The SPEEDY Family of Block Ciphers

Engineering an Ultra Low-Latency Cipher from Gate Level for Secure Processor Architectures

Gregor Leander[1], Thorben Moos[1], Amir Moradi[1] and Shahram Rasoolzadeh*[2]

[1] Ruhr University Bochum, Horst Görtz Institute for IT Security, Bochum, Germany
firstname.lastname@rub.de
[2] Radboud University, Nijmegen, The Netherlands
firstname.lastname@ru.nl

Abstract. We introduce SPEEDY, a family of ultra low-latency block ciphers. We mix engineering expertise into each step of the cipher's design process in order to create a secure encryption primitive with an extremely low latency in CMOS hardware. The centerpiece of our constructions is a high-speed 6-bit substitution box whose coordinate functions are realized as two-level NAND trees. In contrast to other low-latency block ciphers such as PRINCE, PRINCEv2, MANTIS and QARMA, we neither constrain ourselves by demanding decryption at low overhead, nor by requiring a super low area or energy. This freedom together with our gate- and transistor-level considerations allows us to create an ultra low-latency cipher which outperforms all known solutions in single-cycle encryption speed. Our main result, SPEEDY-6-192, is a 6-round 192-bit block and 192-bit key cipher which can be executed faster in hardware than any other known encryption primitive (including Gimli in Even-Mansour scheme and the Orthros pseudorandom function) and offers 128-bit security. One round more, i.e., SPEEDY-7-192, provides full 192-bit security. SPEEDY primarily targets hardware security solutions embedded in high-end CPUs, where area and energy restrictions are secondary while high performance is the number one priority.

Keywords: Low-Latency Cryptography, High-Speed Encryption, Block Cipher

1 Introduction

In this paper we revisit the following fundamental problem: How do we design a secure encryption algorithm whose hardware implementation is fast? Specifically, we care about the entire latency of the hardware circuit from the point where the inputs are provided to the point where the final outputs are ready and stable, i.e., the latency of a fully-unrolled hardware implementation entirely made from combinatorial logic. Previous approaches, which led to the design of established low-latency constructions like PRINCE [BCG+12], PRINCEv2 [BEK+20], MANTIS [BJK+16] and QARMA [Ava17], considered a low number of rounds and, to some extent, a small gate depth as design criteria. While both are obviously important factors to achieve a low latency, there are further aspects which have been ignored at the design level in the past – first and foremost the latency characteristics of the underlying hardware. At first sight it may appear to be of limited interest to tailor a cryptographic primitive towards one specific device technology due to the potential loss of generality. However, in the hardware world there has been only one de-facto standard for integrated circuit fabrication since the 1980s, namely Complementary Metal–Oxide–Semiconductor

*Part of this work was accomplished when S. Rasoolzadeh was at Ruhr University Bochum.

(CMOS) technology. The construction of CMOS logic gates, i.e., the arrangement of p- and n-channel MOSFETs (Metal–Oxide–Semiconductor Field-Effect Transistors) to create a certain functionality, has remained largely unchanged since its original proposal in 1963. In other words, CMOS logic gates – the essential building blocks for the vast majority of our computing technology today – have not experienced any fundamental redesign in almost 6 decades. Merely their size has seen a progressive decrease according to Moore's famous law [Moo65].

Notably, there are some operations which can be constructed more naturally from complementary logic. In particular, complementary gates in silicon hardware are naturally inverting and non-inverting Boolean functions cannot be realized in a single stage (i.e., they require more than one pull-up and pull-down network) [RCN04]. Among the naturally inverting logic gates some can be realized using only the minimum (lower bound) of $2n$ transistors, where n is the number of inputs the gate receives. These $2n$ transistors are then arranged in the classical layout of one pull-up network, built from p-channel MOSFETs (PMOS), and one pull-down network, built from n-channel MOSFETs (NMOS). The simple Boolean functions NAND, NOR and INV/NOT are constructed this way, but also the compound or complex logic gates AND-OR-INV (AOI) and OR-AND-INV (OAI). We argue that logic cells with these properties are immensely beneficial for low-latency constructions as they produce outputs much faster than their counterparts, independent of the particular specifications or the minimum feature size of the fabrication process.

When diving deeper into the physical characteristics of hardware circuits built from silicon, it is possible to make even further distinctions. In particular, we point out that cell layouts which require PMOS transistors to be connected in series (stacked) suffer from the lower mobility of PMOS compared to NMOS transistors more significantly. In consequence, a noticeable negative impact on the latency of such gates can be observed and larger transistor widths are required to partially offset this performance loss at the price of an increased area [RCN04]. Among the previously listed cells, only NAND and INV/NOT gates do not classically require PMOS transistors to be stacked. NOR gates with more than two inputs suffer most severely from the mobility mismatch due to the larger PMOS stacks. To clarify the impact of such observations on the performance of gates in common standard cell libraries, we present latency figures for individual logic gates exemplarily for NanGate 45 nm and 15 nm Open Cell Libraries (OCLs) in Section 2.

All gate- and transistor-level considerations described above are universally applicable to CMOS standard cells, independent of the particular foundry, manufacturing process and minimum feature size. Hence, it makes sense to take such characteristics into account when attempting to implement a certain function, like an encryption algorithm, as a hardware circuit with minimum latency. When revisiting previous latency-driven constructions in cryptography, it is clear that such low-level observations have not been considered in the past. We provide first contributions towards hardware-aware low-latency design and construct a family of ultra low-latency block ciphers based on the underlying principles.

1.1 Motivation

Approaches to secure the internals of modern Central Processing Units (CPUs) have received significant attention in the last few years as microarchitectural attacks, notably Meltdown [LSG+18] and Spectre [KHF+19], revealed serious shortcomings in the security architectures of widely deployed high-end processors. Hardware-based mitigations for such attacks are proposed "en masse". Many of them call for a higher level of encrypted communication *inside* of CPUs as well as between CPUs and their surrounding hardware components. Among the former are proposals for secure caches such as ScatterCache [WUG+19] and CEASER [Qur18]. Both of them are compared to a number of further cache architectures in [DXS19]. To implement new features of this kind in the next generations of mainstream processors without causing a large performance penalty, high-speed encryption primitives are among the most important building blocks.

Secure caches are only one example of security applications in CPU environments that require high-speed encryption. Dedicated hardware instructions, memory encryption, pointer authentication (as renownedly implemented using QARMA in ARM processors) and similar hardware-assisted mechanisms against software exploitation fall into this category as well. We expect to see a lot more of such features implemented in future generations of secure processor architectures, especially when more highly-optimized cryptographic primitives become available. SPEEDY is meant as a general purpose high-speed encryption primitive for all these applications and not limited or tailored to a subset of them.
Most low-latency ciphers published in the literature so far, such as PRINCE [BCG+12], PRINCEv2 [BEK+20], MANTIS [BJK+16] and QARMA [Ava17], try to meet tight area and energy requirements in addition to low latency. These properties make them particularly suitable for highly-constrained microcontrollers in the Internet of Things (IoT). However, keeping the primitives suited for battery-powered devices requires sacrifices with respect to maximum performance. High-end CPUs do not impose the same kind of restrictions on area and energy, yet they require even higher performance in terms of latency and throughput. SPEEDY is able to outperform the state of the art by focusing on maximum encryption speed and high security only.

1.2 Related Work

Designing cryptographic primitives with minimum execution time in hardware is still a young and emergent research discipline. At CHES 2012 the authors of [KNR12] delivered first results in that area by comparing the latency properties of multiple (lightweight) block ciphers. It was concluded that, among other factors, the use of cryptographically-strong 4-bit (or even 3-bit) S-boxes should be favored over larger substitutions and that a low number of rounds should be maintained even at the price of a heavier linear layer when designing a low-latency primitive. These demands were immediately met by the first dedicated low-latency block cipher called PRINCE which has been presented at ASIACRYPT 2012. PRINCE is a 64-bit block cipher with a 128-bit key and 12 cipher rounds which features an innovative reflection property that allows to encrypt and decrypt data with essentially the same circuit. Recently, an updated version called PRINCEv2 has been proposed which claims to increase the security level of PRINCE by making small modifications to the key schedule and the middle rounds [BEK+20]. This work also provides a comparison of multiple low-latency block ciphers which confirms that PRINCE and PRINCEv2 are still the fastest such primitives in public literature [BEK+20]. The comparison also includes the tweakable block ciphers MANTIS [BJK+16] and QARMA [Ava17] as well as the low-energy block cipher Midori [BBI+15] and demonstrates that all three of them come at a latency overhead between 22 % and 42 % (considering the encryption-only variants) compared to PRINCE in open-source NanGate libraries. This result may not come as a surprise, since tweakable block ciphers such as MANTIS and QARMA are expected to require a larger circuit depth due to the additional tweak input and since Midori has not been designed with low latency being the primary design goal, although its substitution layer has been chosen particularly to offer a small delay. However, two recent works claim that cryptographic primitives aside from traditional block ciphers are able to outperform PRINCE in terms of latency. First, the high performance cross-platform permutation Gimli introduced in [BKL+17] is claimed to enable encryption with a 1.7 times smaller latency than PRINCE in [GKD20], while the low-latency pseudorandom function (PRF) Orthros introduced in [BIL+21] claims to achieve a latency about 7 % below PRINCE's. We analyze both claims in our comparison in Section 7 and conclude that the latter is consistent with our results, while the former is clearly not. Orthros is able to achieve a lower latency than PRINCE by computing the sum of two keyed permutations [BIL+21] which makes the resulting primitive non-invertible (in contrast to block ciphers like SPEEDY).
Apart from the full cryptographic primitives discussed above, there are also some works

focusing on particular cryptographic building blocks only. For instance, in [LSL+19] it is shown how to construct involutory low-latency Maximal Distance Separable (MDS) matrices. The authors of [BFP19] present techniques for finding small low-depth circuits for cryptographic functions. In [BMD+20] the main goal is to construct S-boxes whose masked variants (i.e., their side-channel protected versions) have a low latency in hardware which conceptually requires a low AND depth and AND gate complexity. Low-latency hardware masking in general, used to protect cryptographic primitives against side-channel attacks, has received significant attention in the last few years, as demonstrated in [MS16, GIB18, ABP+18, BKN19, SBHM20]. However, this field is not directly related to the development of low-latency symmetric primitives in general, as the requirements are vastly different and sometimes even direct opposites.[1]

