combinatorial mix columns operation is performed in the first four cycles of each encryption round.

Table 2. AES control during one encryption round.

Cycle	Round Key	S-box Input	State Array	Key Array
0	$\texttt{K0}_\circ \oplus \texttt{SB}_\circ \oplus$ rcon	$\texttt{MC}_\circ \oplus \texttt{MUXC}_\circ$	shift rows	round key
1	$\texttt{K0}_\circ \oplus \texttt{K3}_\circ$	$\texttt{MC}_\circ \oplus \texttt{MUXC}_\circ$	$\texttt{SB}_\circ$	round key
2	$\texttt{K0}_\circ \oplus \texttt{K3}_\circ$	$\texttt{MC}_\circ \oplus \texttt{MUXC}_\circ$	$\texttt{SB}_\circ$	round key
3	$\texttt{K0}_\circ \oplus \texttt{K3}_\circ$	$\texttt{MC}_\circ \oplus \texttt{MUXC}_\circ$	$\texttt{SB}_\circ$	round key
4	--	rotate[$\texttt{K3}_\circ$]	$\texttt{SB}_\circ$	stand by

Since we compute four S-boxes within one clock cycle, the system requires $144(d + d^2)$ refresh bits per clock cycle to protect the AES encryption, with d the masking protection order.

4 Implementation results

We use Synopsys Design Compiler in order to synthesize our design using the CMOS 40-nm standard cell library with a target frequency of 100 MHz. The area results are normalized in terms of gate equivalents (GE) with a two-input NAND gate from the selected library as reference. No `compile_ultra` scripts were used in this work. We refer to Table 3, which reports the performance figures of our self-timed S-box implementations compared to the state-of-the-art.

Among the low-latency S-boxes, Sasdrich et al. [28] present the smallest first-order design. Nevertheless, our solution can be considered competitive in terms of gate counting and allows arbitrary protection order. Compared to the design proposed by Gross et al. [12], the area and randomness overheads of our solution show a better result, thanks to the presence of the compression step after the synchronization layers.

Nagpal et al. [23] propose two AES S-box designs: the Canright's [6] AES S-box based on composite fields and the Boyar and Peralta's [5] bit-slicing approach. Comparing the composite field implementations, our self-timed masking outperforms the area requirements of the solution proposed by Nagpal et al.,

Table 3. Performance figures of different masked S-box implementations.

Design	Masking Order	Area [kGE]	Refresh [bits]	Latency [cycles]
Gross et al. [13]	1^{st}	2.6	18	8
Arribas et al. [1]	1^{st}	25.8	0	1
Gross et al. [12]	1^{st}	60.7	2048	1
Sasdrich et al. [28]	1^{st}	3.5	36	1
Nagpal et al. [23] {Canright's design}	1^{st}	7.6	18	1
Nagpal et al. [23] {Boyar and Peralta's design}	1^{st}	4.0	34	1
this work	1^{st}	6.1	36	1
Gross et al. [12]	2^{nd}	57.1	4446	1
Nagpal et al. [23] {Canright's design}	2^{nd}	14.8	51	1
Nagpal et al. [23] {Boyar and Peralta's design}	2^{nd}	9.3	102	1
this work	2^{nd}	11.4	108	1

which is not the case when comparing our implementation to to their bit-slice AES S-box. However, the DOM gadget applied to the [5] multiplication may produce a flawed masking due to share's collision leakages [12].

Since we built our designs upon DOM gadgets, one S-box computation requires $36(d+d^2)$ fresh randomness per clock cycle. However, to securely compute the AES encryption, our AES design requires $144(d+d^2)$ refresh bits, since four SubBytes are computed within one clock cycle with a single S-box. However, despite being able to compute four SubBytes in a single-cycle, our design has shown to be slow compared to other solutions. Indeed, the throughput of our first-order S-box is approximately 5.3MB/s. Hence, our self-timed masking offers a trade-off between latency and throughput to designers. Although the latency is reduced, the clocked version of the same S-box implementation would perform better in terms of operating frequency, since the synchronous circuit has a smaller critical path.

