Self-Timed Masking

Implementing First-Order Masked S-Boxes Without Registers

Mateus Simoes[1,2], Lilian Bossuet[1], Nicolas Bruneau[2], Vincent Grosso[1] and Patrick Haddad[2]

[1] Laboratoire Hubert Curien, Saint-Etienne, France
{mateus.simoes,lilian.bossuet,vincent.grosso}@univ-st-etienne.fr
[2] STMicroelectronics, Rousset, France {nicolas.bruneau,patrick.haddad}@st.com

Abstract. Passive physical attacks represent a threat to microelectronics systems by exploiting leakages through side-channels, such as power consumption and electromagnetic radiation. In this context, masking is a sound countermeasure against side-channel attacks, which splits the secret data into several randomly uniform data, achieving independence between the data processing and the secret variable. However, a secure masking scheme requires additional implementation costs. Furthermore, glitches and early evaluation can temporally weaken a masked implementation in hardware, creating a potential source of exploitable leakages.
This work shows how to create register-free masking schemes that avoid the early evaluation effect with the help of the dual-rail logic. Moreover, we employ monotonic functions with the purpose of eliminating the occurrence of glitches in combinational circuits. Finally, we evaluate different 2-share masked implementations of the PRESENT and AES S-boxes in a noiseless scenario in order to detect potential first-order leakages and to determine data propagation profiles correlated to the secret variables.

Keywords: masking · hardware security · self-timed circuits

1 Introduction

Physical attacks represent a threat to electronic systems. They allow an adversary to discover sensitive information — e.g. a private key — from a targeted device. Some physical attacks exploit side-channel leakages, such as power consumption and electromagnetic emanation, in order to create a statistical model of the system's behavior, granting to the adversary a robust way to rank secret variable hypothesis [KJJ99].

Several techniques exist to counter side-channel attacks; one of the most studied is the Boolean masking, introduced by Goubin and Patarin [GP99]. This countermeasure makes the data computation independent of the secret data itself by splitting the n-bit sensitive variable into uniformly distributed shares over $\mathbb{F}_2^n$. In other words, a secret x is d^{th} order masked with $d+1$ shares as shown in equation (1), with $(x_0, x_1, \ldots, d_{d-1})$ the random shares and x_d the masked value.

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

Thus, a masked circuit uses the shares to compute a given secret instead of the secret itself, rendering the correlation between the side-channel leakages and the sensitive variable more complex. At the appropriate moment, the shares are linearly recombined to uncover the secret, that is, $x = \bigoplus_{i=0}^{d} x_i$. Note that security comes at the cost of higher implementation complexity, raising the transistor count in the circuit.

Furthermore, a function $f : \{0,1\}^n \mapsto \{0,1\}$ can be masked by grouping its shared terms among k component functions f_i in a way that a masked function is thus represented as $f(x) = \bigoplus_{i=0}^{k} f_i(\mathcal{S}^i)$, with $\mathcal{S}^i$ a proper subset of $\mathcal{S} = (x_0, x_1, \ldots, x_d)$ the sharing of x. 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 over $\mathbb{F}_2^n$ with its d^{th} order masking expressed as shown in the equation (2).

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

However, masking non-linear functions, 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. Consequently, equation (2) does not stand for non-linear masking. In addition, when designing a masked circuit, sharing recombination may be the source of exploitable side-channel leakages, rendering non-linear masking a critical task for security engineers. In this context, techniques such as threshold implementations (TI) [NRR06] and domain-oriented masking (DOM) [GMK16] were proposed, upon which we can rely to mask non-linear functions up to a certain order [MMSS19].

In both examples, DOM and TI, glitches were an important factor to take account when building the masking schemes. In fact, this physical hazard represents a threat to masked circuits, since a glitchy function consumes an unnecessary power consumption that may be correlated to the unshared variable [MPG05, MPO05]. In order to guarantee the security of non-linear functions in the presence of glitches, register barriers are employed to cease the spurious propagation [RBN+15]. Reciprocally, the component functions can group the shares in a non-complete way, that is, avoiding masked functions that manipulate all shares at once, so that a glitchy behavior does not correlate to the unshared sensitive variable [NRR06].

A popular method to determine the local security of a masked gadget is the probing model of Ishai et al. [ISW03], a leakage analysis on which the adversary can place up to d probes on different wires of the circuit in order to obtain their current logic states. Then, one can compute the logic state distribution of the probed wires based on different sharing inputs, providing us with clues about a potential dependence between the unshared value and the internal signals. This model was enhanced in order to take into account physical defaults such as glitches [FGP+18], highlighting the importance of register layers in order to synchronize the data flow.

Other methods determine the composability security of a masked gadget, introducing the notions of Non-Interference (NI) and Strong Non-Interference (SNI) [BBD+16]. In order to satisfy these theoretical models, enabling the secure inter-connection of different masked blocks, many solutions employ random bits to refresh the masks during the computation. In consequence, masking a circuit is expensive, since the latency is increased as well as the gate counting. Moreover, the need to refresh the masks is an additional overhead in masked solutions, requiring random number generators.

Therefore, reducing the masking costs is a pertinent branch of research within side-channel analysis and the main topic of relevant techniques published recently. For example, Sugawara proposed a 3-share AES S-box implementation without random refresh bits [Sug19] using the changing of the guards [Dae17]. In parallel, Gross et al. proposed a low-latency masking at the cost of increasing both the gate counting and the randomness [GIB18]. Indeed, many recent solutions try to balance those costs in order to propose a secure masking that may fit different constraints.

In the same manner, this work explores an atypical circuit design approach with the purpose of balancing the masking costs. We aim at reducing the number clock cycles needed to compute the whole masked function. For that, we present a self-timed masking scheme built upon the Muller c-element [MB59]. Additionally, we employ the dual-rail protocol to create glitch-free combinational circuits with the help of monotonic gates in a pre-charge / evaluate logic.

Our contribution Aiming at studying the behavior of a single cycle masked solution, we show how to replace registers by self-timed latches built upon Muller c-elements, assuring data synchronization among different combinational layers. Also, we detail the operation of monotonic functions relying on the dual-rail protocol, allowing us to create glitch-free masked circuits. Furthermore, we present our locally asynchronous globally synchronous designs, showing the implementation results of 2-share self-timed masked PRESENT and AES S-boxes. Finally, we evaluate the side-channel leakages based on experimental measurements and, based on the results, we consider the main drawbacks of our self-timed solutions and how to mitigate them.

Paper organization Section 2 introduces the notation used along this paper. Next, in the section 3, we briefly describe the dual-rail encoding and some features that it unlocks in digital circuit design. These features are employed on a self-timed masked implementation of the PRESENT and AES S-boxes, sections 4.1 and 4.2 respectively. For both designs, the implementation results are detailed in the section 5. Finally, in section 6, we evaluate the robustness of our countermeasure against side-channel attacks using the test vector leakage assessment methodology.

2 Notations

We denote binary random variables in $\mathbb{F}_2$ with lower-case letters, e.g. x, binary words in $\mathbb{F}_2^{n>1}$ with upper-case letters, whose j^{th} element of X with superscript x^j and the i^{th} share of a variable with subscript x_i. The lower-case letter r refers, exclusively, to a uniformly distributed bit used for mask refreshing.

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 $x = \bigoplus_{i=0}^{d} x_i$. Similarly, a Boolean function $f : \{0,1\}^n \mapsto \{0,1\}$ can be d^{th} order masked, resulting in a set of k masked functions $f_i(\mathcal{S}^i)$ denoted with lower-case letters, with k the cardinality of this set of functions and $\mathcal{S}^i$ a proper subset of $\mathcal{S}$, so that $f(x) = \bigoplus_{i=0}^{k} f_i(\mathcal{S}^i)$.

We use typewriter fonts to denote binary random variables `x`, vectors `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 $\texttt{x.t} = x$ while `x.f` signalizes the complement $\texttt{x.f} = \overline{x}$. A dual-rail token is then referred as $\overset{*}{\texttt{x}} = \texttt{(x.t,x.f)}$.

3 Background

Aiming at the study of masking schemes without registers, we show in this work the design of self-timed S-boxes. In fact, we are aware of the importance of synchronization layers to isolate different combinational blocks in masked solutions, avoiding glitches and satisfying non-completeness [RBN+15]. Therefore, our goal here is to provide the analysis of masked circuits in which the register layers are replaced by self-timed synchronization gadgets while preserving relevant security properties. To achieve such self-timed feature, we make use of the dual-rail encoding.

Indeed, several works have already addressed the use of the dual-rail encoding in cryptographic applications [TV04, PM05, CZ06, LMW14, SBHM20]. Similarly, we also employ the dual-rail protocol in a pre-charge / evaluate logic style with the purpose of creating monotonic functions that are free of glitches. Moreover, our monotonic circuits is resistant against the early propagation phenomenon.

3.1 Secure masking implementations

This work relies on known secure masked implementations. In fact, it has already been shown that an effective d^{th} order masking in hardware can be achieved using $d+1$ input shares [RBN+15, GMK16, FGP+18]. However, in order to satisfy d-glitch-extended probing security, a masked gadget can be divided into two register-isolated steps, which we identify, in this work, as processing and compression.

For example, let us take the first order DOM-*indep* gadget that computes the function $z = f(a,b) = a \wedge b$ with $\mathcal{A} = (a_0, a_1)$ and $\mathcal{B} = (b_0, b_1)$ the input shares and $\mathcal{Z} = (z_0, z_1)$ the output sharing [GMK16]. In short, assuming that the input sharings are uniform, we want to find a secure way to compute $(z_0 \oplus z_1) = (a_0 \oplus a_1) \wedge (b_0 \oplus b_1)$. Hence, the process step computes the product terms a_0b_0, a_0b_1, a_1b_0, a_1b_1 and adds a fresh share r to the cross-domain ones, that is, a_0b_1 and a_1b_0. Then, to assure non-completeness, registers ($\longrightarrow$) store the resulting shares (x'_0, x'_1, x'_2, x'_3), as we can see in the equation (3).

$$\begin{aligned} f_0(a_0, b_0) &= a_0b_0 && \longrightarrow x'_0 \\ f_1(a_0, b_1) &= a_0b_1 \oplus r && \longrightarrow x'_1 \\ f_2(a_1, b_0) &= a_1b_0 \oplus r && \longrightarrow x'_2 \\ f_3(a_1, b_1) &= a_1b_1 && \longrightarrow x'_3 \end{aligned} \tag{3}$$

The process step produces four shares, albeit the DOM AND gate has only two shares at the output. In order to reduce the number of output shares, there exists a compression step, as shown in the equation (4). Thanks to the register barrier between both steps, this DOM gadget satisfies first-order glitch-extended security evaluation [FGP+18].

$$\begin{aligned} z_0 &= x'_0 \oplus x'_1 \\ z_1 &= x'_2 \oplus x'_3 \end{aligned} \tag{4}$$

The process step uses a random bit r to mask cross-domain multiplications in $\mathbb{F}_2$. Nonetheless, Shahmirzadi and Moradi have shown how to find a set of glitch-extended component functions without fresh randomness [SM21]. Even without fresh randomness, their solution has a uniform distribution of all intermediate signals, preventing sensitive data dependence on ISW-based probing models. For instance, the process step for the function $f(a,b) = \overline{a} \wedge b$ can be built without mask refreshing, as shown in the equation (5).

$$\begin{aligned} f_0(a_0, b_0) &= a_0b_0 \oplus b_0 && \longrightarrow x'_0 \\ f_1(a_0, b_1) &= a_0b_1 && \longrightarrow x'_1 \\ f_2(a_1, b_0) &= a_1b_0 && \longrightarrow x'_2 \\ f_3(a_1, b_1) &= a_1b_1 \oplus b_1 && \longrightarrow x'_3 \end{aligned} \tag{5}$$

Our work relies on both implementations [GMK16, SM21] to construct our self-timed designs. We do not aim at comparing both works, but at studying the behavior of self-timed masked circuits with and without random mask refreshing.