1.3 Our Contribution

We introduce SPEEDY, a family of ultra low-latency block ciphers dedicated to semi-custom, i.e., standard-cell-based, integrated circuit design. In order to tailor this cryptographic primitive towards maximum execution speed in hardware we first analyze which type of logic gates and circuit topologies are particularly suited for ultra low-latency encryption. Our considerations in this regard are novel and have, to the best of our knowledge, not been applied in previous designs of symmetric cryptographic primitives.
SPEEDY can be instantiated with different block and key sizes and varying numbers of rounds. However, due to our S-box width of 6 bits and our main target application of 64-bit high-end CPUs we decided to use the least common multiple of 6 and 64, namely 192 as the default block and key size and call this instance SPEEDY-r-192. We claim that SPEEDY-r-192 achieves 128-bit security when iterated over $\mathtt{r} = 6$ rounds and full 192-bit security when iterated over $\mathtt{r} = 7$ rounds, while the $\mathtt{r} = 5$ round variant already provides a decent security level that is sufficient for many practical applications. Our extensive evaluation of hardware implementations in 6 different standard cell libraries shows that both SPEEDY-5-192 and SPEEDY-6-192 achieve a lower latency in hardware than any other known encryption primitive, while SPEEDY-7-192 is only marginally slower than PRINCE. Considering the provided security levels this is a significant improvement over the state of the art in the area of (ultra) low-latency cryptography.

2 Background

In this section we revisit the necessary concepts which build the foundation for SPEEDY and analyze the primary traits that make certain CMOS standard cells and circuit topologies particularly useful for high-speed cryptography.

2.1 Natural CMOS Gates (NCGs)

A static CMOS gate is constructed by combining a pull-up with a pull-down network. The pull-up network, as the name suggests, is responsible for pulling the output of the gate up to VDD whenever the Boolean function should result in a logical '1'. The pull-down network, analogously, is responsible for pulling the output down to GND whenever the Boolean function should output a logical '0'. The networks are built in a mutually exclusive manner such that only one of them is conductive for each combination of input signals [RCN04]. While the pull-up networks are exclusively built from PMOS devices, the pull-down networks are built from NMOS devices. PMOS devices can be understood

[1] In regular cryptographic S-boxes, non-linear gates such as AND and NAND are beneficial for area and latency over linear gates like XOR and XNOR for instance. In masked S-boxes on the other hand, linear operations are optimal and non-linear gates are the primary cost factor [BMD+20].

as switches that conduct current between their drain and source terminals whenever their gate voltage is low, NMOS devices conduct current between the terminals whenever their gate voltage is high. For the opposite gate voltages the transistors are in a high-resistance state. The assignment of PMOS transistors to pull-up networks and NMOS to pull-down networks originates from the fact that PMOS devices cannot produce so-called *strong zeros*, while NMOS devices cannot produce *strong ones* [RCN04]. In consequence, static CMOS gates with a single stage are naturally inverting by design. Non-inverting Boolean functions require at least two stages of pull-up and pull-down networks. Thus, as already discussed in Section 1, certain logic functions are a more natural fit for technologies that are based on complementary metal–oxide–semiconductor logic. Inverting Boolean functions include for instance the common logic gates INV/NOT, NAND, NOR, XNOR, AOI and OAI. Most of them (all except XNOR) can be realized as static gates by using only the lower bound of $2n$ devices, namely n PMOS and n NMOS transistors. We call all inverting logic gates which require only one stage and $2n$ transistors for their implementation *Natural CMOS Gates (NCGs)*. All NCGs commonly found in standard cell libraries with $1 \leq n \leq 4$ inputs are depicted in Figure 1. Such logic cells are not only interesting from a hardware design perspective because they require a lower number of transistors and therefore have a smaller area footprint, they are also faster than their opposition and therefore beneficial for low-latency constructions.

2.2 Latency of CMOS Logic Gates

The time that a physical instance of a logic gate requires to respond to a change in its input signals by updating its output signal is called the delay or the latency of a cell. Considering CMOS hardware, the latency of a physical instance of a logic cell depends on a number of factors. Besides environmental influences like the temperature and the supply voltage, also the transition time of the input signals and the capacitance that needs to be driven at its output play a significant role. In this subsection, however, we want to compare the base latencies of static CMOS gates when all outside factors are equal. Tables 1 and 2 list the latencies of common logic gates in two open-source standard cell libraries, namely NanGate 45 nm and 15 nm Open Cell Libraries (OCLs), respectively. The latency values are given in picoseconds and have been obtained by analyzing a netlist containing only the individual logic gate enclosed between standard D-flip-flop cells for typical operating conditions (25 °C, nominal voltage) with the Electronic Design Automation (EDA) software *Synopsys Design Compiler Version O-2018.06-SP4* using Composite Current Source (CCS) models of the standard cells. Please note that for simplicity only the logic gates with the minimum drive strength (denoted by the suffix `"_X1"` in NanGate libraries) are shown here. However, the following arguments and considerations also apply to the higher drive strength variants. As expected, the natural CMOS gates, defined in the previous subsection, produce their outputs significantly faster than the competition. Interestingly, though, some significant differences between analogous natural gates such as NAND and NOR can be observed. In NanGate 45 nm technology for example, the `NAND4_X1` cell is more than twice as fast as the `NOR4_X1` cell. This is due to the different physical behavior of p-type and n-type MOSFETs realized in silicon as semiconductor material. In n-type MOSFETs the majority carriers are electrons which are negatively charged. In p-type MOSFETs on the other hand, the majority carriers are positively charged holes [RCN04]. Holes are less mobile than electrons, which means they move slower. Therefore, simply speaking, PMOS transistors operate slower than NMOS transistors of the same size. This situation is even amplified when connecting PMOS devices in series (stacking) and leads to a significant performance degradation and an increased area demand due to the larger widths required to partially offset the performance penalty and achieve balanced rise and fall times. Classic CMOS NOR gates require stacks of n PMOS transistors and are therefore among the logic functions which suffer the most from the lower mobility of holes as majority carriers. Since

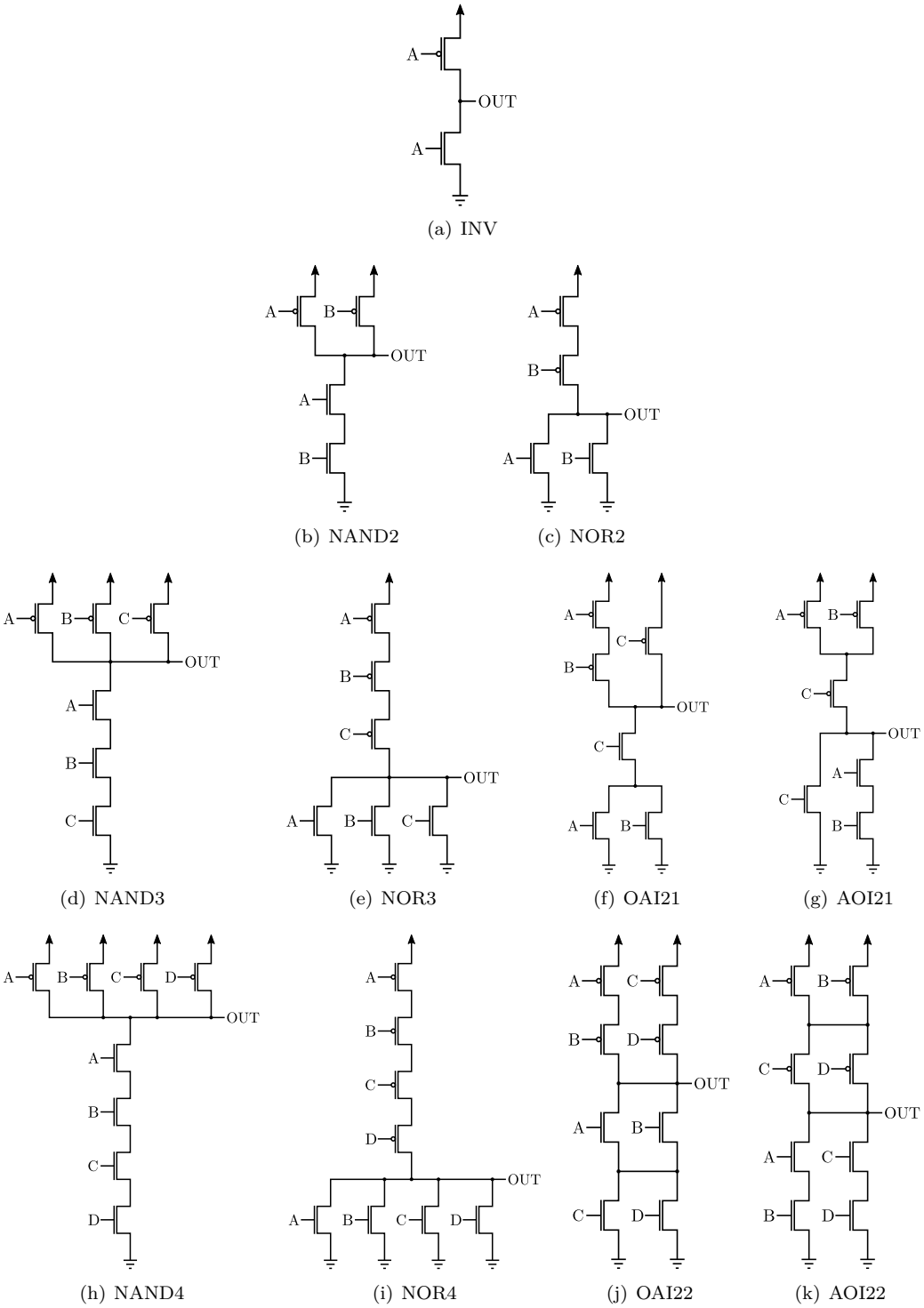


Figure 1: *Natural CMOS Gates (NCGs)*: Inverting logic cells realizable in only one stage of $2n$ MOSFETs as static CMOS gates, where n is the number of inputs.

both types of complex gates, AOI and OAI, require stacked PMOS transistors in their layouts as well, we can make similar arguments here, although the effect is less striking

Table 1: Fan-In, Latency, Fan-In-to-Latency-Ratio and Linearity of logic gates in NanGate 45nm Open Cell Library (OCL) for typical operating conditions.