One reason for this lack of performance is the number of S-box stages in our design. In fact, the *simple* variant of the DOM AES S-box has eight stages, as shown in the Figure 10. Since the handshake signal needs to travel from the request input to the acknowledgement output to shift a token from a latch N to a latch $N+1$, the higher is the number of stages, the slower is the token flow. Moreover, the latches have to be pre-charged after evaluating a valid token, which also compromises the speed. Based on this aspect, our solution may be more suitable for S-box designs with less combinatorial stages.

Thanks to our choice of design, our AES encryption can be computed in 54 clock cycles and the gate count results are significantly lower, compared to other low-latency solutions, as shown in Table 4. In this manner, low-latency masked S-boxes built within our 32-bit AES architecture would allow the hardware designer to obtain a smaller silicon area at the cost of increasing the number of clock cycles needed to perform an AES encryption.

Table 4. Performance figures of different low-latency masked AES implementations.

Design	Masking Order	Area [kGE]	Refresh [bits]	Latency [cycles]	Area x Latency [kGE · cycles]
Sasdrich et al. [28]	1st	157.5	976	11	1,732.5
Nagpal et al. [23]	1st	104.9	680	11	1,153.9
this work	1st	14.2	144	54	766.8
Nagpal et al. [23]	2nd	203.9	2040	11	2,242.9
this work	2nd	23.4	432	54	1,263.6

5 Side-Channel Analysis

To verify the effectiveness of a countermeasure, it is common to simulate the power consumption of a device under attack. Moreover, implementing self-timed circuits on an FPGA is not straightforward, since the Muller c-elements and the handshake logic contains combinatorial loops. For these reasons, we use simulated traces in order to evaluate the side-channel vulnerability of our design.

To obtain a realistic acquisition, our simulations take into account the standard cell timing behavior so that the occurrence of glitches is possible. The logic simulation outputs value change dump (VCD) files which can be parsed to model the power consumption by processing the toggling activity of all wires in the device under test (DUT). Thus, we model the system's power consumption in a noiseless manner with a sampling frequency of 1 GHz. We refer to Figure 12, which shows the block diagram illustrating the process to obtain the power consumption traces used in this work.

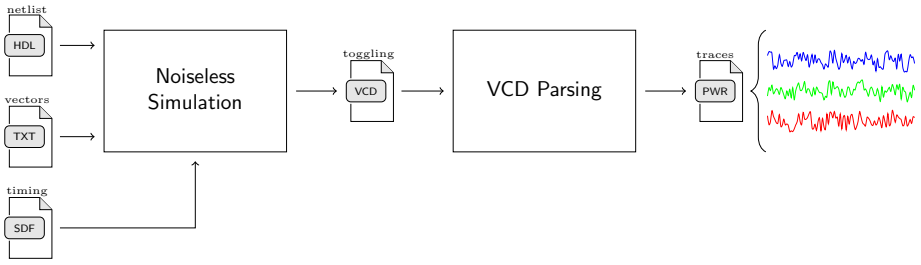


Fig. 12. Modeling power traces from VCD files generated after timing simulations.

We apply the *fixed vs random t-test* methodology[3] proposed by Goodwill et al. [10]. It uses the Welch's t-test to determine whether the difference of two dataset means provides sufficient evidence to reject the null hypothesis.

[3] We use SCALib for side-channel analysis: github.com/simple-crypto/scalib

5.1 AES S-box analysis

We start with a side-channel evaluation of our AES S-box. Figures 13 and 14 show the first and second-order univariate t-test results using one million simulated traces for the first-order implementation of our self-timed AES S-box. As expected, no exploitable side-channel leakages were identified in the first-order analysis. But second-order leakages were spotted for our first-order design.

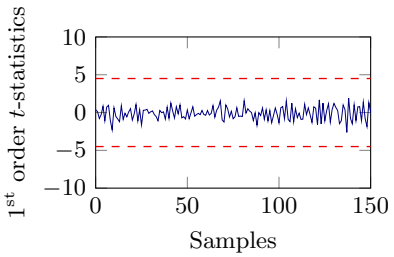


Fig. 13. 1st order TVLA results for the 1st order masked AES S-box based on one million simulated traces.