3.2 The dual-rail encoding

The dual-rail protocol encodes a bit using two signal wires [DN95]. To illustrate, let us take the dual-rail encoding of a variable x; a wire `x.t` carries its current logic value while a second wire `x.f` transports its complement. The piece of information x is then dual-rail encoded as a token $\overset{*}{\mathtt{x}} = (\mathtt{x.t}, \mathtt{x.f}) = (x, \overline{x})$ containing both signals. In this configuration, a valid bit is obtained when one, and only one, signal wire is active (i.e. in a high logic state), although a null information $\varnothing$ is encoded when both wires are deactivated, that is, $\varnothing = (0,0)$. Finally, the encoding $(1,1)$ is never used because it does not represent neither a valid neither a null token in our constructions. Moreover, the behavior of our designs after the injection of this invalid token is out of scope of this work.

The data computation in our asynchronous designs is done in two times. First, the circuit is pre-charged, setting all intermediate wires to the null state. Then, a valid token is processed in the evaluate phase, toggling the intermediate signals in order to obtain a valid 0 or a valid 1. Table 1 summarizes the dual-rail logic encoding.

Table 1: The dual-rail tokens.

token	`x.t`	`x.f`
null	0	0
$x = 0$	0	1
$x = 1$	1	0
illegal	1	1

With this architecture, the dual-rail (DR) NOT gate comes for "free" in terms of transistors, since it is computed by swapping the dual-rail wires. However, building other logic gates is slightly more complex. See equations (6), (7) and (8) for the dual-rail implementation of the NOT, AND and OR functions, respectively.

$$\begin{aligned} z = \neg a \iff \texttt{z.t} &= \texttt{a.f} \\ \texttt{z.f} &= \texttt{a.t} \end{aligned} \tag{6}$$

$$\begin{aligned} z = a \wedge b \iff \texttt{z.t} &= \texttt{a.t} \wedge \texttt{b.t} \\ \texttt{z.f} &= \texttt{a.f} \vee \texttt{b.f} \end{aligned} \tag{7}$$

$$\begin{aligned} z = a \vee b \iff \texttt{z.t} &= \texttt{a.t} \vee \texttt{b.t} \\ \texttt{z.f} &= \texttt{a.f} \wedge \texttt{b.f} \end{aligned} \tag{8}$$

In the same manner, the DR-XOR can be obtained as shown in the equation (9). Despite requiring more logic gates than the other basic Boolean functions, the dual-rail XOR has, approximately, twice the surface of its single-rail equivalent.

$$\begin{aligned} z = a \oplus b \iff \texttt{z.t} &= (\texttt{a.t} \wedge \texttt{b.f}) \vee (\texttt{a.f} \wedge \texttt{b.t}) \\ \texttt{z.f} &= (\texttt{a.f} \wedge \texttt{b.f}) \vee (\texttt{a.t} \wedge \texttt{b.t}) \end{aligned} \tag{9}$$

Note that only regular AND and OR gates are used to construct these Boolean functions in dual-rail logic. Indeed, the dual-rail gates are expressed in their disjunctive normal form (DNF). The motivation behind this choice of design is to obtain a monotonic behavior, allowing us to construct a glitch-free hardware architecture [Juk21].

At first glance, the dual-rail functions may look disadvantageous when compared to their single-rail form. Truly, the gate count is increased by a factor of two, approximately, which brings an unfavorable outcome for limited area and speed applications. Nevertheless, the dual-rail encoding is an interesting design alternative when avoiding glitches is necessary or there exists latency constraints to implement secure masking schemes, as we will show in this work.

3.3 Eliminating glitches with monotonic logic

Glitches are critical in hardware security implementations [MPG05, MPO05]. For instance, let us take the DOM gadget introduced in the subsection 3.1. For the sake of simplicity, we ignore the random refresh bits. If we remove the registers and the refresh bit we obtain the combinational function shown in equation (10).

$$\begin{aligned} z_0 &= f_0(a_0, b_0, b_1) = a_0 b_0 \oplus a_0 b_1 \\ z_1 &= f_1(a_1, b_0, b_1) = a_1 b_0 \oplus a_1 b_1 \end{aligned} \tag{10}$$

Due to physical aspects, such as wiring or gate delay, the correct sharing of b can arrive earlier than the correct sharing a at the AND gates. This scenario may produce spurious intermediate signal transitions, leading to an unexpected behavior during a small period of time Δ_t. In this context, let us observe the output share $z_0 = f_0(a_0, b_0, b_1)$ when the input changes from $f_0(1, 0, 0)$ to $f_0(0, 1, 0)$, for example. In this case, f_0 behaves as a glitchy function $f_{\lightning} = b_0 \oplus b_1 = b$ whose output produces a hazard $z_0 = b^{\lightning} \rightarrow 0$, revealing the secret variable b during a short moment. In consequence, this glitch produces an unnecessary power consumption correlated to the variable b that may be exploited by an adversary to discover this sensitive information. Indeed, one can easily infer from equation (10) that the output shares reveal the secret b. Figure 1 illustrates this scenario in a gate-level abstraction.

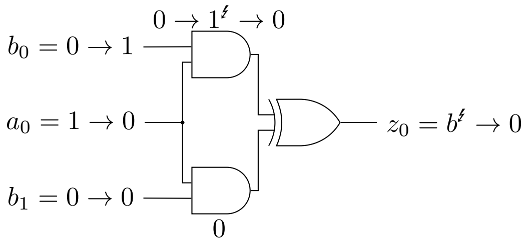
$0 \rightarrow 1^{\lightning} \rightarrow 0$
$b_0 = 0 \rightarrow 1$
$a_0 = 1 \rightarrow 0$
$b_1 = 0 \rightarrow 0$
0
$z_0 = b^{\lightning} \rightarrow 0$

Figure 1: A glitchy circuit.

One can use the dual-rail encoding with the aim of avoiding glitches in the combinational circuits, but the protocol alone does not eliminate the hazard nevertheless. Table 2 shows several glitchy scenarios for the dual-rail AND, OR and XOR gates, the digits denote their equivalent dual-rail token and the subscripts I and F indicate the initial and final instants of both a and b input signal transitions. Here, we assume that both dual-rail wires toggles at the same instant, that is, the $(\texttt{x.t}, \texttt{x.f}) = (0, 0)$ and $(\texttt{x.t}, \texttt{x.f}) = (1, 1)$ cases do not happen. However, in this abstract case, the signal $\overset{*}{\texttt{b}}$ arrives earlier than $\overset{*}{\texttt{a}}$.

Table 2: Examples of occurrence of glitches for different Boolean functions $z = f(a, b)$ in their dual-rail form.

Transition			Function		
$\overset{*}{\texttt{a}}_I \rightarrow \overset{*}{\texttt{a}}_F$	$\overset{*}{\texttt{b}}_I \rightarrow \overset{*}{\texttt{b}}_F$		$\overset{*}{\texttt{z}} = \overset{*}{\texttt{a}} \wedge \overset{*}{\texttt{b}}$	$\overset{*}{\texttt{z}} = \overset{*}{\texttt{a}} \vee \overset{*}{\texttt{b}}$	$\overset{*}{\texttt{z}} = \overset{*}{\texttt{a}} \oplus \overset{*}{\texttt{b}}$
$0 \rightarrow 1$	$0 \rightarrow 1$		$0 \rightarrow 1$	$0 \rightarrow 1$	$0 \rightarrow 1^{\lightning} \rightarrow 0$
$0 \rightarrow 1$	$1 \rightarrow 0$	`z.t`	0	$1 \rightarrow 0^{\lightning} \rightarrow 1$	$1 \rightarrow 0^{\lightning} \rightarrow 1$
$1 \rightarrow 0$	$0 \rightarrow 1$		$0 \rightarrow 1^{\lightning} \rightarrow 0$	1	$1 \rightarrow 0^{\lightning} \rightarrow 1$
$1 \rightarrow 0$	$1 \rightarrow 0$		$1 \rightarrow 0$	$1 \rightarrow 0$	$0 \rightarrow 1^{\lightning} \rightarrow 0$
$0 \rightarrow 1$	$0 \rightarrow 1$		$1 \rightarrow 0$	$1 \rightarrow 0$	$1 \rightarrow 0^{\lightning} \rightarrow 1$
$0 \rightarrow 1$	$1 \rightarrow 0$	`z.f`	1	$0 \rightarrow 1^{\lightning} \rightarrow 0$	$0 \rightarrow 1^{\lightning} \rightarrow 0$
$1 \rightarrow 0$	$0 \rightarrow 1$		$1 \rightarrow 0^{\lightning} \rightarrow 1$	0	$0 \rightarrow 1^{\lightning} \rightarrow 0$
$1 \rightarrow 0$	$1 \rightarrow 0$		$0 \rightarrow 1$	$0 \rightarrow 1$	$1 \rightarrow 0^{\lightning} \rightarrow 1$

Nonetheless, one can completely eliminate the occurrence of glitches in a digital circuit by employing monotonic gates in a pre-charge logic [PM05]. In other words, a null information is propagated before a valid token in order to assure a uniform and known transition polarity in the circuit. Thus, a gate switches at most once within an evaluation phase after being pre-charged. This functioning, however, requires a full monotonic circuit.

A Boolean function $f : \{0,1\}^n \mapsto \{0,1\}$ is called monotonic, or more specifically, monotonically increasing if:

$$\forall\, X, Y \in \{0,1\}^n \quad x^1 \leq y^1,\ x^2 \leq y^2,\ \ldots,\ x^n \leq y^n \Rightarrow f(X) \leq f(Y)$$

If we consider $X = (0, 0, \ldots, 0)$ (i.e., the minimum element of $\mathbb{F}_2^n$) the pre-charge input, evaluating a monotonic function, whose initial state is determined by $f(X)$, triggers a toggling activity from 0 to 1, never the inverse. In other words, let us say that the data processing starts with a pre-charge, in which both dual-rail wires are set to a low logic level. Then, in a second phase, a valid token is propagated in the evaluate phase, thus a $(\mathtt{x.t}, \mathtt{x.f}) = (1, 0)$ or $(\mathtt{x.t}, \mathtt{x.f}) = (0, 1)$ dual-rail composition is set. Since the pre-charge phase sets a known logic level, the circuit's switching activity occurs in only one direction when the valid token arrives, because only one wire per bit has to be set, excluding the occurrence of glitches.

Traditional AND and OR gates are examples of monotonically increasing functions. For this reason, we rely on both gates only to build a full monotonic circuits. Table 3 shows switching activities in both pre-charge and evaluate phases. The $\varnothing$ symbol represents the null data state, i.e. $\varnothing = (\mathtt{x.t}, \mathtt{x.f}) = (0, 0)$, and the digits denote their equivalent dual-rail tokens.

Table 3: Eliminating glitches with monotonic functions in a pre-charge / evaluate logic.