Cell Name	Fan-In	Latency [ps]	FLR	Linearity
INV_X1	1	22.047900	0.045356	2
BUF_X1	1	33.556521	0.029800	2
AND2_X1	2	40.170699	0.049788	2
NAND2_X1	2	27.885556	0.071722	2
NOR2_X1	2	40.649809	0.049201	2
OR2_X1	2	56.413554	0.035452	2
XNOR2_X1	2	57.604454	0.034720	4
XOR2_X1	2	73.018849	0.027390	4
AND3_X1	3	51.869132	0.057838	6
AOI21_X1	3	51.618919	0.058118	6
MUX2_X1	3	75.174913	0.039907	4
NAND3_X1	3	34.766912	0.086289	6
NOR3_X1	3	61.542571	0.048747	6
OAI21_X1	3	32.650799	0.091881	6
OR3_X1	3	85.839920	0.034949	6
AND4_X1	4	65.491892	0.061076	14
AOI22_X1	4	57.255469	0.069862	6
NAND4_X1	4	44.487149	0.089914	14
NOR4_X1	4	91.312885	0.043805	14
OAI22_X1	4	54.596245	0.073265	6
OR4_X1	4	118.592046	0.033729	14

since the stacks are smaller. OAI gates are typically faster than AOI gates in common standard cell libraries since the internal capacitances in the pull-up networks of AOI gates are larger. NAND and INV/NOT gates are the only NCGs that do not require PMOS stacks in their classical layout. As a result, INV/NOT and NAND2 gates are almost exclusively the fastest CMOS gates for $n = 1$ and $n = 2$ in any CMOS gate library. For $n = 3$ and $n = 4$ the situation depends on the exact sizing of the transistors chosen by the cell designer for each particular gate. This choice determines the trade-off between area and latency of the logic cells. Typically, either NAND3 and NAND4 or OAI21 and OAI22 are the fastest gates for $n = 3$ and $n = 4$, respectively. In NanGate 45 nm technology OAI21 ($n = 3$) and NAND4 ($n = 4$) are the fastest cells for their respective number of inputs while in 15 nm technology NAND3 ($n = 3$) and OAI22 ($n = 4$) cells are the fastest, as apparent in Tables 1 and 2.

2.2.1 Suitability for High-Speed Encryption

There are several factors to be considered when determining which cells in a standard gate library are most suitable for low-latency encryption. Building a low-latency encryption primitive in hardware is essentially the task of creating a circuit that, as quickly as possible, establishes an, as highly as possible, non-linear relationship between the plaintext and, as many as possible, independent key bits. Of course, this is an extreme oversimplification of the large number of requirements that symmetric cryptographic primitives need to fulfill in order parry all known attacks. Yet, when following this simplified idea, the design process for an ultra low-latency cipher should start at the gate level. In particular, we are interested in logic gates that are capable of establishing a Boolean relationship between as many inputs as possible in a short period of time. In that regard, we introduce a new metric, which we call the Fan-in-to-Latency Ratio (FLR). Essentially, we divide the fan-in

Table 2: Fan-In, Latency, Fan-In-to-Latency-Ratio and Linearity of logic gates in NanGate 15nm Open Cell Library (OCL) for typical operating conditions.

Cell Name	Fan-In	Latency [ps]	FLR	Linearity
INV_X1	1	1.580082	0.632879	2
BUF_X1	1	3.068201	0.325924	2
AND2_X1	2	3.579786	0.558692	2
NAND2_X1	2	2.030621	0.984920	2
NOR2_X1	2	2.554366	0.782973	2
OR2_X1	2	3.643867	0.548867	2
XNOR2_X1	2	6.788322	0.294624	4
XOR2_X1	2	5.268465	0.379617	4
AND3_X1	3	5.496015	0.545850	6
AOI21_X1	3	3.394032	0.883904	6
MUX2_X1	3	6.133133	0.489146	4
NAND3_X1	3	2.360978	1.270660	6
NOR3_X1	3	3.787567	0.792065	6
OAI21_X1	3	2.830147	1.060016	6
OR3_X1	3	5.862194	0.511754	6
AND4_X1	4	7.125210	0.561387	14
AOI22_X1	4	4.070343	0.982718	6
NAND4_X1	4	4.659015	0.858551	14
NOR4_X1	4	5.250172	0.761880	14
OAI22_X1	4	3.775570	1.059443	6
OR4_X1	4	7.682688	0.520651	14

n of each gate (i.e., the number of inputs it receives) by its latency. Let $f : \mathbb{F}_2^n \to \mathbb{F}_2$ be the Boolean function of a logic gate and n the number of inputs it receives (i.e., the fan-in), then the *Fan-in-to-Latency Ratio (FLR)* of f can be expressed as Equation 1.

$$\mathrm{FLR}(f) = \frac{n}{\mathrm{latency}(f)} \tag{1}$$

By calculating the FLR for each logic gate in a standard cell library one can rank the gates by their suitability for ultra low-latency encryption. Tables 1 and 2 list the FLR scores for all standard logic gates with n inputs for $1 \leq n \leq 4$. The FLR score reflects the ability of a logic gate to rapidly evaluate a Boolean function on multiple inputs. Hence, the higher the value in the FLR-column for a logic gate, the higher is its potential to be suitable for ultra low-latency encryption. NAND and OAI gates are among the logic cells with the highest FLR scores, while XOR and XNOR gates are among the worst performers. Thus, despite the importance of XOR (and XNOR) gates in symmetric cryptography (mostly for key addition and strong linear layers) it is prudent to limit their occurrence to a minimum. Obviously, the kind of Boolean logic function that is evaluated plays a significant role in determining its suitability for high-speed encryption as well. In that regard, a further important aspect is the linearity of a function. $\mathrm{Lin}(f)$ denotes the linearity of the Boolean function f, defined by Equation 2, where $\widehat{f} : \mathbb{F}_2^n \to \mathbb{Z}$ is the Fourier transform of f given by Equation 3.

$$\mathrm{Lin}(f) := \max_{\alpha \in \mathbb{F}_2^n} \left| \widehat{f}(\alpha) \right| \tag{2}$$

$$\widehat{f}(\alpha) = \sum_{x \in \mathbb{F}_2^n} (-1)^{f(x) + \langle \alpha, x \rangle} \tag{3}$$

Tables 1 and 2 provide the linearity of all listed logic gates. The linearity of a Boolean function $f : \mathbb{F}_2^n \to \mathbb{F}_2$ is lower bounded by $2^{\frac{n}{2}}$ and upper bounded by 2^n. Whenever

$\mathrm{Lin}(f) = 2^n$, f is an affine function, i.e., Equation 4 holds with $\alpha \in \mathbb{F}_2^n, c \in \mathbb{F}_2$.

$$f(x) = \langle \alpha, x \rangle + c \tag{4}$$

In our tables, the logic functions INV/NOT, BUF, XOR, XNOR have maximum linearity (2^n) and can be expressed as constant or affine functions, while the logic gates AND2, NAND2, NOR2 and OR2 reach the lower bound for the linearity of $2^{\frac{n}{2}}$.
While both, linear and non-linear functions, are useful for the construction of secure encryption algorithms, they are typically used in different layers or round operations. The non-linear layer in block cipher design is typically the substitution layer while all other operations tend to be linear. Often the substitution boxes, in short S-boxes, are among the most resource consuming elements in terms of area, energy and latency. Therefore, it is particularly interesting to optimize this building block towards the desired design goal when developing and implementing a cipher. In that regard, non-linear gates with a high FLR score, like NAND and OAI, are the prime candidates for building strong and fast S-boxes.