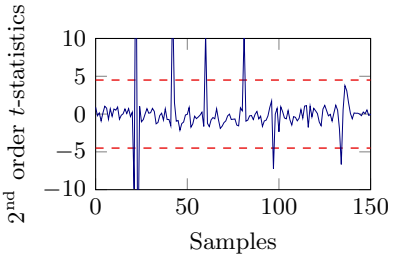


Fig. 14. 2nd order TVLA results for the 1st order masked AES S-box based on one million simulated traces.

Figures 15 and 16 show the second and third-order univariate t-test results using one million simulated traces for the second order implementation of our self-timed AES S-box.

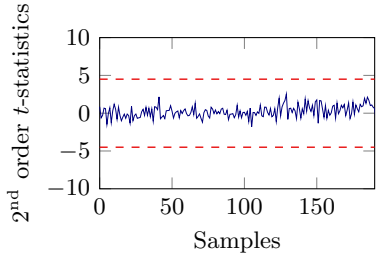


Fig. 15. 2nd order TVLA results for the 2nd order masked AES S-box based on one million simulated traces.

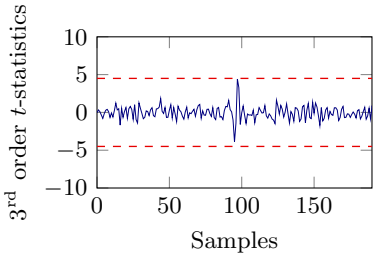


Fig. 16. 3rd order TVLA results for the 2nd order masked AES S-box based on one million simulated traces.

Again, the results confirm the robustness of our designs against side-channel analysis, even for the second-order implementation. These results are in adequacy with the DOM [13]. Thus, replacing registers by Muller c-element latches does not introduce weaknesses in the higher-order design.

5.2 Full design analysis

We evaluate the whole AES encryption in the same manner. Figures 17 and 18 show the first and second-order univariate t-test results using one hundred thousand simulated traces for the first-order implementation of our AES. Similar to the S-box, no first-order leakages were detected for the first-order design.

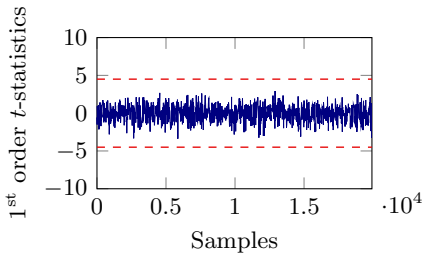


Fig. 17. 1^{st} order TVLA results for the 1^{st} order masked AES encryption based on one hundred thousand simulated traces.

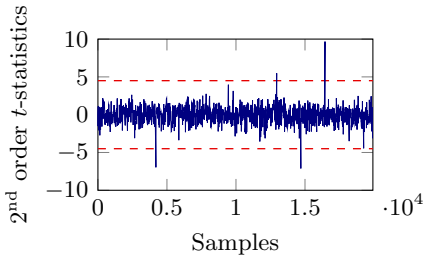


Fig. 18. 2^{nd} order TVLA results for the 1^{st} order masked AES encryption based on one hundred thousand simulated traces.

Figures 19 and 20 show the second and third-order univariate t-test results using one hundred thousand simulated traces for the second-order implementation of our self-timed AES. Once again, the side-channel evaluation was performed on whole the encryption. As expected, the second-order evaluation shows no leakage, but third-order univariate analysis shows that some points cross the threshold line.

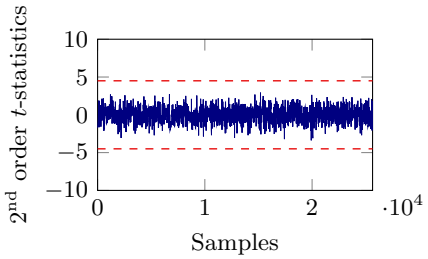


Fig. 19. 2^{nd} order TVLA results for the 2^{nd} order masked AES encryption based on one million simulated traces.