Stage	Transition		Function		
	$\overset{*}{\mathtt{a}}_{\mathtt{eval}} \to \overset{*}{\mathtt{a}}_{\mathtt{prch}}$	$\overset{*}{\mathtt{b}}_{\mathtt{eval}} \to \overset{*}{\mathtt{b}}_{\mathtt{prch}}$	$\overset{*}{\mathtt{a}} \wedge \overset{*}{\mathtt{b}}$	$\overset{*}{\mathtt{a}} \vee \overset{*}{\mathtt{b}}$	$\overset{*}{\mathtt{a}} \oplus \overset{*}{\mathtt{b}}$
Pre-charge	$0 \to \varnothing$	$0 \to \varnothing$	$0 \to \varnothing$	$0 \to \varnothing$	$0 \to \varnothing$
	$0 \to \varnothing$	$1 \to \varnothing$	$0 \to \varnothing$	$1 \to \varnothing$	$1 \to \varnothing$
	$1 \to \varnothing$	$0 \to \varnothing$	$0 \to \varnothing$	$1 \to \varnothing$	$1 \to \varnothing$
	$1 \to \varnothing$	$1 \to \varnothing$	$1 \to \varnothing$	$1 \to \varnothing$	$0 \to \varnothing$
	$\overset{*}{\mathtt{a}}_{\mathtt{prch}} \to \overset{*}{\mathtt{a}}_{\mathtt{eval}}$	$\overset{*}{\mathtt{b}}_{\mathtt{prch}} \to \overset{*}{\mathtt{b}}_{\mathtt{eval}}$	$\overset{*}{\mathtt{a}} \wedge \overset{*}{\mathtt{b}}$	$\overset{*}{\mathtt{a}} \vee \overset{*}{\mathtt{b}}$	$\overset{*}{\mathtt{a}} \oplus \overset{*}{\mathtt{b}}$
Evaluate	$\varnothing \to 0$	$\varnothing \to 0$	$\varnothing \to 0$	$\varnothing \to 0$	$\varnothing \to 0$
	$\varnothing \to 0$	$\varnothing \to 1$	$\varnothing \to 0$	$\varnothing \to 1$	$\varnothing \to 1$
	$\varnothing \to 1$	$\varnothing \to 0$	$\varnothing \to 0$	$\varnothing \to 1$	$\varnothing \to 1$
	$\varnothing \to 1$	$\varnothing \to 1$	$\varnothing \to 1$	$\varnothing \to 1$	$\varnothing \to 0$

Certainly, the pre-charge / evaluation logic is not limited to self-timed systems, as it has already been exploited by different works within masking applications [LMW14, SBHM20]. For this reason, associating the dual-rail protocol with monotonic functions represents a straight forward solution to completely avoid the occurrence of glitches in a combinational circuit. In consequence, the power consumption leakages are mitigated due to this glitch-free characteristic. Moreover, despite having more logic gates, a dual-rail circuit may fit low-power applications due to its lower logic toggling probability.

3.4 Avoiding the early propagation effect

The early propagation happens when the correct output of a logic function is determined and stabilizes without necessarily knowing all input signals. This behavior is critical to dual-rail solutions since a signal propagation tied to a given input may determine the signal toggling profile within a time sample. In fact, it has already been shown that this hazard may be the origin of exploitable side-channel leakages [KKT06, SS06].

To illustrate this phenomenon, consider the simple circuit shown in figure 2, representing a gate-level implementation of a perfectly balanced dual-rail design of the Boolean function $\overset{*}{\mathtt{z}} = \overset{*}{\mathtt{a}} \vee (\overset{*}{\mathtt{b}} \wedge \overset{*}{\mathtt{c}})$. Suppose that the inputs are synchronized and the logic gates have a constant propagation delay of 1 ns. Considering also that the circuit is pre-charged, we obtain the signal propagation profile shown in table 4. From this result, one can observe the dependence between the instant a gate transitions and the inputs. Indeed, when $\mathtt{a.t} = 1$ the function produces its correct output earlier, configuring a power consumption dependence with $\overset{*}{\mathtt{a}}$.

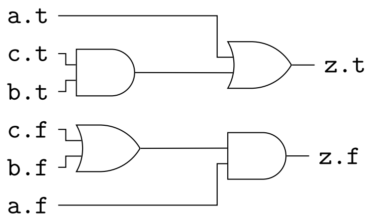

Figure 2: A simple circuit susceptible to data-dependent propagation.

$\overset{*}{\mathtt{a}}\ \overset{*}{\mathtt{b}}\ \overset{*}{\mathtt{c}}$	1 ns	2 ns
0 0 0	1	1
0 0 1	1	1
0 1 0	1	1
0 1 1	1	1
1 0 0	2	0
1 1 1	2	0
1 1 0	2	0
1 1 1	2	0

Table 4: Number of logic gate transitions for different inputs.

Futhermore, if the inputs are not synchronized, the $\overset{*}{\mathtt{a}}$ signal may arrive first, depending on physical aspects such as aging and gate delay, producing the final output without having the knowledge of the inputs $\overset{*}{\mathtt{b}}$ and $\overset{*}{\mathtt{c}}$ when $\mathtt{a.t} = 1$.

$$\overset{*}{\mathtt{a}} \vee (\overset{*}{\mathtt{b}} \wedge \overset{*}{\mathtt{c}}) = \begin{cases} \overset{*}{\mathtt{z}} = (1, 0), & \text{if } \mathtt{a.t} = 1 \\ \overset{*}{\mathtt{z}} = (\overset{*}{\mathtt{b}} \wedge \overset{*}{\mathtt{c}}), & \text{otherwise} \end{cases}$$

Different works have already addressed this hazard, proposing solutions to avoid security flaws [CZ06, PKZM07]. In this work, recognizing that logic gates do not have a homogeneous propagation delay, we observe the fact that the dual-rail XOR, equation (9), does not propagate early, since both inputs have to be valid to produce a valid output token, considering a pre-charged gadget. Based on this observation, we modify the DR-AND, equation (7), in order to obtain the same behavior. Now, we build the dual-rail gates mapping from the sum-of-minterms expressions of both wires, as well as the DR-XOR. Hence, the gadget waits for all it inputs to become valid before propagating a valid output. Equation (11) shows the resulting expressions for the DR-AND gate.

$$\begin{aligned} z = a \wedge b \Longleftrightarrow \mathtt{z.t} &= (\mathtt{a.t} \wedge \mathtt{b.t}) \\ \mathtt{z.f} &= (\mathtt{a.f} \wedge \mathtt{b.f}) \vee (\mathtt{a.f} \wedge \mathtt{b.t}) \vee (\mathtt{a.t} \wedge \mathtt{b.f}) \end{aligned} \tag{11}$$

The reasoning for the DR-OR is equivalent:

$$\begin{aligned} z = a \vee b \Longleftrightarrow \mathtt{z.t} &= (\mathtt{a.t} \wedge \mathtt{b.t}) \vee (\mathtt{a.t} \wedge \mathtt{b.f}) \vee (\mathtt{a.f} \wedge \mathtt{b.t}) \\ \mathtt{z.f} &= (\mathtt{a.f} \wedge \mathtt{b.f}) \end{aligned} \tag{12}$$

Therefore, the dual-rail gates requires the knowledge of all valid inputs in order to produce their non-null outputs. Table 5 illustrates this behavior, in a generic way, in order to mitigate the early propagation phenomenon.

Table 5: A generic truth table for the dual-rail logic gates.

$\overset{*}{\mathtt{a}}$	$\overset{*}{\mathtt{b}}$	$\overset{*}{\mathtt{z}}$
∅	∅	∅
∅	valid	∅
valid	∅	∅
valid	valid	valid

Moreover, an additional OR gate can be used to balance the `z.t` and `z.f` wires in the DR-AND and DR-OR cases, respectively, achieving an equivalent gate-depth for every input combination. The figure 3 shows the balanced function $\overset{*}{\mathtt{z}} = \overset{*}{\mathtt{a}} \vee (\overset{*}{\mathtt{b}} \wedge \overset{*}{\mathtt{c}})$ that avoids the early propagation.

We perform the previous wire toggling analysis of the improved gate-level implementation, whose results is shown in table 6 containing the propagation profile for the pre-charged circuit. Once again, suppose a constant propagation delay of 1 ns.

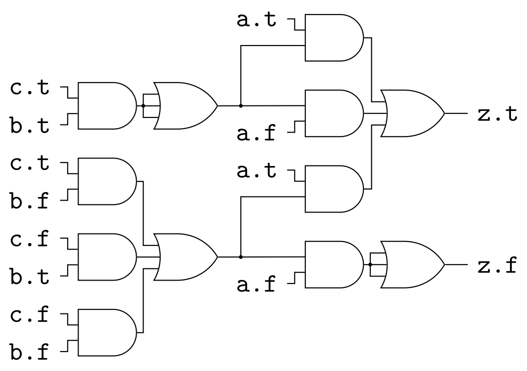

Figure 3: A balanced circuit non-susceptible to data-dependent propagation.

$\overset{*}{\mathtt{a}}\,\overset{*}{\mathtt{b}}\,\overset{*}{\mathtt{c}}$	1 ns	2 ns	3 ns	4 ns
0 0 0	1	1	1	1
0 0 1	1	1	1	1
0 1 0	1	1	1	1
0 1 1	1	1	1	1
1 0 0	1	1	1	1
1 1 1	1	1	1	1
1 1 0	1	1	1	1
1 1 1	1	1	1	1

Table 6: Number of logic gate transitions for different inputs.

As expected, the circuit is slower due to its higher logic depth, as we can observe from table 6. Therefore, avoiding the early propagation effect using this solution imposes additional gate count and lower throughput. Nevertheless, avoiding the early propagation brings a constant number of gate transitions, regardless of the input, mitigating hazardous side-channel leakages due to the power consumption.

3.5 Data synchronization with the Muller c-elements

Registers are very important components in masked circuits due to their role in synchronizing the boundaries of different combinational blocks. For instance, synchronization is a necessary aspect in order to satisfy non-completeness in threshold implementations, mitigating side-channel leakages due to the presence of glitches [RBN+15]. Indeed, registers are of primordial necessity in secure masking schemes in general [MMSS19].

However, although limiting the combinational data path, registers increase latency by requiring additional clock cycles to process the whole circuit. In this work, we study an alternative state-holding element, commonly used in asynchronous designs, in order to create single cycle S-boxes while preserving the non-completeness property.

Despite avoiding glitches with the help of monotonic functions, we maintain the synchronization layers in order to satisfy robust probing security constraints [FGP+18]. In fact, in this work we highlight the importance of data synchronization layers even in a glitch-free design. We demonstrate, based on empirical results in further sections, that synchronizing is also a relevant aspect in long combinational data paths.

The memory element used in this work is built upon the Muller c-element [MB59], whose symbol is shown in figure 4 alongside with a summary of its logical behavior, table 7.

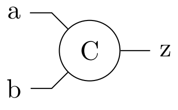

Figure 4: A Muller c-element symbol.

a	b	z
0	0	0
0	1	no change
1	0	no change
1	1	1

Table 7: The Muller c-element truth table.

From table 7, the c-element can be modelled as $z = (a \wedge b) \vee (z \wedge a) \vee (z \wedge b)$. It outputs 0 when all inputs have a low logic level, and when all inputs have a high logic level it outputs 1. In contrast, the c-element maintains its current steady state if the inputs are different. The gate-level implementation of the 2-input c-element used in this work is shown in figure 5.

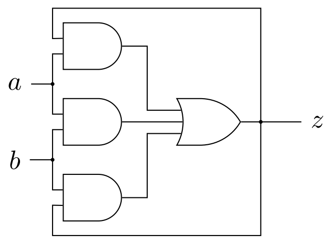

Figure 5: Gate-level implementation of the 2-input c-element.