2.3 Latency of Logic Circuits

It is insufficient to consider only the latencies of individual logic elements in order to determine the resulting total latency of a combinatorial circuit or path. When connecting logic gates to logic circuits, the individual propagation delays of the gates depend significantly on their direct electrical environment. Merely summing up the base latencies of the gates in a path (e.g., the values given in Tables 1 and 2) may give a *very* incorrect idea about the path's total latency. Despite the fact that some obvious correlation between these quantities can be observed, the gate depth of a path is not always directly proportional to its latency. Therefore, it is important to also consider adequate circuit topologies which minimize the latency of combinatorial circuits when designing a low-latency cipher. In this regard, we first want to dispel two common myths about the latency of CMOS circuits:

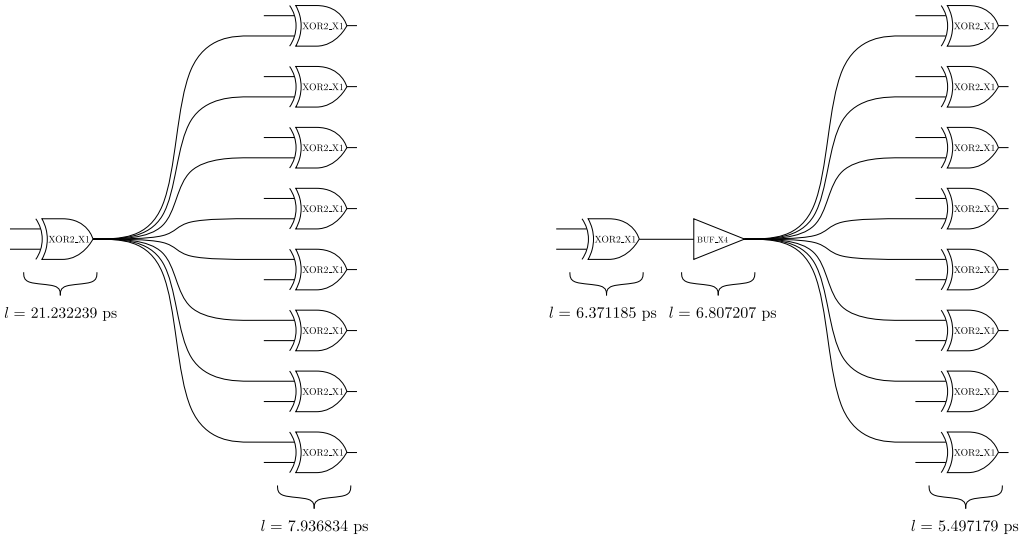
- *Myth 1:* Each CMOS standard cell has a fixed delay and each instantiation of the same exact standard cell adds (approximately) the same latency to a path.

  Truth: This is false. The propagation delay of a CMOS cell is always a function of the transition time of its input signals, which is influenced by the drive strength of preceding cells and the capacitance of the nets they need to drive, as well as the capacitive load that the CMOS cell itself needs to drive at its output. The variations of the delay of a cell instance depending on its electrical environment can easily be in the range of 200-300%. Therefore, it is not uncommon that two instances of the same cell in different positions of a logic circuit have delays associated with them (e.g., in a timing report) that differ by a factor of 3 or 4.

- *Myth 2:* Adding a gate to a path of a circuit and not making any other changes to the path will always increase the path's latency.

  Truth: This is also false. Often, adding a well-placed buffer or inverter (where logically applicable) to a path in order to charge a significant capacitive load faster can decrease the overall latency of the path. Hence, the mere gate depth is not always indicative of the latency of a circuit. Generally, the topology of a circuit, primarily the fan-out of the logic cells, is similarly important as the number and type of gates in its critical path when determining the maximum latency.

In the following we provide an example which demonstrates the incorrectness of the two myths. We consider a simple circuit in Figure 2(a) where the output signal of a single XOR logic gate in NanGate 15 nm technology (`XOR2_X1`) is the input to 8 further XOR



(a) without buffering (b) with buffering

Figure 2: Impact on the latency of the circuit in NanGate 15 nm technology when buffering the high fan-out net. Total latency is 29.169073 ps without the buffer (left) and 18.675571 ps with the buffer (right), despite the larger gate depth on the right.

cells. The respective maximum latencies for each of the two circuit stages are denoted below the gates in Figure 2. While the base latency of a simple XOR logic gate in this technology is 5.268465 ps according to Table 2, it is obvious that the actual latencies of the gates in this circuit are significantly larger. The first XOR gate in particular which feeds the other 8 gates requires a latency which is more than 4 times as large as its base latency due to the significant capacitive load it needs to drive. The XOR gates in the second stage do not drive any large loads but their latency is increased because their input signals have a large transition time. It is noteworthy that this is a synthesis result, which means that the actual capacitances and resistances of the routing (i.e., wiring) are not even considered yet. After placing and routing this circuit in a chip design the latencies would likely be even larger. Figure 2(b) shows a circuit with the same logic functionality and the same 9 total XOR gates, but here the output of the first stage XOR is buffered by a drive strength buffer (`BUF_X4`). Although this change increases the gate depth of the circuit, it decreases its overall latency. The first stage XOR now only needs to drive a small load and the last stage XORs are driven by input signals with a short transition time. As a result, the buffered circuit has a total latency of 18.675571 ps (Fig. 2(b)) while the circuit without a buffer has a total latency of 29.169073 ps (Fig. 2(a)). Hence, the buffered circuit is more than 35% faster. Please note that the NanGate 15 nm library does not provide XOR gates with a higher drive strength, thus up-sizing the first stage XOR itself is not an option here and buffering the high fan-out net is inevitable when the latency should be reduced. Of course, this is done automatically by the synthesis tool. Our point here is simply that, regardless of how the large fan-out is addressed by the tool or the designer, e.g., up-sizing the gate or inserting a buffer, it assuredly causes an increased latency compared to a circuit with the same depth and the same gates in both levels, but with smaller fan-outs. Thus, we conclude that dedicated low-latency circuits should use topologies where the fan-outs of the logic gates are as small as possible (e.g. tree-based).

2.3.1 Finding Circuits with Minimum Latency

We would like to caution against the common perception that professional synthesis tools can readily be used to find and generate a netlist with minimum achievable latency for a simple Boolean function like an S-box coordinate function. First of all, the complexity of checking any possible circuit representation composed of a finite (but usually large) set of standard cells for a Boolean function is often remarkably high and market-leading EDA tools are built for time efficiency (especially the synthesis routines). Furthermore, the proprietary synthesis algorithms may not be sufficiently configurable to consider latency as the only or primary design goal. The tools may rather take area and energy into account as well and not consider latency optimizations that come at a harsh penalty for the other two optimization goals. In our own experience, the thresholds for such decisions cannot be adjusted sufficiently by the designer. Thus, we have found that constructing optimal building blocks for ultra low-latency cryptography needs to be done from scratch (by hand or via heuristics) instead of analyzing many different variants with a synthesis tool and selecting the ones that delivered the best performance. In our evaluations, the synthesis algorithms usually produced the best results with respect to low latency, when the underlying gate structure was already given and only incremental performance optimizations were required.

3 Ultra Low-Latency 6-bit S-box

In this section, we describe the technique we have used to build an ultra low-latency S-box from gate level. In order to design an S-box which is extremely fast in CMOS hardware while at the same time provides good cryptographic properties, we used the following criteria:

- Ultra low-latency: As explained in Subsection 2.2, NAND and OAI gates are among the best-suited logic gates for low-latency S-box design. Thus, we search for S-boxes that can be realized with as few as possible levels of only NAND and OAI gates. Furthermore, as discussed in Subsection 2.3, we try to make sure that in as many stages as possible the logic gates have a minimum fan-out.
- Bijective mapping with fully-dependent outputs: Since we aim for an SPN cipher, we need the S-box to be a bijective mapping. Moreover, we restrict the search to the S-boxes with fully-dependent outputs. In more detail, this means that all input bits are involved in the computation of each output bit.
- Small linearity and uniformity: To provide strong resistance against differential and linear attacks, we are only interested in S-boxes with small *uniformity* u and *linearity* l defined as

$$u = \mathrm{Uni}(S) := \max_{\substack{\alpha,\beta\in\mathbb{F}_2^n \\ \alpha\neq 0}} \left|\{x \in \mathbb{F}_2^n | S(x) \oplus S(x \oplus \alpha) = \beta\}\right|,$$

$$l = \mathrm{Lin}(S) := \max_{\substack{\alpha,\beta\in\mathbb{F}_2^n \\ \beta\neq 0}} \left| \sum_{x\in\mathbb{F}_2^n} (-1)^{\langle\alpha,x\rangle\oplus\langle\beta,S(x)\rangle} \right|.$$

By definition, the latency of a vectorial Boolean function, e.g., an S-box, is the maximum of the latencies of its coordinate Boolean functions. Besides, to have a bijective fully-dependent S-box with a small linearity, all of its coordinate functions must be balanced, fully-dependent and have a small linearity. Hence, our strategy was to first find low-latency Boolean functions and in a second step try to combine those into an S-box.
It is noteworthy that the S-boxes within the same class of extended bit-permutation

equivalence have roughly the same latency cost (with a small margin of tolerance). Moreover, those functions will have the same uniformity and linearity. We recall from [LP07] that two n-bit to m-bit vectorial Boolean functions F and G of the form $\mathbb{F}_2^n \mapsto \mathbb{F}_2^m$ are called extended bit-permutation equivalent, if there exist $a \in \mathbb{F}_2^n$, $b \in \mathbb{F}_2^m$, P_{in} a bit permutation function of n bits and P_{out} a bit permutation function of m bits such that

$$G(x) = P_{out} \circ F \circ P_{in}(x \oplus a) \oplus b \quad \forall x \in \mathbb{F}_2^n \,.$$

Therefore, it is sufficient to consider S-boxes only up to this equivalence.