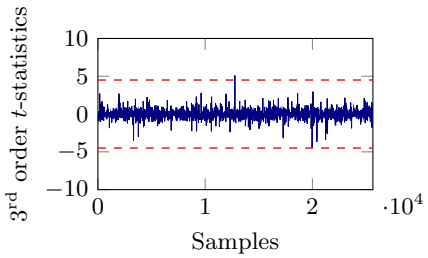


Fig. 20. 3^{rd} order TVLA results for the 2^{nd} order masked AES encryption based on one million simulated traces.

These results show that if time leakages are present in our design, they are difficult to exploit and to detect even in high resolution and low noise environment.

5.3 Bivariate analysis

Figure 21 shows the bivariate analysis for the second-order AES S-box for one million traces, whereas Figure 22 shows the t-test results for the second-order AES encryption using one hundred thousand traces. We stress that our side-channel evaluations were made in a noiseless high-resolution setting. This is due to measurement methodology used in this work, resulting in power consumption traces modeled from post-synthesis simulation taking into account the gate-level delays.

In both analysis, the upper triangle shows the results when random mask refresh is disabled, while the lower triangle illustrates the side-channel analysis when fresh randomness is employed. The result obtained from the unprotected setting uses only 10% of the amount of traces used in the protected scenario: one hundred thousand traces for the S-box and ten thousand traces for the AES encryption.

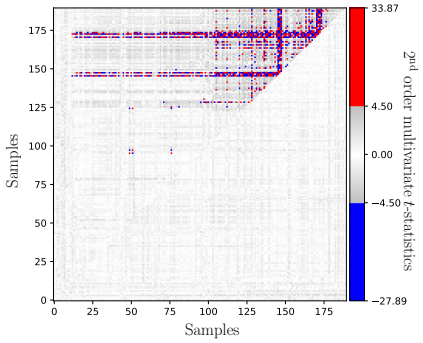


Fig. 21. Bivariate analysis of the 2nd order self-timed AES S-box implementation. The lower triangle shows the 2nd order t-test using one million traces with mask refresh enabled. The upper triangle shows the 2nd order t-test using one hundred thousand traces with mask refresh disabled.

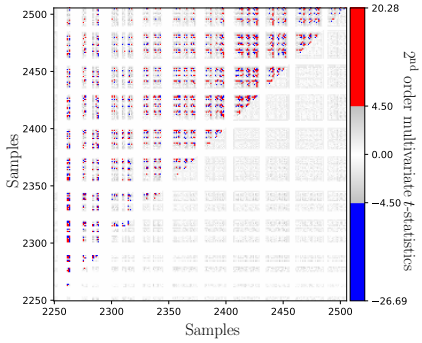


Fig. 22. Bivariate analysis of the 2nd order self-timed AES encryption implementation. The lower triangle shows the 2nd order t-test using one hundred thousand traces with mask refresh enabled. The upper triangle shows the 2nd order t-test using ten thousand traces with mask refresh disabled.

The blue and red dots shown in the Figures 21 and 22 represent sample points in which the multivariate t-statistics exceed the ± 4.5 threshold. For ease of visualization, only the first encryption round is shown in the AES analysis. As expected when the random number generator is shut off, second-order leakages are detected in our design, confirming the need of random refresh in DOM. It also confirms that our register replacement approach do not bring any weakness at higher-order in bivariate setting.

6 Conclusion

As previously stated, synchronizing the intermediate shares at the boundaries of combinatorial blocks is of high importance to obtain a secure masking implementation. Thus, this work presented a generic solution, that may be applied to different masked S-boxes, permitting the designer to obtain single-cycle implementations while assuring secure masking properties. Indeed, the main asset of our work is the reduction of the S-box latency to a single clock cycle, a feature achieved when replacing the register layers with self-timed latches. Nevertheless, the dual-rail logic adds a significant gate-count overhead to the final implementation, limiting its application in low area scenarios.