The c-element is the base component of our self-timed designs. We use the term "self-timed" due to the handshake logic within the data storage unit that is managed by the data itself, excluding the need of a synchronous clock signal to pace the token flow. In this context, the data streams like a wave, with the intermediate states oscillating between a valid and a null token, configuring what is known as pre-charge / evaluate logic.

Based on c-elements, one can build dual-rail latches to operate as memory devices in a self-timed pipeline. Figures 6 and 7 show a 2-bit wide implementation. The dual-rail latches can be characterized as either strongly indicating or weakly indicating, depending on how their acknowledgement signal is computed. A strongly indicating latch, figure 6, waits for all of its inputs to become valid, or null, before sending the respective acknowledgement. In contrast, a weakly indicating latch, figure 7, waits for only one specific input token to become valid or null before authenticating its current state [Spa20].

In both cases, weakly or strongly indicating, n pairs of c-elements stores a n-bit token $\mathtt{\overset{*}{x}}$ and a regular 2-input NOR gate configures completion detection device for each pair — or a single pair for the weakly indicating version — managing the acknowledgement signal logic. Note that, for a n-bit wide latch, a n-input c-element is needed to store the acknowledgement output in the strongly indicating latch case, which represents an important area overhead.

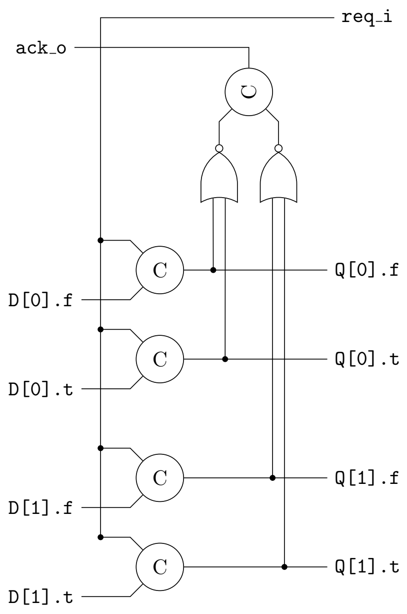

Figure 6: A 2-bit wide strongly indicating asynchronous latch.

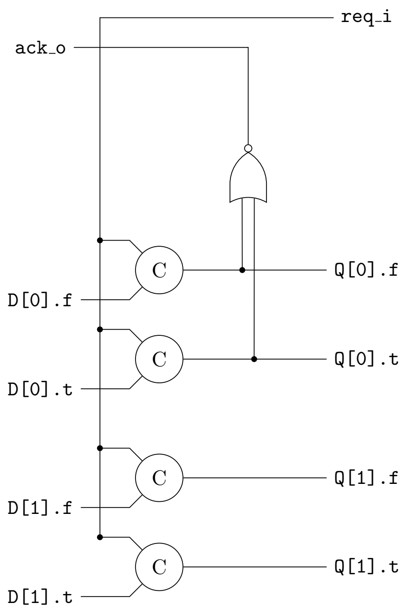

Figure 7: A 2-bit wide weakly indicating asynchronous latch.

The handshake logic contains two signals: a require input, denoted `req_i`, and an acknowledgement output, expressed 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. Similarly, the request signal — which can be the `ack_o` signal from the following latch in the pipeline or an external signal — switches the data flow. In order to operate as a switch, `req_i` is connected to one of the c-elements inputs. Thus, `req_i` = 0 requires the storage of an empty token (i.e., the pre-charge), while `req_i` = 1 means that the combinational block following the latch is ready to evaluate a new valid token. Figure 8 shows the functioning of the handshake logic.

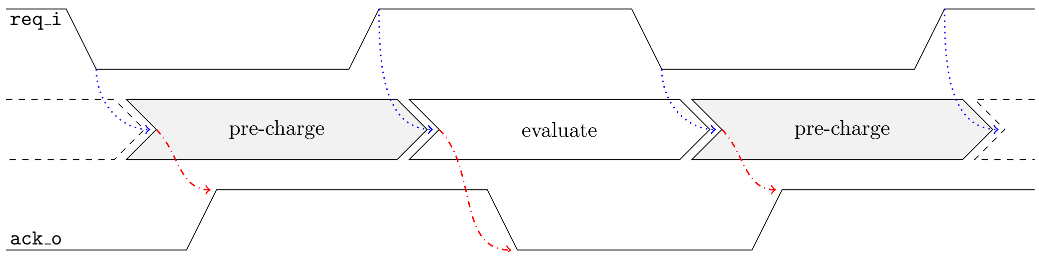

Figure 8: Self-timed handshake in a pre-charge / evaluate logic.

To illustrate the operation of a self-timed circuit, consider the following two-stage pipeline, figure 9, in which C denotes a combinational increasing monotonic circuit. For ease of visualization, ★ represents a random valid data 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), identified as the wire `ack_s`.

There is a valid token at the input, representing the information to be computed. Considering that C_A was pre-charged, this valid token was processed as soon as its arrival at the pipeline input. The latch (A) keeps its logic state since its `ack_s` = 0.

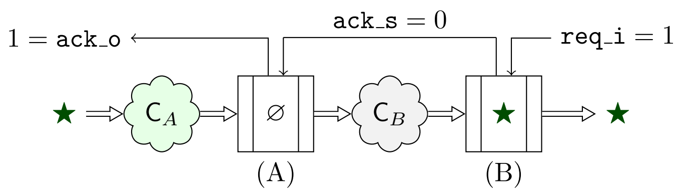

Figure 9: A two-stage pipeline example: initial state.

When `req_i` switches to 0, (B) absorbs the ∅ token 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 $\mathsf{C}_A(\bigstar)$, which is them computed by C_B.

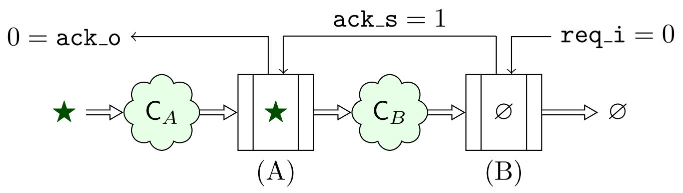

Figure 10: A two-stage pipeline example: `req_i` switches to 0.

The external circuit issues a null token, pre-charging the combinational circuit C_A; the latches (A) and (B) stand by, since its logic state is stable at `req_i` = 0.

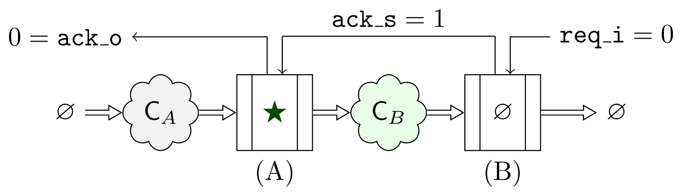

Figure 11: A two-stage pipeline example: the external circuit issues a null token.

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, completing the pre-charge phase of C_A.

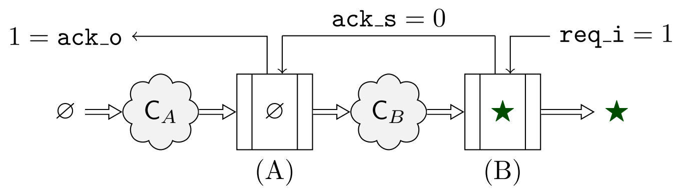

Figure 12: A two-stage pipeline example: `req_i` switches to 1.

Finally, The external circuit issues a new valid token ★. With this new token, the system is back to its initial state, similar to the configuration shown in figure 9.

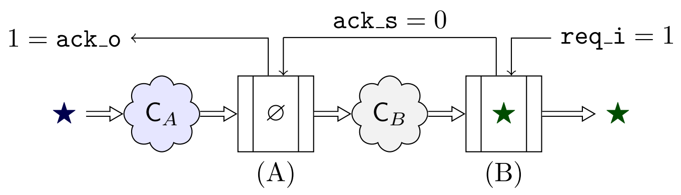

Figure 13: A two-stage pipeline example: the external circuit issues a new valid token.

Whatever is pipeline length, in our designs the latches are initialized with null tokens, similar to configuration shown in figure 9, and need to be loaded before operating in a normal mode. In other words, the latches operate in a normal mode when they are arranged in a manner that there is always null state latch neighboring a valid state one, enabling the pre-charge / evaluate stream. Loading the pipeline can be achieved by setting the `req_i` input active until the moment in which the pipeline is fully ordered, that is, with valid tokens interleaving null state latches. In the following descriptions, we assume that pipeline is already loaded.

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

1. *Area*: compared to the strongly indication version, a tree of c-elements in the token state detection device is replaced by a single traditional 2-input NOR gate, reducing the total silicon area and the acknowledgement logic depth.

2. *Speed*: since a single bit triggers the acknowledgement signal, the latch does not have to wait all valid, or null, signals to recognize its current token. Also, the weakly version has a lower completion detection logic depth, thus it provides its acknowledgement faster. We assume that the Δ_t among the input signals is small enough to permit the absorption of all tokens before the arrival of the next one.

3. *Security*: by using a single bit instead of the whole vector to trigger the latch state detector, we render the system less susceptible to data dependent evaluation time. Indeed, with this last aspect, we want to avoid an intermediate signal timing profile correlated with the sensitive variable. Since the latch only needs to know whenever a valid or a null token is available, a single pair of wires is sufficient to administer the acknowledgement signal.

4 Register-free implementations

In order to evaluate the robustness of our self-timed solutions, this section describes the designs used in this work. Due to its criticality, we focus on the design of the S-box. We start from single-rail implementations, whose performance has been already assured by existing papers, to base our approach and to study the resulting overheads when enabling self-timed features. Finally, since one of our goals is to reduce the latency of this non-linear function to a single clock cycle, the global cipher context is still synchronous. Therefore, no descriptions or implementation results for the full cipher are given in this work, only for their S-boxes.

4.1 PRESENT S-box implementation

Our dual-rail 4-bit PRESENT S-box [BKL+07] is based on the first-order masked with two shares implementation by Shahmirzadi and Moradi [SM21]. Since we are interested in determining potential side-channel leakages due to the time of evaluation profiles when employing self-timed gadgets, we chose this design because no online refresh bits are used and the single-rail implementation security is already assured by the authors. In the original design, there are two register barriers: one to isolate the processing and the compression, and a second barrier at the output. We replace both synchronization layers by weakly indicating asynchronous latches, see figure 7.

Figure 14 shows the steady state of the 2-share self-timed PRESENT S-box used in our work. Besides the latches, there are two combinational blocks P and C, denoting the process and the compress steps, as well as in the original version. The first combinational circuit, the processing, receives four 2-share masked inputs, one for each input bit, and computes four 8-share outputs. Then, the compression reduces the number of output shares from eight down to a pair of shared bits.

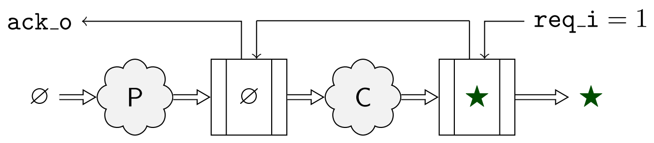

Figure 14: Steady state of the self-timed PRESENT S-box pipeline.

Our scope is a globally synchronous hardware architecture, although the S-box is a self-timed circuit. Thus, we use the rising clock edge to trigger the asynchronous pipeline, shifting the system from an idle state to an operation state and setting the request input of the S-box to `req_i` $= 0$. When the positive clock edge happens, the valid token to be computed is already available in the S-box input. With the request signal set to 0, the null token replaces the valid token in the last latch, pre-charging the compression circuit. With the pre-charge complete, the latch sends the signal that enables the valid token absorption by the first latch.