3.1 Suitable Boolean Functions

To achieve a minimal latency, we searched for coordinate functions that can be realized in only two levels of NAND and OAI gates, or more specifically NAND2, NAND3, NAND4, OAI21 and OAI22 gates, while the larger and slower NAND4 and OAI22 gates should only be used in one of both levels. Additionally the first stage of NAND and OAI gates should have a fan-out of 1 for each gate. With this restriction, we are able to find Boolean functions with an extremely low latency in CMOS hardware.
We empirically found that Boolean functions based on NAND gates exclusively achieve the best cryptographic properties and latencies with only two levels at a higher quantity; therefore, in the following we limit ourselves to S-boxes which are possible to be built only from NAND gates. However, using the same process described in the following we have created S-boxes based on OAI gates exclusively (functions based on a mix between NAND and OAI have shown to be less promising) and compare them to the NAND-based boxes at the end of this section.
By considering all the possibilities for the inputs of the NAND gates at the first level, we aim at building all the n-bit Boolean functions $f(x_0, \ldots, x_{n-1})$; i.e., for each input for NAND gates we test $2n$ possible inputs: either x_i or its inverted value $\neg x_i$ with $0 \leq i < n$. We then filter the Boolean functions with respect to the aforementioned criteria, that is balancedness and low-linearity. Please note that selecting the inverted inputs requires additional inverter gates before the first stage of NAND gates. Yet, since each of the S-box inputs feeds multiple coordinate Boolean functions at the same time it is prudent to instantiate buffers to drive those nets anyway and an inverter can serve the same purpose. Following this argument, the inverted inputs do not cause any significant extra cost.
The first step is to find all the Boolean functions $f : \mathbb{F}_2^n \mapsto \mathbb{F}_2$ which are: 1) possible to be built by using two levels of NAND gates as explained previously, 2) balanced, 3) fully-dependent on all the input bits, and 4) with linearity at most l. It is important to mention that the order of checking these features is quite important for reducing the computational cost.
We save all those Boolean functions in a set, named $\mathcal{F}$. Note that if there is a function $f \in \mathcal{F}$, then all of its extended bit-permutation equivalent functions such as $g(\cdot) = f \circ P(\cdot \oplus a) \oplus b$ with $a \in \mathbb{F}_2^n$, $b \in \mathbb{F}_2$ and P a bit permutation function of n bits, are included in $\mathcal{F}$. Next, we reduce the Boolean functions within $\mathcal{F}$ by the extended bit-permutation equivalence, and only keep one representative of each equivalence class in another set $\mathcal{F}^*$. Note that if there are N_f^* Boolean functions in $\mathcal{F}^*$, then there are about $N_f = N_f^* \cdot (n! \cdot 2^{n+1})$ functions in $\mathcal{F}$. This reduction corresponds to the $n!$ permutations of the input bits, the 2^n constants we can add to the input and the single bit we can add to the output.

3.2 Building Sboxes

To find all the bijective S-boxes $S = (f_0, \ldots, f_{n-1})$ such that each coordinate function is in $\mathcal{F}$, we can simply choose n of those N_f functions and then check for the necessary criteria, but this requires about $(N_f)^n$ steps of checking all the criteria which for $n > 4$

is a large computation cost. The two main options to reduce this cost is (i) considering permutation equivalence and (ii) to select the coordinate function step-by-step and filter after each additional choice.
Since it is sufficient to find the bijective S-boxes up-to the extended bit-permutation equivalence, we can restrict the first coordinate function f_0 to be chosen from $\mathcal{F}^*$ that is due to the freedom on choosing the constant and the bit-permutation in the input of the S-box. Besides, for all the other coordinate functions $f_1, \ldots, f_{n-1}$, we can fix an input's output to a constant, e.g., $f_i(0) = 0$ and this is because of the freedom in the output constant of the S-box. Note that since f_0 is chosen from $\mathcal{F}^*$ and it is a representative function, we already considered that $f_0(0) = 0$. Moreover, since we are still left with the freedom on the output bit-permutation of the S-box, we can fix the order of the coordinate functions of the S-box. In other words, if we consider that the elements of $\mathcal{F}$ are indexed, then we can fix the index of f_1 to be smaller than the index of f_2 and both are smaller than the index of f_3 and so on. This way, we reduce the number of choices to build an S-box to about $N_f^n/(n! \cdot 2^n)^2 \approx (N_f^*)^n \cdot (n!)^{n-2} \cdot 2^{n^2-n}$. In case of $n = 5$, this number is about $(N_f^*)^5 \cdot 2^{41}$ which is still not feasible to search.
The other main technique to reduce the computation cost of this search is that instead of choosing all the coordinate functions at once and then check for the criteria, we choose them one by one and in each step of choosing a coordinate function, we check for the probable possible criteria. In more details, in step one, we choose $f_0 \in \mathcal{F}^*$, then in step 2, we choose $f_1 \in \mathcal{F}$. Before, going to step 3, we can check for balancedness and linearity of the component function $f_0 \oplus f_1$. We go to the next step, if the criteria for $f_0 \oplus f_1$ have met, otherwise, we stay in step 2 and choose another function as f_1. In step 3, after choosing $f_2 \in \mathcal{F}$, we again can check for balancedness and linearity of the component functions $f_0 \oplus f_2$, $f_1 \oplus f_2$, $f_0 \oplus f_1 \oplus f_2$. We go to step 4, if all these criteria have met. In this way, we choose all the n coordinate functions to build the S-box, and then we can check for the uniformity criterion.
This technique, together with several other low-level techniques for speeding up the search, reduces the computation cost of this search significantly. Our search algorithm is written in `C++` code and we run it on an Intel Core i7 CPU with 8 threads for about 10 days to exhaustively search all the possible 6-bit S-boxes. Finding all 5-bit S-boxes only requires about two hours.
We also constructed 7- and 8-bit S-boxes, but due to the larger linearity or uniformity value, they would not have been beneficial over the 6-bit S-box.

3.3 Results

In case of 6-bit S-boxes, the minimum linearity and the minimum uniformity of all S-boxes possible to built, is 24 and 8, respectively. For these properties, up to the extended bit-permutation equivalence, there are only two class of such S-boxes. We choose the S-box class equivalent to the one shown in Figure 3 and given in Table 3, because of the higher algebraic degree.
For the chosen S-box class, we have the freedom to choose the input/output constants a and b and also P_{in} and P_{out} bit-permutation functions. We choose the output constant b in such a way that there is no need to insert an inverter in the output of the NAND gates of the second gate level. Even though it is a tiny improvement, the input constant a is chosen in a way to minimize the latency of the whole structure.
Finally, we choose the bit-permutations in such a way that it improves the cryptographic properties of the round function for `SPEEDY` which is explained in more detail in Section 6. Note that the optimum choice of these bit-permutations can be different for round functions of different primitives. Altogether, we end up with the S-box presented in Table 3. Its corresponding implementation is depicted in Figure 3. Furthermore, the disjunctive normal form (DNF) of the S-box is presented below, which is equivalent to the representation by

Table 3: The 6-bit S-box of SPEEDY.

x_0x_1	$x_2x_3x_4x_5$															
	.0	.1	.2	.3	.4	.5	.6	.7	.8	.9	.a	.b	.c	.d	.e	.f
0.	08	00	09	03	38	10	29	13	0c	0d	04	07	30	01	20	23
1.	1a	12	18	32	3e	16	2c	36	1c	1d	14	37	34	05	24	27
2.	02	06	0b	0f	33	17	21	15	0a	1b	0e	1f	31	11	25	35
3.	22	26	2a	2e	3a	1e	28	3c	2b	3b	2f	3f	39	19	2d	3d

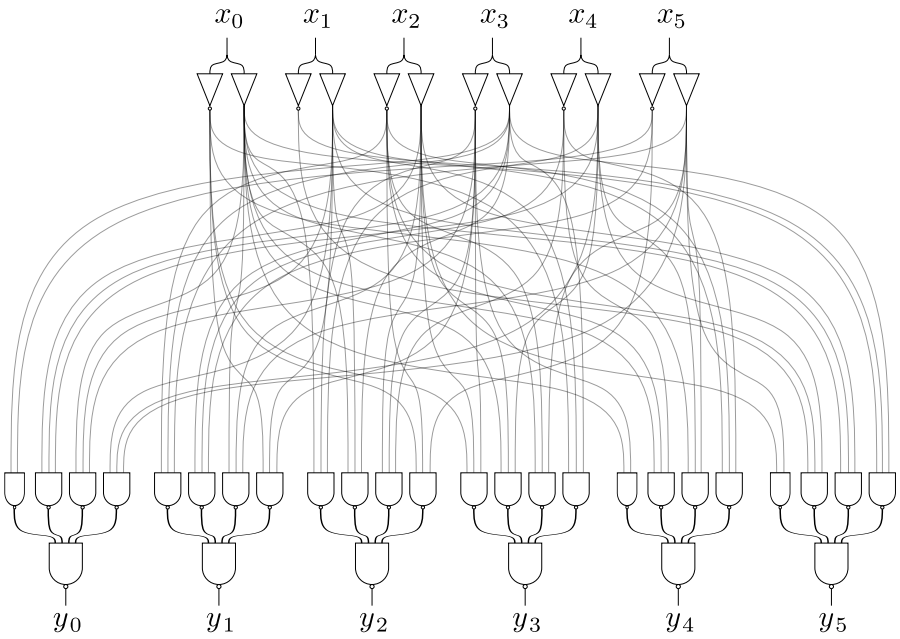


Figure 3: Implementation of the 6-bit S-box of SPEEDY based on two-level NAND trees.

the 2-level NAND gates.