We observe that our solution has a low throughput compared to other low-latency solutions. This is due to the number of S-box stages in the pipeline, slowing the handshake logic propagation. For example, the throughput could be improved by reducing the number of S-box stages, enhancing the acknowledgement logic depth by reducing the number of intermediate handshakes.

Although being resistant against glitches, our DOM-based design uses the pre-charge / evaluate logic with monotonic cells, eliminating this hazard. Further research can be done to relax the security properties of the masked gadget when glitches are eliminated in order to reduce the overall implementation costs.

In order to evaluate our designs, we describe the implementation of a self-timed AES S-box and provide leakage assessment results based on noiseless side-channel analysis. One of the motivations behind a noiseless leakage assessment is to observe potential timing leakages due to the self-timed behavior of our S-box.

Furthermore, we present a 32-bit data path AES128 architecture to obtain a smaller silicon area design at the cost of computing one encryption round of the AES in five clock cycles. We highlight that our first-order AES encryption requires 54 clock cycles with a total area of 14 kGE approximately.

Finally, despite the area and throughput overheads, replacing the register by self-timed latches does not bring any weakness to the DOM implementation, even in second-order multivariate leakage analysis.

References

1. Arribas, V., Zhang, Z., Nikova, S.: LLTI: low-latency threshold implementations. IEEE Trans. Inf. Forensics Secur. **16**, 5108–5123 (2021). https://doi.org/10.1109/TIFS.2021.3123527, https://doi.org/10.1109/TIFS.2021.3123527
2. Barthe, G., Belaïd, S., Dupressoir, F., Fouque, P., Grégoire, B., Strub, P., Zucchini, R.: Strong non-interference and type-directed higher-order masking. In: Weippl, E.R., Katzenbeisser, S., Kruegel, C., Myers, A.C., Halevi, S. (eds.) Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016. pp. 116–129. ACM (2016). https://doi.org/10.1145/2976749.2978427, https://doi.org/10.1145/2976749.2978427

3. Batina, L., Robshaw, M. (eds.): Cryptographic Hardware and Embedded Systems - CHES 2014 - 16th International Workshop, Busan, South Korea, September 23-26, 2014. Proceedings, Lecture Notes in Computer Science, vol. 8731. Springer (2014). https://doi.org/10.1007/978-3-662-44709-3, https://doi.org/10.1007/978-3-662-44709-3
4. Bhasin, S., Guilley, S., Flament, F., Selmane, N., Danger, J.: Countering early evaluation: an approach towards robust dual-rail precharge logic. In: Proceedings of the 5th Workshop on Embedded Systems Security, WESS 2010, Scottsdale, AZ, USA, October 24, 2010. p. 6. ACM (2010). https://doi.org/10.1145/1873548.1873554, https://doi.org/10.1145/1873548.1873554
5. Boyar, J., Peralta, R.: A small depth-16 circuit for the AES S-box. In: Gritzalis, D., Furnell, S., Theoharidou, M. (eds.) Information Security and Privacy Research - 27th IFIP TC 11 Information Security and Privacy Conference, SEC 2012, Heraklion, Crete, Greece, June 4-6, 2012. Proceedings. IFIP Advances in Information and Communication Technology, vol. 376, pp. 287–298. Springer (2012). https://doi.org/10.1007/978-3-642-30436-1_24, https://doi.org/10.1007/978-3-642-30436-1_24
6. Canright, D.: A very compact S-box for AES. In: Rao and Sunar [26], pp. 441–455. https://doi.org/10.1007/11545262_32, https://doi.org/10.1007/11545262_32
7. Chari, S., Jutla, C.S., Rao, J.R., Rohatgi, P.: Towards sound approaches to counteract power-analysis attacks. In: Wiener, M.J. (ed.) Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings. Lecture Notes in Computer Science, vol. 1666, pp. 398–412. Springer (1999). https://doi.org/10.1007/3-540-48405-1_26, https://doi.org/10.1007/3-540-48405-1_26
8. Davis, A., Nowick, S.M.: Asynchronous circuit design: Motivation, background, & methods. In: Asynchronous Digital Circuit Design, pp. 1–49. Springer (1995)
9. Faust, S., Grosso, V., Pozo, S.M.D., Paglialonga, C., Standaert, F.: Composable masking schemes in the presence of physical defaults & the robust probing model. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2018**(3), 89–120 (2018). https://doi.org/10.13154/tches.v2018.i3.89-120, https://doi.org/10.13154/tches.v2018.i3.89-120
10. Goodwill, G., Jun, B., Jaffe, J., Rohatgi, P.: A testing methodology for side-channel resistance validation. In: NIST non-invasive attack testing (NIAT) workshop. vol. 7, pp. 115–136 (2011)
11. Goubin, L., Patarin, J.: DES and differential power analysis (the "duplication" method). In: Koç, Ç.K., Paar, C. (eds.) Cryptographic Hardware and Embedded Systems, First International Workshop, CHES'99, Worcester, MA, USA, August 12-13, 1999, Proceedings. Lecture Notes in Computer Science, vol. 1717, pp. 158–172. Springer (1999). https://doi.org/10.1007/3-540-48059-5_15, https://doi.org/10.1007/3-540-48059-5_15
12. Groß, H., Iusupov, R., Bloem, R.: Generic low-latency masking in hardware. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2018**(2), 1–21 (2018). https://doi.org/10.13154/tches.v2018.i2.1-21, https://doi.org/10.13154/tches.v2018.i2.1-21
13. Groß, H., Mangard, S., Korak, T.: Domain-oriented masking: Compact masked hardware implementations with arbitrary protection order. In: Bilgin, B., Nikova, S., Rijmen, V. (eds.) Proceedings of the ACM Workshop on Theory of Implementation Security, TIS@CCS 2016 Vienna, Aus-