Meanwhile, the finite state machine (FSM) awaits for the acknowledgement output of the S-box to bet set to 0, indicating that the valid token has been absorbed, and sets the request signal to 1, starting the compression for the current valid token. In parallel, the null token is set at the S-box input, with the purpose of pre-charging the process step. Hence, `ack_o` $= 0$ means that the first stage is ready to be pre-charged. When the acknowledgement toggles, the eight shares have been compressed, and the first step is already pre-charged. The described steps are then repeated for the subsequent valid tokens. If the clock period is adequate, the correct S-box output is already available at the end of the current cycle. The asynchronous FSM described in this paragraph is shown in figure 15. Table 8 shows the S-box input for each FSM state.

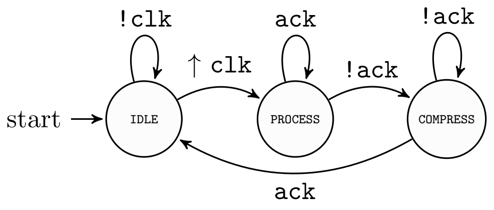

Figure 15: FSM for the PRESENT S-box.

State	S-box inputs	
	token	request
IDLE	∅	1
PROCESS	★	0
COMPRESS	∅	1

Table 8: PRESENT S-box FSM controls.

One of our goals with this work, is to study the impact of refreshing the masks in self-timed circuits. Therefore, relying on the PRESENT S-box implementation without random refresh masks [SM21], we designed the same masked S-box whose component functions are simplified, neglecting their security properties in a first moment. Then, in order to obtain a uniform distributed output, we added two refresh bits per shared function.

4.2 AES S-box implementation

Due to its importance, the Advanced Encryption Standard (AES) S-box [DR00] is a typical benchmark for evaluating countermeasures against side-channel attacks. Thus, this section presents the design of two first order masked self-timed AES S-boxes: one without online refresh bits, whose original version can be found in [SM21], and the *simple* variant of the domain-oriented masked (DOM) AES S-box proposed by Gross et al. in [GMK16], which requires 18 random refresh bits per clock cycle.

As previously announced, we do not aim at comparing both works. In truth, we present our implementation results and contrast with different solutions with the purpose of weighing the overheads. Futhermore, we want to study the security performance of self-timed masking with and without randomness. Thus, we rely on both works whose performance figures are already known.

4.2.1 Self-timed DOM AND

In order to evaluate the masked AES S-box scheme with online mask refresh, instead of simply adding random refresh bits to the design without randomness, we favor the implementation of the 2-share DOM-*simple* S-box. Their first order AND gadget has four registers, which will be replaced by dual-rail latches with the purpose to enable the aforementioned self-timed features. For the same reasons summarized in the section 3.5, we employ the weakly indicating latch, see figure 7. The resulting DOM gadget used in our implementation is shown in figure 16, in which all logic gate symbols express their dual-rail variant.

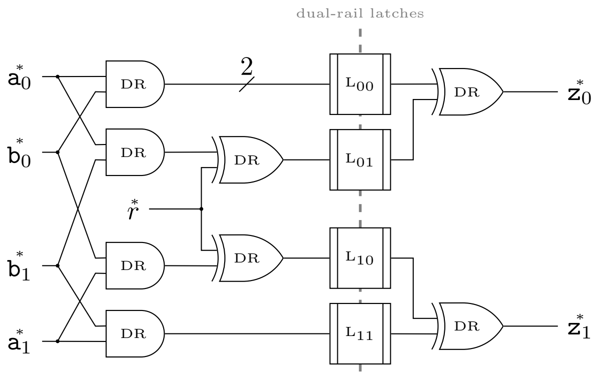

Figure 16: Dual-rail DOM AND.

Since the system is pre-charged before evaluating, the dual-rail XOR gates do not propagate earlier. Therefore, even if the random refresh bit reaches the gates first, the DR-XOR functions do not produce their output before the arrival of the cross-domain products (i.e. $\overset{*}{a}_0 \wedge \overset{*}{b}_1$ and $\overset{*}{a}_1 \wedge \overset{*}{b}_0$). Hence, considering a monotonic behavior, the DR-XOR gate maintains its pre-charged state until the arrival of both valid inputs, toggling its state only once within the evaluate phase. The same reasoning is not true for the pre-charge, since a single null state input is necessary to produce a null output, see table 5. For this reason, although updating the refresh bits after every evaluation, the random input is always valid, since the input sharing is enough to propagate the null tokens among the dual-rail functions.

Moreover, the blocks L_{ij} represent the pair of c-elements storing the product $\mathtt{a}^*_\mathtt{i} \wedge \mathtt{b}^*_\mathtt{j}$. We emphasize that the four pairs of c-elements form a weakly indicating latch. The handshake signals are implicit in figure 16, albeit the logic is the same: there is an output acknowledgement signal that indicates the latch state and a request signal, whose logic level controls the token flow.

4.2.2 The token flow in the self-timed AES S-box

Both S-box designs are based on the Canright's implementation [Can05] and have eight combinational stages, which are summarized in table 9. For further descriptions of each AES S-box stage, the reader may refer to the the original works [GMK16, SM21].

Table 9: Combinational stages in both AES S-boxes.

	Shahmirzadi and Moradi [SM21]	Gross et al. [GMK16]
STAGE 1	$GF(2^8) \longrightarrow GF(((2^2)^2)^2)$	$GF(2^8) \longrightarrow GF(((2^2)^2)^2)$
STAGE 2	$GF(2^4)$ square-scale-multipliers[a]	$GF(2^4)$ square-scale-multiplier
STAGE 3	compression	$GF(2^4)$ addition
STAGE 4	$GF(2^4)$ inverter	$GF(2^2)$ square-scale-multiplier
STAGE 5	compression	$GF(2^2)$ inverter
STAGE 6	$GF(2^4)$ multipliers	$GF(2^2)$ multipliers
STAGE 7	compression	$GF(2^4)$ multipliers
STAGE 8	$GF(((2^2)^2)^2) \longrightarrow GF(2^8)$	$GF(((2^2)^2)^2) \longrightarrow GF(2^8)$

[a]This version uses two jointly-uniform square-scale-multipliers [SM21]

Since the two self-timed AES S-box architectures have eight combinational stages and eight latches, the handshake logic is identical for both. A high-level representation of the self-timed system is shown in figure 17 in which the combinational circuits C refer to the the eight S-box stages from table 9. The figure shows the pipeline steady state when the system is ready to compute a new byte.

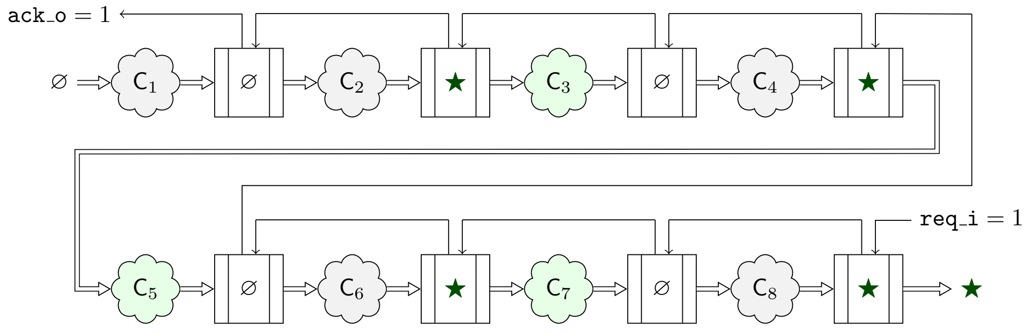

Figure 17: Steady state of the self-timed AES S-box pipeline.

As stated before, our scope is globally synchronous and locally asynchronous. In this context, a positive clock edge triggers the domino logic, allowing us to synchronize the computation of the correct token. If the clock period is adequate, the correct S-box output is ready before the next positive edge, achieving a single cycle masked AES S-box operation.

As well as the PRESENT S-box implementation case, the FSM has three states — `IDLE`, `COMPRESS` and `PROCESS` — with similar conditions for the state transition diagram.

However, since the AES pipeline has more stages, we count the number of occurrences of the positive acknowledgement edge in order to track the desired token progression in the pipeline. Within a full AES S-box computation the system issues four null tokens for the pre-charge. Hence, the FSM counts to four before it can return to the idle state. The counter is update once when reaching the `PROCESS` state.

Figure 18 shows the FSM for the self-timed AES S-boxes with the signal `done = 1` if the counter is equal to 4 and deactivated otherwise. The S-box inputs are shown in table 10 and the `IDLE` state resets the counter.

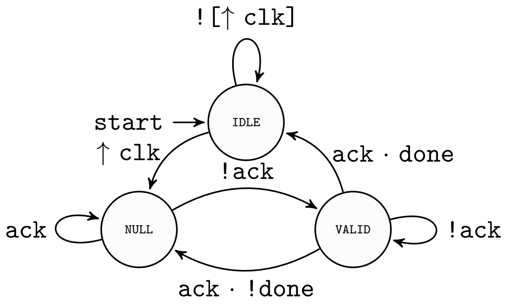

Figure 18: FSM for the AES S-box.

State	Counter	S-box inputs token	S-box inputs request
`IDLE`	reset	∅	1
`NULL`	++	★	0
`VALID`	keep	∅	1

Table 10: AES S-box FSM controls.

5 Implementation results

This section summarizes the area and performance results of our hardware implementations. All designs were described using the hardware description language (HDL) Verilog and we use Synopsys Design Compiler S-2021.06-SP1 with a target frequency of 100MHz to synthesize the netlists. The standard cell library used in the synthesis flow was STMicroelectronics CMOSM40. 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.

Table 11: Performance figures for the self-timed masked S-box implementations using Synopsis Design Compiler and ST CMOSM40 standard cell library. No `compile_ultra`.

S-box	Version	Area [kGE]	Refresh [bits]	Latency [cycles]	Delay [ns]
PRESENT	no refresh	0.99	0	1	≈ 12
PRESENT	with refresh	1.02	8	1	≈ 12
AES	no refresh	7.79	0	1	≈ 220
AES	domain oriented	6.07	18	1	≈ 208

5.1 Comparison with related works

We refer to table 12, which reports the performance figures of our implementations compared to the state of the art.

Table 12: Performance figures of different masked S-box implementations.

Design	Shares	Area [kGE]	Refresh [bits]	Latency [cycles]	Standard Cell Library
PRESENT [PMK+11]	3	0.36	0	1	UCM 180 nm
PRESENT [*this work*]	2	0.99	0	1	STM 40 nm
PRESENT [*this work*]	2	1.02	8	1	STM 40 nm
AES [UHA17]	2	1.4	64	5	TSMC 65-nm
AES [WM18]	4	4.2	0	16	UCM 180 nm
AES [Sug19]	3	3.5	0	4	NanGate 45-nm
AES [GMK16]	2	2.8	28	5	UCM 180 nm
AES [GMK16]	2	2.6	18	8	UCM 180 nm
AES [*this work*]	2	7.79	0	1	STM 40 nm
AES [*this work*]	2	6.07	18	1	STM 40 nm

As previously stated, synchronizing the intermediate shares at the boundaries of combinational blocks is of high importance to obtain a secure masking implementation. Thus, this work presented a generic solution, that may be applied to different S-box designs, 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 by 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.

6 Side-Channel Analysis

In order to evaluate the robustness of our masked implementations against practical side-channel analysis, we apply the test vector leakage assessment (TVLA) methodology proposed by Goodwill et al. [GJJR11]. It uses the Welch's t-test to determine whether the difference of two dataset means provides sufficient evidence to reject the null hypothesis.