$$
\begin{aligned}
y_0 &= (\ x_3 \wedge \neg x_5 \quad\ ) \vee (\ x_3 \wedge x_4 \wedge x_2\ ) \vee (\neg x_3 \wedge x_1 \wedge x_0) \vee (\ x_5 \wedge x_4 \wedge x_1\ ),\\
y_1 &= (\ x_5 \wedge x_3 \wedge \neg x_2) \vee (\neg x_5 \wedge x_3 \wedge \neg x_4) \vee (\ x_5 \wedge x_2 \wedge x_0) \vee (\neg x_3 \wedge \neg x_0 \wedge x_1\ ),\\
y_2 &= (\neg x_3 \wedge x_0 \wedge x_4\ ) \vee (\ x_3 \wedge x_0 \wedge x_1\ ) \vee (\neg x_3 \wedge \neg x_4 \wedge x_2) \vee (\neg x_0 \wedge \neg x_2 \wedge \neg x_5),\\
y_3 &= (\neg x_0 \wedge x_2 \wedge \neg x_3) \vee (\ x_0 \wedge x_2 \wedge x_4\ ) \vee (\ x_0 \wedge \neg x_2 \wedge x_5) \vee (\neg x_0 \wedge x_3 \wedge x_1\ ),\\
y_4 &= (\ x_0 \wedge \neg x_3 \quad\ ) \vee (\ x_0 \wedge \neg x_4 \wedge \neg x_2) \vee (\neg x_0 \wedge x_4 \wedge x_5) \vee (\neg x_4 \wedge \neg x_2 \wedge x_1\ ),\\
y_5 &= (\ x_2 \wedge x_5 \quad\ ) \vee (\neg x_2 \wedge \neg x_1 \wedge x_4\ ) \vee (\ x_2 \wedge x_1 \wedge x_0) \vee (\neg x_1 \wedge x_0 \wedge x_3\ ).
\end{aligned}
$$

3.4 S-box Latency Comparison

We benchmark our chosen S-box with respect to minimum latency in hardware and compare it to a number of other S-boxes from literature in Table 4. Details about the synthesis tools and process are given in Section 7. Please note that up to now only 4-bit S-boxes have been proposed for low-latency constructions in literature, namely (in alphabetical order) the Midori S-boxes [BBI+15], the Orthros S-box [BIL+21], the PRINCE S-box [BCG+12] and the QARMA S-boxes [Ava17]. Yet, in order to compare the SPEEDY S-box also to larger substitution boxes we chose the ASCON 5-bit S-box [DEMS19], the Data Encryption Standard (DES) S_1 6-to-4-bit box (as a representative of the 8 different DES S-boxes) [oST79], the Q2263 6-bit S-box [BMD+20] and the Advanced Encryption

Table 4: Latency comparison of different S-boxes with varying numbers of input bits (#ib). If not stated otherwise, each S-box is implemented as a lookup table (using with/select in VHDL).

		Minimum Latency [ns]					
		Commercial Foundry				NanGate OCL	
#ib	S-box	90 nm LP	65 nm LP	40 nm LP	28 nm HPC	45 nm	15 nm
4	`Midori Sb`$_0$	0.089098	0.070579	0.055577	0.021051	0.111156	0.010619
4	`Midori Sb`$_1$	0.132489	0.095724	0.080657	0.026898	0.119637	0.009058
4	`Orthros`	0.075344	0.051435	0.055908	0.021003	0.133932	0.008821
4	`PRINCE`	0.087938	0.066545	0.052826	0.031010	0.126588	0.010176
4	`QARMA` σ_0	0.090568	0.057602	0.051993	0.022180	0.128350	0.009409
4	`QARMA` σ_1	0.144465	0.101487	0.077186	0.031306	0.156462	0.011272
4	`QARMA` σ_2	0.100530	0.075846	0.081528	0.036485	0.154379	0.013354
5	`ASCON`	0.197794	0.151025	0.123356	0.057595	0.210599	0.019854
6	`DES S`$_1$	0.260286	0.190725	0.153514	0.069299	0.309009	0.030846
6	`OAIU8L24`	0.138926	0.111734	0.088775	0.046295	0.215628	0.017971
6	`Q2263`	0.233256	0.171537	0.157194	0.068870	0.246198	0.028648
6	min(`RU8L24`)	0.220168	0.144777	0.126819	0.060535	0.240982	0.026696
6	`SPEEDY`	0.106872	0.081330	0.065966	0.029890	0.161653	0.016124
6	`SPEEDY` *	0.096468	0.073253	0.064215	0.029470	0.138825	0.012799
6	`SPEEDY_INV`	0.207746	0.152161	0.129433	0.071523	0.278395	0.025665
8	`AES`	0.407332	0.304098	0.248914	0.130490	0.491570	0.048258

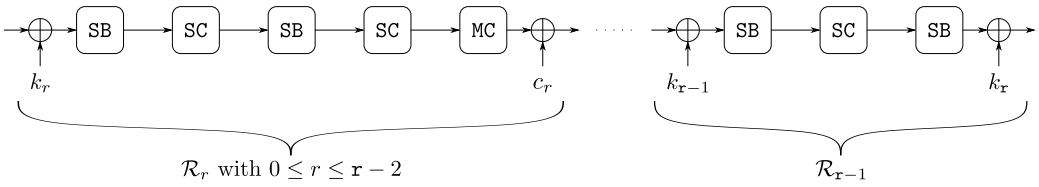
* = Optimized HDL code with direct instantiation of library cells based on Figure 3.

Standard (AES) 8-bit S-box [oST01] for the comparison. Under the abbreviation `OAIU8L24` we have listed a 6-bit S-box built from two levels of OAI22 gates with uniformity 8 and linearity 24 (same properties as the SPEEDY S-box). By min(`RU8L24`) we denote the minimum latency achieved among 10 randomly generated 6-bit S-boxes with uniformity 8 and linearity 24 (without focusing on a particularly efficient implementation). Finally, the inverse of the `SPEEDY` S-box is included. However, this inverse is not required for the `SPEEDY` encryption and therefore only relevant for the latency of its decryption. Minimizing the decryption's latency is not a focus of this work.
From the comparison it becomes clear that the `SPEEDY` S-box is impressively fast in hardware. It is much faster than any other S-box with more than 4 input bits (#ib), especially when considering the optimized version with direct instantiation of standard cells in the code based on Figure 3. Additionally, it even outperforms multiple of the 4-bit low-latency S-boxes (including `Midori Sb`$_1$, `QARMA` σ_1 and `QARMA` σ_2). This is a great result, since the `SPEEDY` S-box not only provides better diffusion in general but also offers stronger protection against linear and differential attacks than any 4-bit S-box possibly could. Thus, we are confident in our S-box choice as the centerpiece for an ultra low-latency cipher.

4 Specification of `SPEEDY`

`SPEEDY` is a family of ultra low-latency block ciphers with different block and key sizes, and varying numbers of rounds. Precisely, `SPEEDY-r-`6ℓ is an instance of this family with block and key size 6ℓ bits and it iterates over `r` rounds.
The internal state is viewed as an $\ell \times 6$ rectangle array of bits. We use the notation $x_{[i,j]}$ to denote the bit located at row i and column j of the state x with $0 \leq i < \ell$ and $0 \leq j < 6$. It is important to emphasize that in the remainder of this paper, all the indices start from zero and the zero-th bit or word is always considered the most significant one. Besides, note that if there is an addition or a subtraction in the indices of the state, it is always in modulo ℓ for the first (row) index and in modulo 6 for the second (column) index.

Initialization: The cipher receives a 6ℓ-bit plaintext and initializes the internal state with it using the same order used for indexing bits, i.e. it first fills $x_{[0,0]}$, then $x_{[0,1]}$ and so on. Then, $\mathtt{r}$ round functions, $\mathcal{R}_r$ (with $0 \leq r < \mathtt{r}$), are applied on the internal state, the first $\mathtt{r}-1$ ones of which (up to the round keys and round constants) are identical. Each round function is composed of the following four different operations: (2×) `SubBox`, (2×) `ShiftColumns`, `MixColumns`, `AddRoundConstant` and `AddRoundKey`. Considering $x \in \mathbb{F}_2^{\ell \times 6}$ as the input, $y \in \mathbb{F}_2^{\ell \times 6}$ as the output of operations, $0 \leq i < \ell$ and $0 \leq j < 6$, the round operations are defined as follows:



- **`SubBox` (`SB`):** The 6-bit S-box S is applied to each row of the state.

$$(y_{[i,0]}, y_{[i,1]}, y_{[i,2]}, y_{[i,3]}, y_{[i,4]}, y_{[i,5]}) = S(x_{[i,0]}, x_{[i,1]}, x_{[i,2]}, x_{[i,3]}, x_{[i,4]}, x_{[i,5]}) \,, \quad \forall\, i \,.$$

  The table for the S-box (in hexadecimal notation) is given in Table 3 and its implementation based on two-level NAND trees is shown in Figure 3.

- **`ShiftColumns` (`SC`):** The j-th column of the state is rotated upside by j bits.

$$y_{[i,j]} = x_{[i+j,j]} \,, \quad \forall\, i, j \,.$$

- **`MixColumns` (`MC`):** A cyclic binary matrix is multiplied to each column of the state.

$$y_{[i,j]} = x_{i,j} \oplus x_{[i+\alpha_1,j]} \oplus x_{[i+\alpha_2,j]} \oplus x_{[i+\alpha_3,j]} \oplus x_{[i+\alpha_4,j]} \oplus x_{[i+\alpha_5,j]} \oplus x_{[i+\alpha_6,j]} \,, \quad \forall\, i, j \,.$$

  For simplicity, we identify the applied matrix with $\alpha = (\alpha_1, \ldots, \alpha_6)$ that is parameterized for each version of the cipher with different ℓ value.

- **`AddRoundKey` ($\mathtt{A}_{k_r}$):** The 6ℓ-bit round key k_r is XORed to the whole of the state.

$$y_{[i,j]} = x_{[i,j]} \oplus k_{r\,[i,j]} \,, \quad \forall\, i, j \,.$$

- **`AddRoundConstant` ($\mathtt{A}_{c_r}$):** The 6ℓ-bit constant c_r is XORed to the whole of the state.

$$y_{[i,j]} = x_{[i,j]} \oplus c_{r\,[i,j]} \,, \quad \forall\, i, j \,.$$

  Similar to `PRINCE`, the round constants are chosen as the binary digits of the number $\pi - 3 = 0.1415\ldots$. Table 5 presents the first 100×64 bits of this constant. We use the first 6ℓ bits as c_0, the second 6ℓ bits as c_1 and so on.