tria, October, 2016. p. 3. ACM (2016). https://doi.org/10.1145/2996366.2996426, https://doi.org/10.1145/2996366.2996426

14. Ishai, Y., Sahai, A., Wagner, D.A.: Private circuits: Securing hardware against probing attacks. In: Boneh, D. (ed.) Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings. Lecture Notes in Computer Science, vol. 2729, pp. 463–481. Springer (2003). https://doi.org/10.1007/978-3-540-45146-4_27, https://doi.org/10.1007/978-3-540-45146-4_27
15. Jukna, S.: Notes on hazard-free circuits. SIAM J. Discret. Math. **35**(2), 770–787 (2021). https://doi.org/10.1137/20M1355240, https://doi.org/10.1137/20M1355240
16. Kulikowski, K.J., Karpovsky, M.G., Taubin, A.: Power attacks on secure hardware based on early propagation of data. In: 12th IEEE International On-Line Testing Symposium (IOLTS 2006), 10-12 July 2006, Como, Italy. pp. 131–138. IEEE Computer Society (2006). https://doi.org/10.1109/IOLTS.2006.49, https://doi.org/10.1109/IOLTS.2006.49
17. Leiserson, A.J., Marson, M.E., Wachs, M.A.: Gate-level masking under a path-based leakage metric. In: Batina and Robshaw [3], pp. 580–597. https://doi.org/10.1007/978-3-662-44709-3_32, https://doi.org/10.1007/978-3-662-44709-3_32
18. Mangard, S., Popp, T., Gammel, B.M.: Side-channel leakage of masked CMOS gates. In: Menezes, A. (ed.) Topics in Cryptology - CT-RSA 2005, The Cryptographers' Track at the RSA Conference 2005, San Francisco, CA, USA, February 14-18, 2005, Proceedings. Lecture Notes in Computer Science, vol. 3376, pp. 351–365. Springer (2005). https://doi.org/10.1007/978-3-540-30574-3_24, https://doi.org/10.1007/978-3-540-30574-3_24
19. Moradi, A., Immler, V.: Early propagation and imbalanced routing, how to diminish in FPGAs. In: Batina and Robshaw [3], pp. 598–615. https://doi.org/10.1007/978-3-662-44709-3_33, https://doi.org/10.1007/978-3-662-44709-3_33
20. Moradi, A., Poschmann, A., Ling, S., Paar, C., Wang, H.: Pushing the limits: A very compact and a threshold implementation of AES. In: Paterson, K.G. (ed.) Advances in Cryptology - EUROCRYPT 2011 - 30th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tallinn, Estonia, May 15-19, 2011. Proceedings. Lecture Notes in Computer Science, vol. 6632, pp. 69–88. Springer (2011). https://doi.org/10.1007/978-3-642-20465-4_6, https://doi.org/10.1007/978-3-642-20465-4_6
21. Moradi, A., Schneider, T.: Side-channel analysis protection and low-latency in action - case study of PRINCE and Midori. In: Cheon, J.H., Takagi, T. (eds.) Advances in Cryptology - ASIACRYPT 2016 - 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part I. Lecture Notes in Computer Science, vol. 10031, pp. 517–547 (2016). https://doi.org/10.1007/978-3-662-53887-6_19, https://doi.org/10.1007/978-3-662-53887-6_19
22. Muller, D.E., Bartky, W.S.: A theory of asynchronous circuits. In: Proceedings of an International Symposium on the Theory of Switching, April 1957, Part I. the annals of the computation laboratory of Harvard University, vol. XXIX, pp. 204–243. Cambridge University Press, Cambridge, MA, USA (1959). https://doi.org/https://doi.org/10.2307/3611677