We use simulated traces in order to evaluate the side-channel vulnerability of our designs due to data-dependent time of evaluation. Our goal is to model the system's power consumption in a noiseless manner with a timing resolution of 1 ρs. For that purpose, power traces were simulated from value change dump (VCD) files generated by netlist simulation on Mentor Graphics ModelSim. This method allows us to model the power consumption by counting the toggling activity of all wires in the device under test (DUT).

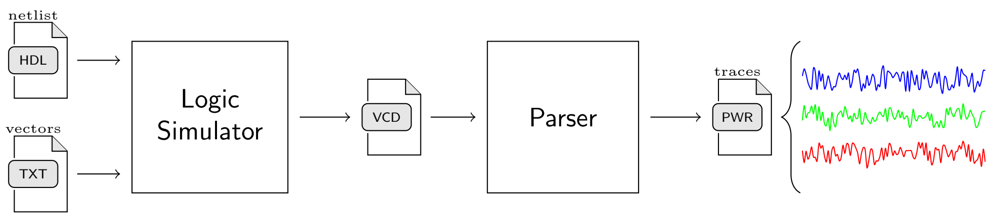

Figure 19: Simulating power traces from VCD files.

The resulting traces will differ depending on the standard cell library. Nevertheless, the following analysis gives us clues about potential pitfalls, and how to locate them in order to improve the final design. Hence, when the timing behavior is a critical aspect when dealing with side-channel leakages, which is potentially the case in a self-timed circuit, a noiseless high resolution power analysis is very pertinent. Figures 20 and 21 show the TVLA analysis for one million simulated traces for the self-timed PRESENT and AES S-boxes without mask refreshing.

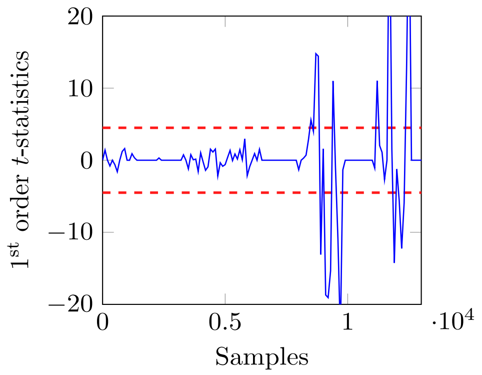

Figure 20: TVLA results based on one million simulated traces for our self-timed 2-shares PRESENT S-box without mask refreshing.

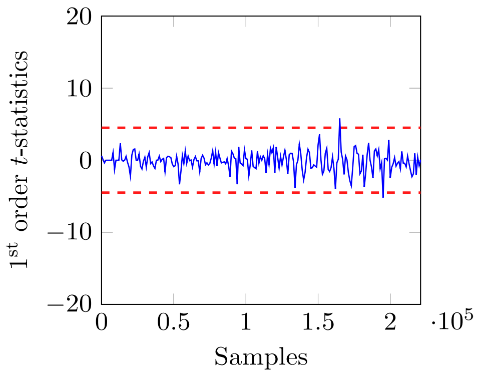

Figure 21: TVLA results based on one million simulated traces for our self-timed 2-shares AES S-box without mask refreshing.

In this noiseless scenario, when no random refresh bits are employed, potential exploitable leakages were identified for both self-timed PRESENT and AES S-boxes. We believe this security flaw is due to an intermediate signal evaluation time profile tied to the sensitive variable. Indeed, since we are observing a noiseless power consumption model based on wire toggling, the switching activity may behave as a signature of the circuit's internal state for a given sharing. As no random mask refreshing is used, a specific pair of input shares produces a particular pair of output shares, causing a consistent internal activity.

Presumably, the larger the combinational data path, the more detectable this data signature may be in the circuit. Thus, in this case, a cryptographic application would be more susceptible to a multivariate analysis, when the univariate leakage assessment is not enough to detect potential flaws. We reiterate the fact that our first order masked S-box implementations without fresh randomness failed in a univariate analysis using simulated traces.

Based on the last argumentation, we must highlight the importance of the clock in defining the cadence of the data flow, which reduces the aforementioned time dependence. Even if timing leakages tied to the data propagation may not be spotted in real digital circuits, in which the signal to noise ration is lower, a constant period synchronization layer limits the combinational critical path, restraining its exploitation by the attacker. Nevertheless, timing leakages exist, as argued from the noiseless results obtained from toggle count modeling.

In parallel, online refresh bits is also a solution for mitigating the data dependence time of evaluation through the combinational path. Not unreasonably, we show in this work two similar masked S-box schemes: one with random refresh bits and another whose online randomness cost is cancelled. The intermediate variables are consistent for a given pair of shares when no refresh bits is present, hence there is an unique toggle count behavior for this same pair. However, the same cannot be said when online refresh bits are present, since the correctness depends on a set of external random information.

Figures 22 and 23 show the TVLA results using one million simulated traces for the self-timed PRESENT S-box with random refresh bits and the *simple* DOM implementation of the AES S-box, respectively. We believe that the discrepancy between this analysis and the previous results is due to mask refreshing, a randomly distributed information that shuffles the internal signal logic while maintaining their correctness.

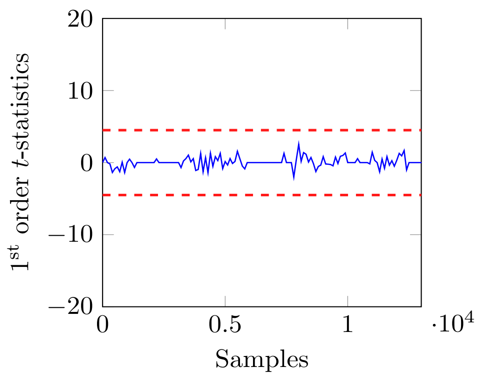

Figure 22: TVLA results based on one million simulated traces for our self-timed 2-shares PRESENT S-box with mask refreshing.

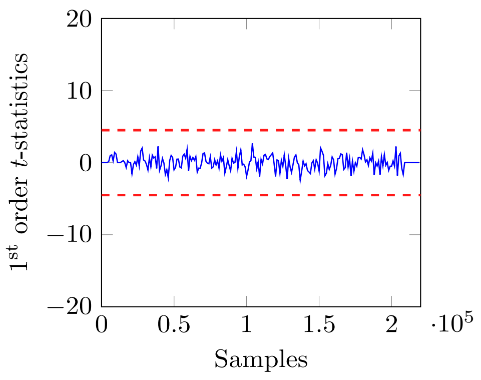

Figure 23: TVLA results based on one million simulated traces for our self-timed 2-shares AES S-box with mask refreshing.

We do not deny the possibility of, with enough high resolution and noiseless traces, observing exploitable side-channel leakages using the current analysis again. In fact, one could say that, mask refreshing makes the circuit noisier, which makes the data dependent time of evaluation assessment more complex. However, we perform a noiseless side-channel analysis. Despite not presenting real side-channel measurements, the presence of noise in an FPGA or ASIC application may mitigate such leakages.

7 Conclusion

In this work, we have shown how to construct self-timed masking schemes based on the Muller c-elements. Although most examples on which we rely to build our designs are already resistant against glitches, based on the glitch-extended probing model analysis, we detail a method to eliminate glitches in combinatorial circuits using monotonic logic. This method can be easily applied in different implementations when glitch-extended cannot be achieved. Also, we show how to avoid early propagation in dual-rail functions.

In order to evaluate our designs, we describe the implementation of self-timed PRESENT and AES S-boxes and provide leakage assessment results based on noiseless side-channel analysis. The motivation to perform a noiseless leakage assessment is to observe potential timing leakages due to the self-timed behavior of our S-boxes. Based on TVLA results, we stress the importance of fresh randomness in our register-free implementations. Indeed, if the signal to noise is high enough, data-dependent evaluation time is a potential pitfall, leading to first-order leakages. Hence, shuffling the intermediate signals with fresh masks has shown to be an efficient way to mitigate this flaw.

Finally, despite the area and throughput overheads, our designs may be an interesting solution when latency is a critical aspect. In truth, the main asset of our work is a generic way to reduce the latency of masked S-boxes to a single clock cycle, while preserving secure masking properties.

References

[BBD$^+$16] Gilles Barthe, Sonia Belaïd, François Dupressoir, Pierre-Alain Fouque, Benjamin Grégoire, Pierre-Yves Strub, and Rébecca Zucchini. Strong non-interference and type-directed higher-order masking. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, pages 116–129. ACM, 2016.

[BKL$^+$07] Andrey Bogdanov, Lars R. Knudsen, Gregor Leander, Christof Paar, Axel Poschmann, Matthew J. B. Robshaw, Yannick Seurin, and C. Vikkelsoe. PRESENT: an ultra-lightweight block cipher. In Paillier and Verbauwhede [PV07], pages 450–466.

[Can05] David Canright. A very compact S-box for AES. In Rao and Sunar [RS05], pages 441–455.

[CZ06] Zhimin Chen and Yujie Zhou. Dual-rail random switching logic: A countermeasure to reduce side channel leakage. In Goubin and Matsui [GM06], pages 242–254.

[Dae17] Joan Daemen. Changing of the guards: A simple and efficient method for achieving uniformity in threshold sharing. In Wieland Fischer and Naofumi Homma, editors, *Cryptographic Hardware and Embedded Systems - CHES 2017 - 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings*, volume 10529 of *Lecture Notes in Computer Science*, pages 137–153. Springer, 2017.

[DN95] Al Davis and Steven M Nowick. Asynchronous circuit design: Motivation, background, & methods. In *Asynchronous Digital Circuit Design*, pages 1–49. Springer, 1995.

[DR00] Joan Daemen and Vincent Rijmen. Rijndael for AES. In *The Third Advanced Encryption Standard Candidate Conference, April 13-14, 2000, New York, New York, USA*, pages 343–348. National Institute of Standards and Technology,, 2000.

[FGP$^+$18] Sebastian Faust, Vincent Grosso, Santos Merino Del Pozo, Clara Paglialonga, and François-Xavier Standaert. Composable masking schemes in the presence of physical defaults & the robust probing model. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2018(3):89–120, 2018.

[GIB18] Hannes Groß, Rinat Iusupov, and Roderick Bloem. Generic low-latency masking in hardware. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2018(2):1–21, 2018.

[GJJR11] Gilbert Goodwill, Benjamin Jun, Joshua Jaffe, and Pankaj Rohatgi. A testing methodology for side-channel resistance validation. In *NIST non-invasive attack testing (NIAT) workshop*, volume 7, pages 115–136, 2011.

[GM06] Louis Goubin and Mitsuru Matsui, editors. *Cryptographic Hardware and Embedded Systems - CHES 2006, 8th International Workshop, Yokohama, Japan, October 10-13, 2006, Proceedings*, volume 4249 of *Lecture Notes in Computer Science*. Springer, 2006.

[GMK16] Hannes Groß, Stefan Mangard, and Thomas Korak. Domain-oriented masking: Compact masked hardware implementations with arbitrary protection order. In Begül Bilgin, Svetla Nikova, and Vincent Rijmen, editors, *Proceedings of the ACM Workshop on Theory of Implementation Security, TIS@CCS 2016 Vienna, Austria, October, 2016*, page 3. ACM, 2016.

[GP99] Louis Goubin and Jacques Patarin. DES and differential power analysis (the "duplication" method). In Çetin Kaya Koç and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems, First International Workshop, CHES'99, Worcester, MA, USA, August 12-13, 1999, Proceedings*, volume 1717 of *Lecture Notes in Computer Science*, pages 158–172. Springer, 1999.