Round Function: Using the above mentioned round operations, the first $\mathtt{r}-1$ round functions (with $0 \leq r \leq \mathtt{r}-2$) are defined as

$$\mathcal{R}_r = \mathtt{A}_{c_r} \circ \mathtt{MC} \circ \mathtt{SC} \circ \mathtt{SB} \circ \mathtt{SC} \circ \mathtt{SB} \circ \mathtt{A}_{k_r} \,,$$

while in the last round, the linear layer and constant addition are omitted, and instead an extra key addition is applied, i.e.,

$$\mathcal{R}_{\mathtt{r}-1} = \mathtt{A}_{k_{\mathtt{r}}} \circ \mathtt{SB} \circ \mathtt{SC} \circ \mathtt{SB} \circ \mathtt{A}_{k_{\mathtt{r}-1}} \,.$$

Table 5: The first 100×64 bits of the constant used in the round constants of SPEEDY.

0	243f6a8885a308d3	13198a2e03707344	a4093822299f31d0	082efa98ec4e6c89
1	452821e638d01377	be5466cf34e90c6c	c0ac29b7c97c50dd	3f84d5b5b5470917
2	9216d5d98979fb1b	d1310ba698dfb5ac	2ffd72dbd01adfb7	b8e1afed6a267e96
3	ba7c9045f12c7f99	24a19947b3916cf7	0801f2e2858efc16	636920d871574e69
4	a458fea3f4933d7e	0d95748f728eb658	718bcd5882154aee	7b54a41dc25a59b5
5	9c30d5392af26013	c5d1b023286085f0	ca417918b8db38ef	8e79dcb0603a180e
6	6c9e0e8bb01e8a3e	d71577c1bd314b27	78af2fda55605c60	e65525f3aa55ab94
7	5748986263e81440	55ca396a2aab10b6	b4cc5c341141e8ce	a15486af7c72e993
8	b3ee1411636fbc2a	2ba9c55d741831f6	ce5c3e169b87931e	afd6ba336c24cf5c
9	7a32538128958677	3b8f48986b4bb9af	c4bfe81b66282193	61d809ccfb21a991
10	487cac605dec8032	ef845d5de98575b1	dc262302eb651b88	23893e81d396acc5
11	0f6d6ff383f44239	2e0b4482a4842004	69c8f04a9e1f9b5e	21c66842f6e96c9a
12	670c9c61abd388f0	6a51a0d2d8542f68	960fa728ab5133a3	6eef0b6c137a3be4
13	ba3bf0507efb2a98	a1f1651d39af0176	66ca593e82430e88	8cee8619456f9fb4
14	7d84a5c33b8b5ebe	e06f75d885c12073	401a449f56c16aa6	4ed3aa62363f7706
15	1bfedf72429b023d	37d0d724d00a1248	db0fead349f1c09b	075372c980991b7b
16	25d479d8f6e8def7	e3fe501ab6794c3b	976ce0bd04c006ba	c1a94fb6409f60c4
17	5e5c9ec2196a2463	68fb6faf3e6c53b5	1339b2eb3b52ec6f	6dfc511f9b30952c
18	cc814544af5ebd09	bee3d004de334afd	660f2807192e4bb3	c0cba85745c8740f
19	d20b5f39b9d3fbdb	5579c0bd1a60320a	d6a100c6402c7279	679f25fefb1fa3cc
20	8ea5e9f8db3222f8	3c7516dffd616b15	2f501ec8ad0552ab	323db5fafd238760
21	53317b483e00df82	9e5c57bbca6f8ca0	1a87562edf1769db	d542a8f6287effc3
22	ac6732c68c4f5573	695b27b0bbca58c8	e1ffa35db8f011a0	10fa3d98fd2183b8
23	4afcb56c2dd1d35b	9a53e479b6f84565	d28e49bc4bfb9790	e1ddf2daa4cb7e33
24	62fb1341cee4c6e8	ef20cada36774c01	d07e9efe2bf11fb4	95dbda4dae909198

Key Schedule: The cipher receives a 6ℓ-bit master key and initializes it to the state of the zero-th round key (k_0). Then, it applies the bit permutation PB to compute the next round key, i.e., using the following permutation P, the positions of the bits are changed. That is

$$k_{r+1} = \mathtt{PB}(k_r) \quad \text{with} \quad k_{r+1\,[i',j']} = k_{r\,[i,j]}\,,$$

such that

$$(i', j') := P(i, j) \quad \text{with} \quad (6i' + j') \equiv \big(\beta \cdot (6i + j) + \gamma\big) \bmod 6\ell\,,$$

i.e., i' and j' are the quotient and remainder of dividing $\big(\beta \cdot (6i + j) + \gamma\big) \bmod 6\ell$ to 6, respectively. The parameters β and γ are dependent on the block length of the cipher with the condition of $\gcd(\beta, 6\ell) = 1$.

Instantiation: As already mentioned, SPEEDY is a family of block ciphers that allows instantiations of a wide range of block sizes and security levels. One may choose the block size of the encryption (6ℓ) by to the type of data blocks that need to be encrypted, and select the number of rounds (r) based on the necessary security level. By applying an appropriate $\alpha = (\alpha_1, \ldots, \alpha_6)$ value with regards to the rationale explained in Section 5, SPEEDY-r-6ℓ is ready to use.
To provide encryption of 64-bit blocks, which is the common instruction and data width in modern CPUs, we suggest to instantiate SPEEDY-r-192 with $\alpha = (1, 5, 9, 15, 21, 26)$ as the linear layer's parameter. We leave the number of rounds to be chosen based on the required security level. That is, for 128- and 192-bit security levels, we recommend using $\mathtt{r} \geq 6$ and $\mathtt{r} \geq 7$ rounds, respectively. More details about our security claims are provided

Table 6: P bit-permutation for `SPEEDY-r-192` with $\ell = 32$, $\beta = 7$ and $\gamma = 1$.

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
$P(i)$	1	8	15	22	29	36	43	50	57	64	71	78	85	92	99	106	113	120	127	134	141	148	155	162
i	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
$P(i)$	169	176	183	190	5	12	19	26	33	40	47	54	61	68	75	82	89	96	103	110	117	124	131	138
i	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71
$P(i)$	145	152	159	166	173	180	187	2	9	16	23	30	37	44	51	58	65	72	79	86	93	100	107	114
i	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
$P(i)$	121	128	135	142	149	156	163	170	177	184	191	6	13	20	27	34	41	48	55	62	69	76	83	90
i	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111	112	113	114	115	116	117	118	119
$P(i)$	97	104	111	118	125	132	139	146	153	160	167	174	181	188	3	10	17	24	31	38	45	52	59	66
i	120	121	122	123	124	125	126	127	128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143
$P(i)$	73	80	87	94	101	108	115	122	129	136	143	150	157	164	171	178	185	0	7	14	21	28	35	42
i	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159	160	161	162	163	164	165	166	167
$P(i)$	49	56	63	70	77	84	91	98	105	112	119	126	133	140	147	154	161	168	175	182	189	4	11	18
i	168	169	170	171	172	173	174	175	176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191
$P(i)$	25	32	39	46	53	60	67	74	81	88	95	102	109	116	123	130	137	144	151	158	165	172	179	186

below. The security analysis and the implementation of this instance are discussed in Section 6 and Section 7, respectively. Furthermore, for this instance we suggest to use $\beta = 7$ and $\gamma = 1$ for the key schedule parameters that the corresponding permutation P (given in Table 6) receives.
We provide several test vectors for `SPEEDY-r-192` encryption in Appendix C.

Security Claim While `SPEEDY` can be instantiated with different block and key sizes, the default is 192 bit as it constitutes the least common multiple of 6 (our S-box width) and 64 (the instruction width in high-end CPUs). We expect that `SPEEDY-r-192` achieves 128-bit security when iterated over $\mathtt{r} = 6$ rounds and full 192-bit security when iterated over $\mathtt{r} = 7$ rounds, while the $\mathtt{r} = 5$ round variant already provides a decent security level that is sufficient for many practical applications ($\geq 2^{128}$ time complexity when data complexity is limited to $\leq 2^{64}$). Compared to the security claims made for example for `PRINCE` ($\geq 2^{127-n}$ time complexity when data complexity is limited to $\leq 2^n$) or `PRINCEv2` ($\geq 2^{112}$ time complexity when data complexity is limited to $\leq 2^{50}$) the security level claimed by `SPEEDY-5-192` is already superior.

5 Design Rationale

The primary criterion for the design of `SPEEDY` is to use round operations with a low latency that still provide good enough cryptographic properties to provide a secure encryption with a small number of rounds. To achieve this goal, we applied the ultra low-latency S-box found in Section 3. While the design approach for the S-box is described in Section 3, all details regarding the design choices for the other round operations are explained in the following.

`MixColumns`: It is clear that the latency cost (in terms of XOR gate depth) of XORing n bits, i.e., $x_0 \oplus \ldots \oplus x_{n-1}$ is equal to $d = \lfloor \log_2 n \rfloor$. This means that XORing n bits with $2^{d-1} < n \leq 2^d$, has the same cost for all n values with respect to the latency of the circuit (considering identical topology). Therefore, to use the maximum capacity of the given latency, it is prudent to choose $n = 2^d$.