23. Nagpal, R., Gigerl, B., Primas, R., Mangard, S.: Riding the waves towards generic single-cycle masking in hardware. IACR Cryptol. ePrint Arch. p. 505 (2022), https://eprint.iacr.org/2022/505
24. Nikova, S., Rechberger, C., Rijmen, V.: Threshold implementations against side-channel attacks and glitches. In: Ning, P., Qing, S., Li, N. (eds.) Information and Communications Security, 8th International Conference, ICICS 2006, Raleigh, NC, USA, December 4-7, 2006, Proceedings. Lecture Notes in Computer Science, vol. 4307, pp. 529–545. Springer (2006). https://doi.org/10.1007/11935308_38, https://doi.org/10.1007/11935308_38
25. Popp, T., Mangard, S.: Masked dual-rail pre-charge logic: DPA-resistance without routing constraints. In: Rao and Sunar [26], pp. 172–186. https://doi.org/10.1007/11545262_13, https://doi.org/10.1007/11545262_13
26. Rao, J.R., Sunar, B. (eds.): Cryptographic Hardware and Embedded Systems - CHES 2005, 7th International Workshop, Edinburgh, UK, August 29 - September 1, 2005, Proceedings, Lecture Notes in Computer Science, vol. 3659. Springer (2005). https://doi.org/10.1007/11545262, https://doi.org/10.1007/11545262
27. Reparaz, O., Bilgin, B., Nikova, S., Gierlichs, B., Verbauwhede, I.: Consolidating masking schemes. In: Gennaro, R., Robshaw, M. (eds.) Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part I. Lecture Notes in Computer Science, vol. 9215, pp. 764–783. Springer (2015). https://doi.org/10.1007/978-3-662-47989-6_37, https://doi.org/10.1007/978-3-662-47989-6_37
28. Sasdrich, P., Bilgin, B., Hutter, M., Marson, M.E.: Low-latency hardware masking with application to AES. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2020**(2), 300–326 (2020). https://doi.org/10.13154/tches.v2020.i2.300-326, https://doi.org/10.13154/tches.v2020.i2.300-326
29. Sparsø, J.: Introduction to Asynchronous Circuit Design. DTU Compute, Technical University of Denmark (2020), paperback edition available here: https://www.amazon.com/dp/B08BF2PFLN
30. Tiri, K., Verbauwhede, I.: A logic level design methodology for a secure DPA resistant ASIC or FPGA implementation. In: 2004 Design, Automation and Test in Europe Conference and Exposition (DATE 2004), 16-20 February 2004, Paris, France. pp. 246–251. IEEE Computer Society (2004). https://doi.org/10.1109/DATE.2004.1268856, https://doi.org/10.1109/DATE.2004.1268856