[ISW03] Yuval Ishai, Amit Sahai, and David A. Wagner. Private circuits: Securing hardware against probing attacks. In Dan Boneh, editor, *Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings*, volume 2729 of *Lecture Notes in Computer Science*, pages 463–481. Springer, 2003.

[Juk21] Stasys Jukna. Notes on hazard-free circuits. *SIAM J. Discret. Math.*, 35(2):770–787, 2021.

[KJJ99] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In Michael J. Wiener, editor, *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer, 1999.

[KKT06] Konrad J. Kulikowski, Mark G. Karpovsky, and Alexander Taubin. 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*, pages 131–138. IEEE Computer Society, 2006.

[LMW14] Andrew J. Leiserson, Mark E. Marson, and Megan A. Wachs. Gate-level masking under a path-based leakage metric. In Lejla Batina and Matthew Robshaw, editors, *Cryptographic Hardware and Embedded Systems - CHES 2014 - 16th International Workshop, Busan, South Korea, September 23-26, 2014. Proceedings*, volume 8731 of *Lecture Notes in Computer Science*, pages 580–597. Springer, 2014.

[MB59] David E. Muller and W. S. Bartky. A theory of asynchronous circuits. In *Proceedings of an International Symposium on the Theory of Switchinge, April 1957, Part I*, volume XXIX of *the annals of the computation laboratory of Harvard University*, pages 204–243, Cambridge, MA, USA, 1959. Cambridge University Press.

[MMSS19] Thorben Moos, Amir Moradi, Tobias Schneider, and François-Xavier Standaert. Glitch-resistant masking revisited or why proofs in the robust probing model are needed. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2019(2):256–292, 2019.

[MPG05] Stefan Mangard, Thomas Popp, and Berndt M. Gammel. Side-channel leakage of masked CMOS gates. In Alfred 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.

[MPO05] Stefan Mangard, Norbert Pramstaller, and Elisabeth Oswald. Successfully attacking masked AES hardware implementations. In Rao and Sunar [RS05], pages 157–171.

[NRR06] Svetla Nikova, Christian Rechberger, and Vincent Rijmen. Threshold implementations against side-channel attacks and glitches. In Peng Ning, Sihan Qing, and Ninghui Li, editors, *Information and Communications Security, 8th International Conference, ICICS 2006, Raleigh, NC, USA, December 4-7, 2006, Proceedings*, volume 4307 of *Lecture Notes in Computer Science*, pages 529–545. Springer, 2006.

[PKZM07] Thomas Popp, Mario Kirschbaum, Thomas Zefferer, and Stefan Mangard. Evaluation of the masked logic style MDPL on a prototype chip. In Paillier and Verbauwhede [PV07], pages 81–94.

[PM05] Thomas Popp and Stefan Mangard. Masked dual-rail pre-charge logic: DPA-resistance without routing constraints. In Rao and Sunar [RS05], pages 172–186.

[PMK$^+$11] Axel Poschmann, Amir Moradi, Khoongming Khoo, Chu-Wee Lim, Huaxiong Wang, and San Ling. Side-channel resistant crypto for less than 2, 300 GE. *J. Cryptol.*, 24(2):322–345, 2011.

[PV07] Pascal Paillier and Ingrid Verbauwhede, editors. *Cryptographic Hardware and Embedded Systems - CHES 2007, 9th International Workshop, Vienna, Austria, September 10-13, 2007, Proceedings*, volume 4727 of *Lecture Notes in Computer Science*. Springer, 2007.

[RBN$^+$15] Oscar Reparaz, Begül Bilgin, Svetla Nikova, Benedikt Gierlichs, and Ingrid Verbauwhede. Consolidating masking schemes. In Rosario Gennaro and Matthew Robshaw, editors, *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part I*, volume 9215 of *Lecture Notes in Computer Science*, pages 764–783. Springer, 2015.

[RS05] Josyula R. Rao and Berk Sunar, editors. *Cryptographic Hardware and Embedded Systems - CHES 2005, 7th International Workshop, Edinburgh, UK, August 29 - September 1, 2005, Proceedings*, volume 3659 of *Lecture Notes in Computer Science*. Springer, 2005.

[SBHM20] Pascal Sasdrich, Begül Bilgin, Michael Hutter, and Mark E. Marson. Low-latency hardware masking with application to AES. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2020(2):300–326, 2020.

[SM21] Aein Rezaei Shahmirzadi and Amir Moradi. Re-consolidating first-order masking schemes nullifying fresh randomness. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2021(1):305–342, 2021.

[Spa20] Jens Sparsø. *Introduction to Asynchronous Circuit Design.* DTU Compute, Technical University of Denmark, 2020. Paperback edition available here: https://www.amazon.com/dp/B08BF2PFLN.

[SS06] Daisuke Suzuki and Minoru Saeki. Security evaluation of DPA countermeasures using dual-rail pre-charge logic style. In Goubin and Matsui [GM06], pages 255–269.

[Sug19] Takeshi Sugawara. 3-share threshold implementation of AES S-box without fresh randomness. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2019(1):123–145, 2019.

[TV04] Kris Tiri and Ingrid Verbauwhede. 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*, pages 246–251. IEEE Computer Society, 2004.

[UHA17] Rei Ueno, Naofumi Homma, and Takafumi Aoki. A systematic design of tamper-resistant galois-field arithmetic circuits based on threshold implementation with (d + 1) input shares. In *47th IEEE International Symposium on Multiple-Valued Logic, ISMVL 2017, Novi Sad, Serbia, May 22-24, 2017*, pages 136–141. IEEE Computer Society, 2017.

[WM18] Felix Wegener and Amir Moradi. A first-order SCA resistant AES without fresh randomness. In Junfeng Fan and Benedikt Gierlichs, editors, *Constructive Side-Channel Analysis and Secure Design - 9th International Workshop, COSADE 2018, Singapore, April 23-24, 2018, Proceedings*, volume 10815 of *Lecture Notes in Computer Science*, pages 245–262. Springer, 2018.

A PRESENT s-box without fresh masks [SM21]

$F(a,b,c,d) : C56B90AD3EF84712$
$w = f^0(a,b,c,d) = a \oplus c \oplus d \oplus bc$
$x = f^1(a,b,c,d) = b \oplus d \oplus bd \oplus cd \oplus abc \oplus abd \oplus acd$
$y = f^2(a,b,c,d) = 1 \oplus c \oplus d \oplus ab \oplus ad \oplus bd \oplus abd \oplus acd$
$z = f^3(a,b,c,d) = 1 \oplus a \oplus b \oplus d \oplus bc \oplus abc \oplus abd \oplus acd$

Process Step

$$
\begin{array}{rlcc}
f^0_0(a_1,b_0,c_0,d_0) = & b_0c_0 \oplus d_0 \oplus c_0 \oplus a_1 & \rightarrow & w'_0 \\
f^0_1(a_1,b_0,c_1,d_1) = & b_0c_1 \oplus c_1 \oplus a_1 & \rightarrow & w'_1 \\
\hline
f^0_2(a_1,b_1,c_0,d_0) = & b_1c_0 \oplus a_1 & \rightarrow & w'_2 \\
f^0_3(a_0,b_1,c_1,d_1) = & b_1c_1 \oplus d_1 \oplus a_0 & \rightarrow & w'_3 \\
 & & & \\
f^1_0(a_1,b_0,c_0,d_0) = & a_1c_0d_0 \oplus a_1b_0d_0 \oplus a_1b_0c_0 \oplus a_1b_0 & \rightarrow & x'_0 \\
f^1_1(a_0,b_0,c_0,d_1) = & a_0c_0d_1 \oplus a_0b_0d_1 \oplus a_0b_0c_0 \oplus a_0d_1 & \rightarrow & x'_1 \\
f^1_2(a_0,b_0,c_1,d_0) = & a_0c_1d_0 \oplus a_0b_0d_0 \oplus a_0b_0c_1 \oplus a_0c_1 \oplus c_1d_0 \oplus b_0d_0 \oplus b_0c_1 \oplus c_1 & \rightarrow & x'_2 \\
f^1_3(a_1,b_0,c_1,d_1) = & a_1c_1d_1 \oplus a_1b_0d_1 \oplus a_1b_0c_1 \oplus b_0c_1 \oplus & & \\
 & \quad \oplus b_0d_1 \oplus c_1d_1 \oplus a_1b_0 \oplus a_1c_1 \oplus a_1d_1 \oplus b_0 \oplus c_1 & \rightarrow & x'_3 \\
\hline
f^1_4(a_0,b_1,c_0,d_0) = & a_0c_0d_0 \oplus a_0b_1d_0 \oplus a_0b_1c_0 \oplus b_1c_0 \oplus & & \\
 & \quad \oplus b_1d_0 \oplus c_0d_0 \oplus a_0b_1 \oplus b_1 \oplus d_0 \oplus a_0 & \rightarrow & x'_4 \\
f^1_5(a_1,b_1,c_0,d_1) = & a_1c_0d_1 \oplus a_1b_1d_1 \oplus a_1b_1c_0 \oplus b_1c_0 \oplus b_1d_1 \oplus c_0d_1 \oplus a_1d_1 \oplus d_1 & \rightarrow & x'_5 \\
f^1_6(a_1,b_1,c_1,d_0) = & a_1c_1d_0 \oplus a_1b_1d_0 \oplus a_1b_1c_1 \oplus a_1c_1 & \rightarrow & x'_6 \\
f^1_7(a_0,b_1,c_1,d_1) = & a_0c_1d_1 \oplus a_0b_1d_1 \oplus a_0b_1c_1 \oplus a_0b_1 \oplus a_0c_1 \oplus a_0d_1 \oplus a_0 & \rightarrow & x'_7 \\
 & & & \\
f^2_0(a_1,b_0,c_0,d_0) = & a_1c_0d_0 \oplus a_1b_0d_0 \oplus a_1b_0 \oplus a_1c_0 & \rightarrow & y'_0 \\
f^2_1(a_0,b_0,c_0,d_1) = & a_0c_0d_1 \oplus a_0b_0d_1 \oplus a_0b_0 \oplus a_0c_0 \oplus a_0d_1 \oplus a_0 & \rightarrow & y'_1 \\
f^2_2(a_0,b_0,c_1,d_0) = & a_0c_1d_0 \oplus a_0b_0d_0 \oplus b_0d_0 \oplus c_1d_0 & \rightarrow & y'_2 \\
f^2_3(a_1,b_0,c_1,d_1) = & a_1c_1d_1 \oplus a_1b_0d_1 \oplus b_0d_1 \oplus c_1d_1 \oplus a_1d_1 \oplus c_1 \oplus a_1 & \rightarrow & y'_3 \\
\hline
f^2_4(a_0,b_1,c_0,d_0) = & a_0c_0d_0 \oplus a_0b_1d_0 \oplus a_0b_1 \oplus a_0c_0 \oplus a_0d_0 \oplus a_0 & \rightarrow & y'_4 \\
f^2_5(a_1,b_1,c_0,d_1) = & a_1c_0d_1 \oplus a_1b_1d_1 \oplus a_1b_1 \oplus a_1c_0 \oplus c_0 \oplus d_1 \oplus a_1 & \rightarrow & y'_5 \\
f^2_6(a_1,b_1,c_1,d_0) = & a_1c_1d_0 \oplus a_1b_1d_0 \oplus b_1d_0 \oplus c_1d_0 \oplus a_1d_0 \oplus d_0 & \rightarrow & y'_6 \\
f^2_7(a_0,b_1,c_1,d_1) = & a_0c_1d_1 \oplus a_0b_1d_1 \oplus b_1d_1 \oplus c_1d_1 & \rightarrow & y'_7 \\
 & & & \\
f^3_0(a_1,b_0,c_0,d_0) = & a_1c_0d_0 \oplus a_1b_0d_0 \oplus a_1b_0 & \rightarrow & z'_0 \\
f^3_1(a_0,b_0,c_0,d_1) = & a_0c_0d_1 \oplus a_0b_0d_1 \oplus b_0c_0 \oplus b_0d_1 \oplus c_0d_1 \oplus a_0d_1 \oplus d_1 & \rightarrow & z'_1 \\
f^3_2(a_0,b_0,c_1,d_0) = & a_0c_1d_0 \oplus a_0b_0d_0 \oplus a_0c_1 & \rightarrow & z'_2 \\
f^3_3(a_1,b_0,c_1,d_1) = & a_1c_1d_1 \oplus a_1b_0d_1 \oplus b_0c_1 \oplus b_0d_1 \oplus & & \\
 & \quad \oplus c_1d_1 \oplus a_1b_0 \oplus a_1c_1 \oplus a_1d_1 \oplus b_0 \oplus c_1 \oplus a_1 & \rightarrow & z'_3 \\
\hline
f^3_4(a_0,b_1,c_0,d_0) = & a_0c_0d_0 \oplus a_0b_1d_0 \oplus a_0b_1 \oplus d_0 & \rightarrow & z'_4 \\
f^3_5(a_1,b_1,c_0,d_1) = & a_1c_0d_1 \oplus a_1b_1d_1 \oplus b_1c_0 \oplus b_1d_1 \oplus c_0d_1 \oplus a_1d_1 \oplus d_1 & \rightarrow & z'_5 \\
f^3_6(a_1,b_1,c_1,d_0) = & a_1c_1d_0 \oplus a_1b_1d_0 \oplus a_1c_1 & \rightarrow & z'_6 \\
f^3_7(a_0,b_1,c_1,d_1) = & a_0c_1d_1 \oplus a_0b_1d_1 \oplus b_1c_1 \oplus b_1d_1 \oplus & & \\
 & \quad \oplus c_1d_1 \oplus a_0b_1 \oplus a_0c_1 \oplus a_0d_1 \oplus b_1 \oplus c_1 \oplus d_1 \oplus a_0 & \rightarrow & z'_7
\end{array}
$$

Compress Step

$$
\begin{array}{ll}
w_0 = w'_0 \oplus w'_1 & w_1 = w'_2 \oplus w'_3 \\
x_0 = x'_0 \oplus x'_1 \oplus x'_2 \oplus x'_3 & x_1 = x'_4 \oplus x'_5 \oplus x'_6 \oplus x'_7 \\
y_0 = y'_0 \oplus y'_1 \oplus y'_2 \oplus y'_3 & y_1 = y'_4 \oplus y'_5 \oplus y'_6 \oplus y'_7 \\
z_0 = z'_0 \oplus z'_1 \oplus z'_2 \oplus z'_3 & z_1 = z'_4 \oplus z'_5 \oplus z'_6 \oplus z'_7
\end{array}
$$

B 2-share masked PRESENT s-box with fresh masks

$F(a,b,c,d) : C56B90AD3EF84712$
$w = f^0(a,b,c,d) = a \oplus c \oplus d \oplus bc$
$x = f^1(a,b,c,d) = b \oplus d \oplus bd \oplus cd \oplus abc \oplus abd \oplus acd$
$y = f^2(a,b,c,d) = 1 \oplus c \oplus d \oplus ab \oplus ad \oplus bd \oplus abd \oplus acd$
$z = f^3(a,b,c,d) = 1 \oplus a \oplus b \oplus d \oplus bc \oplus abc \oplus abd \oplus acd$

$$
\begin{array}{rcl rl}
f^0_0(a_1,b_0,c_0,d_0) & = & b_0c_0 \oplus c_0 \oplus a_1 \oplus r^0 \oplus r^4 & \rightarrow & w'_0 \\
f^0_1(a_1,b_0,c_1,d_1) & = & b_0c_1 \oplus d_1 \oplus r^3 \oplus r^7 & \rightarrow & w'_1 \\
\hline
f^0_2(a_1,b_1,c_0,d_0) & = & b_1c_0 \oplus d_0 \oplus r^0 \oplus r^7 & \rightarrow & w'_2 \\
f^0_3(a_0,b_1,c_1,d_1) & = & b_1c_1 \oplus c_1 \oplus a_0 \oplus r^3 \oplus r^4 & \rightarrow & w'_3 \\
 & & & & \\
f^1_0(a_1,b_0,c_0,d_0) & = & a_1c_0d_0 \oplus a_1b_0d_0 \oplus a_1b_0c_0 \oplus c_0d_0 \oplus r^0 \oplus r^4 & \rightarrow & x'_0 \\
f^1_1(a_0,b_0,c_0,d_1) & = & a_0c_0d_1 \oplus a_0b_0d_1 \oplus a_0b_0c_0 \oplus b_0d_1 \oplus d_1 \oplus r^1 \oplus r^5 & \rightarrow & x'_1 \\
f^1_2(a_0,b_0,c_1,d_0) & = & a_0c_1d_0 \oplus a_0b_0d_0 \oplus a_0b_0c_1 \oplus b_0d_0 \oplus r^2 \oplus r^6 & \rightarrow & x'_2 \\
f^1_3(a_1,b_0,c_1,d_1) & = & a_1c_1d_1 \oplus a_1b_0d_1 \oplus a_1b_0c_1 \oplus c_1d_1 \oplus b_3 \oplus r^0 \oplus r^7 & \rightarrow & x'_3 \\
\hline
f^1_4(a_0,b_1,c_0,d_0) & = & a_0c_0d_0 \oplus a_0b_1d_0 \oplus a_0b_1c_0 \oplus b_1d_0 \oplus d_0 \oplus r^0 \oplus r^7 & \rightarrow & x'_4 \\
f^1_5(a_1,b_1,c_0,d_1) & = & a_1c_0d_1 \oplus a_1b_1d_1 \oplus a_1b_1c_0 \oplus c_0d_1 \oplus r^1 \oplus r^6 & \rightarrow & x'_5 \\
f^1_6(a_1,b_1,c_1,d_0) & = & a_1c_1d_0 \oplus a_1b_1d_0 \oplus a_1b_1c_1 \oplus c_1d_0 \oplus b_1 \oplus r^2 \oplus r^5 & \rightarrow & x'_6 \\
f^1_7(a_0,b_1,c_1,d_1) & = & a_0c_1d_1 \oplus a_0b_1d_1 \oplus a_0b_1c_1 \oplus b_1d_1 \oplus r^3 \oplus r^4 & \rightarrow & x'_7 \\
 & & & & \\
f^2_0(a_1,b_0,c_0,d_0) & = & 1 \oplus a_1c_0d_0 \oplus a_1b_0d_0 \oplus b_0d_0 \oplus c_0 \oplus r^0 \oplus r^4 & \rightarrow & y'_0 \\
f^2_1(a_0,b_0,c_0,d_1) & = & a_0c_0d_1 \oplus a_0b_0d_1 \oplus b_0d_1 \oplus a_0d_1 \oplus r^1 \oplus r^5 & \rightarrow & y'_1 \\
f^2_2(a_0,b_0,c_1,d_0) & = & a_0c_1d_0 \oplus a_0b_0d_0 \oplus a_0d_0 \oplus a_0b_0 \oplus r^2 \oplus r^6 & \rightarrow & y'_2 \\
f^2_3(a_1,b_0,c_1,d_1) & = & a_1c_1d_1 \oplus a_1b_0d_1 \oplus a_1b_0 \oplus c_1 \oplus r^3 \oplus r^7 & \rightarrow & y'_3 \\
\hline
f^2_4(a_0,b_1,c_0,d_0) & = & a_0c_0d_0 \oplus a_0b_1d_0 \oplus b_1d_0 \oplus d_0 \oplus r^0 \oplus r^7 & \rightarrow & y'_4 \\
f^2_5(a_1,b_1,c_0,d_1) & = & a_1c_0d_1 \oplus a_1b_1d_1 \oplus b_1d_1 \oplus a_1d_1 \oplus r^1 \oplus r^6 & \rightarrow & y'_5 \\
f^2_6(a_1,b_1,c_1,d_0) & = & a_1c_1d_0 \oplus a_1b_1d_0 \oplus a_1d_0 \oplus a_1b_1 \oplus r^2 \oplus r^5 & \rightarrow & y'_6 \\
f^2_7(a_0,b_1,c_1,d_1) & = & a_0c_1d_1 \oplus a_0b_1d_1 \oplus a_0b_1 \oplus d_1 \oplus r^3 \oplus r^4 & \rightarrow & y'_7 \\
 & & & & \\
f^3_0(a_1,b_0,c_0,d_0) & = & 1 \oplus a_1c_0d_0 \oplus a_1b_0d_0 \oplus a_1b_0c_0 \oplus b_0c_0 \oplus r^0 \oplus r^4 & \rightarrow & z'_0 \\
f^3_1(a_0,b_0,c_0,d_1) & = & a_0c_0d_1 \oplus a_0b_0d_1 \oplus a_0b_0c_0 \oplus d_1 \oplus a_0 \oplus r^1 \oplus r^5 & \rightarrow & z'_1 \\
f^3_2(a_0,b_0,c_1,d_0) & = & a_0c_1d_0 \oplus a_0b_0d_0 \oplus a_0b_0c_1 \oplus b_0c_1 \oplus r^2 \oplus r^6 & \rightarrow & z'_2 \\
f^3_3(a_1,b_0,c_1,d_1) & = & a_1c_1d_1 \oplus a_1b_0d_1 \oplus a_1b_0c_1 \oplus b_0 \oplus r^3 \oplus r^7 & \rightarrow & z'_3 \\
\hline
f^3_4(a_0,b_1,c_0,d_0) & = & a_0c_0d_0 \oplus a_0b_1d_0 \oplus a_0b_1c_0 \oplus d_0 \oplus r^0 \oplus r^7 & \rightarrow & z'_4 \\
f^3_5(a_1,b_1,c_0,d_1) & = & a_1c_0d_1 \oplus a_1b_1d_1 \oplus a_1b_1c_0 \oplus b_1c_0 \oplus r^1 \oplus r^6 & \rightarrow & z'_5 \\
f^3_6(a_1,b_1,c_1,d_0) & = & a_1c_1d_0 \oplus a_1b_1d_0 \oplus a_1b_1c_1 \oplus b_1 \oplus a_1 \oplus r^2 \oplus r^5 & \rightarrow & z'_6 \\
f^3_7(a_0,b_1,c_1,d_1) & = & a_0c_1d_1 \oplus a_0b_1d_1 \oplus a_0b_1c_1 \oplus b_1c_1 \oplus r^3 \oplus r^4 & \rightarrow & z'_7
\end{array}
$$

$$
\begin{array}{ll}
w_0 = w'_0 \oplus w'_1 & w_1 = w'_2 \oplus w'_3 \\
x_0 = x'_0 \oplus x'_1 \oplus x'_2 \oplus x'_3 & x_1 = x'_4 \oplus x'_5 \oplus x'_6 \oplus x'_7 \\
y_0 = y'_0 \oplus y'_1 \oplus y'_2 \oplus y'_3 & y_1 = y'_4 \oplus y'_5 \oplus y'_6 \oplus y'_7 \\
z_0 = z'_0 \oplus z'_1 \oplus z'_2 \oplus z'_3 & z_1 = z'_4 \oplus z'_5 \oplus z'_6 \oplus z'_7
\end{array}
$$