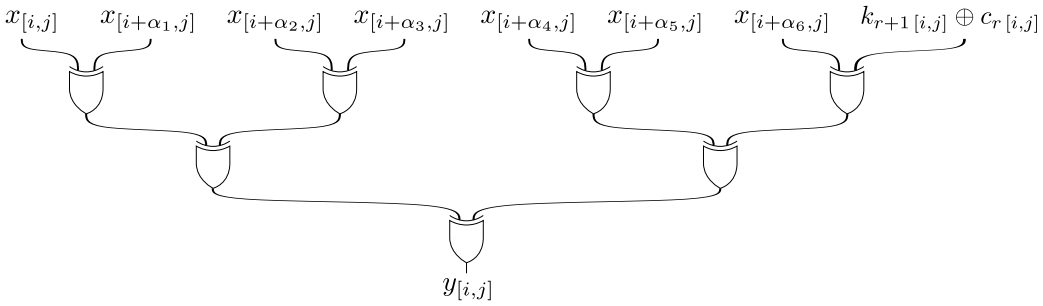


Figure 4: Implementation of each output bit of the merged function $\mathtt{A}_{k_{r+1}} \circ \mathtt{A}_{c_r} \circ \mathtt{MC}$ of the SPEEDY design.

In the design of SPEEDY, since the $\mathtt{A}_{k_{r+1}}$ operation from round $r+1$ occurs right after the $\mathtt{A}_{c_r}$ and MC operations from the r-th round, it is possible to merge all three operations. Considering that x and y from $\mathbb{F}_2^{\ell \times 6}$ are the input and output of the merged $\mathtt{A}_{k_{r+1}} \circ \mathtt{A}_{c_r} \circ \mathtt{MC}$ operation, respectively, then each output bit can be calculated as

$$y_{[i,j]} = x_{[i,j]} \oplus x_{[i+\alpha_1,j]} \oplus x_{[i+\alpha_2,j]} \oplus x_{[i+\alpha_3,j]} \oplus x_{[i+\alpha_4,j]} \oplus x_{[i+\alpha_5,j]} \oplus x_{[i+\alpha_6,j]} \oplus (k_{r+1\,[i,j]} \oplus c_{r\,[i,j]}) \,.$$

Hence, it is possible to implement the whole $\mathtt{A}_{k_{r+1}} \circ \mathtt{A}_{c_r} \circ \mathtt{MC}$ as a merged function within three XOR gate levels. Note that since the input $k_{r+1\,[i,j]}$ is not in the critical path of the circuit, $k_{r+1\,[i,j]}$ and $c_{r\,[i,j]}$ can be combined with each other beforehand. Depending on the value of the round constant bit, we actually only need to use $k_{r+1\,[i,j]}$ itself or its inverted value $\neg k_{r+1\,[i,j]}$. Figure 4 depicts the corresponding circuit to implement each output bit of the merged function. Please note that the fan-out of each XOR gate in this circuit is 1. It is important to consider that for CMOS technologies where the XNOR gate is significantly faster than the XOR gate (such as NanGate 45 nm), it is easily possible to implement this linear layer with only XNOR gates instead of XORs and simply exchange the buffers and inverters of the next S-box stage to revert its inverted output.
For the MC operation, we decided to use the same binary cyclic matrix with polynomial representation of $1 + z^{\alpha_1} + \ldots + z^{\alpha_{w-1}}$ and multiply it with each column of the state. Therefore, each output bit of the MC operation is the XOR of w input bits. As explained above, the optimal choices for w are 3, 7, 15 and so on, so that it is possible to implement the above mentioned merged function with 2, 3, 4 XOR gate levels, respectively. While in PRINCE, MIDORI and QARMA block ciphers, this technique of merging is used by applying cyclic matrices of $w = 3$ and repeated after each S-box layer, we found that it is a good trade-off to use cyclic matrices with $w = 7$, but only after each second S-box layer, which is effectively cheaper from a latency cost perspective.
For each SPEEDY-r-6ℓ version of the cipher, we need to find a bijective $\ell \times \ell$ binary cyclic matrix M with polynomial representation of $1 + z^{\alpha_1} + \ldots + z^{\alpha_6}$. Finding an appropriate bijective cyclic matrix with $w = 7$ being an odd integer, is quite possible for wide range of ℓ. But, since the value of $\alpha = (\alpha_1, \ldots, \alpha_6)$ is always dependent on the value of ℓ, we leave it as a parameter of the cipher's instantiation.
Since, the probability of M being a non-singular matrix is high, we can add extra criteria regarding the choice of the α parameter.

- All values for α_1, $\alpha_2 - \alpha_1$, $\alpha_3 - \alpha_2$, $\alpha_4 - \alpha_3$, $\alpha_5 - \alpha_4$, $\alpha_6 - \alpha_5$ and $\ell - \alpha_6$ need to be smaller or equal to 6. The reason for this criterion is explained later, in the corresponding paragraph for ShiftColumns. Note that this criterion is only possible for $\ell \leq 42$.

- Maximum branch number: Branch number of a matrix is defined as

$$bn := \min_{x \in \mathbb{F}_2^\ell \setminus \{0\}} \mathrm{hw}(x) + \mathrm{hw}(M \times x^T)\,,$$

where hw denotes the Hamming weight of a binary array. In case of a bijective $\ell \times \ell$ binary cyclic matrix M with polynomial representation of $1 + z^{\alpha_1} + \ldots + z^{\alpha_{w-1}}$, the branch number cannot be higher than $w + 1$. In our case, we restrict the choice of the α parameter to the ones which provide maximum branch number, i.e., 8.

- For the corresponding matrix M of parameter $\alpha = (\alpha_1, \ldots, \alpha_6)$, we build a binary table H such that the element in the position (i, j) is `1`, if and only if there is an $x \in \mathbb{F}_2^\ell \setminus \{0\}$ with $\mathrm{hw}(x) = i$ and $\mathrm{hw}(M \times x^T) = j$. Then, we compute the following three numbers:

$$\begin{aligned}
bn_3 &= \min_{\substack{i_1,i_2,i_3 \\ H[i_1][i_2]=H[i_2][i_3]=1}} i_1 + i_2 + i_3\,,\\
bn_4 &= \min_{\substack{i_1,i_2,i_3,i_4 \\ H[i_1][i_2]=H[i_2][i_3]=H[i_3][i_4]=1}} i_1 + i_2 + i_3 + i_4\,,\\
bn_5 &= \min_{\substack{i_1,i_2,i_3,i_4,i_5 \\ H[i_1][i_2]=H[i_2][i_3]=H[i_3][i_4]=H[i_4][i_5]=1}} i_1 + i_2 + i_3 + i_4 + i_5\,.
\end{aligned} \tag{5}$$

As explained later in Section 6, larger values for bn_r lead to a stronger resistance of the r-round `SPEEDY` against differential and linear attacks. Therefore, for all the possible choices of α which are meeting the first two criteria, we compute the above bn_r numbers and choose one of the corresponding α values which leads to the maximum bn_r values.

It is noteworthy that the branch number bn is the same as bn_2 defined as

$$bn_2 = \min_{\substack{i_1,i_2 \\ H[i_1][i_2]=1}} i_1 + i_2\,.$$

Besides, bn_r with $r > 2$ can be considered as an extension for the definition of branch number, and hereafter, we will call it a *higher-order branch number*.

In the case of `SPEEDY-r-192`, with $\ell = 32$, we applied the above criteria and end up with 30 choices from which we choose the first one that is $\alpha = (1, 5, 9, 15, 21, 26)$ with $bn_3 = 13$, $bn_4 = 20$, and $bn_5 = 25$. It is important to mention that the corresponding matrix for inverse of the `MC` operation is a cyclic matrix with $w = 19$ and $\alpha^{-1} = (4, 5, 6, 7, 10, 12, 14, 15, 16, 18, 19, 20, 21, 22, 23, 24, 25, 28)$.

`ShiftColumns`: The existence of the first `SC` operation, right after the first `SB` makes it possible that input bits of each S-box in the second `SB` operation are all from the outputs of different S-boxes of the first `SB` operation. Therefore, since the applied S-box has the full diffusion property (in both straight and inverse direction), each output bit of $\texttt{SB} \circ \texttt{SC} \circ \texttt{SB}$ is a function of 36 consecutive input bits. Namely, for $\texttt{SB} \circ \texttt{SC} \circ \texttt{SB}$, the output bit in the position $[i, j]$ is a function of all input bits in the position of the form $[i + p, q]$ with $0 \leq p, q < 6$, while for $(\texttt{SB} \circ \texttt{SC} \circ \texttt{SB})^{-1}$, the output bit $[i, j]$ is a function of all input bits of the form $[i - p, q]$.
By considering the first criterion for `MixColumns`, namely that α_1, $\alpha_2 - \alpha_1$, $\alpha_3 - \alpha_2$, $\alpha_4 - \alpha_3$, $\alpha_5 - \alpha_4$, $\alpha_6 - \alpha_5$ and $\ell - \alpha_6$ are all smaller or equal to 6, it means that the output bit of $\texttt{MC} \circ \texttt{SB} \circ \texttt{SC} \circ \texttt{SB}$ and equivalently, output of one *key-less round function* $\texttt{MC} \circ \texttt{SC} \circ \texttt{SB} \circ \texttt{SC} \circ \texttt{SB}$ is dependent on the whole 6ℓ input bits. The same holds for $(\texttt{MC} \circ \texttt{SB} \circ \texttt{SC} \circ \texttt{SB})^{-1}$ in the decryption side, hence, the input of one key-less round function is dependent on the whole 6ℓ output bits.

Moreover, the same arguments hold for inserting the second SC, right after the second SB operation, which means that each output bit of SB∘MC∘SC∘SB depends on the whole 6ℓ input bits which equivalently holds for the *rotated key-less round function* SC ∘ SB ∘ MC ∘ SC ∘ SB. Altogether, one key-less round function or rotated round function, in both encryption and decryption directions, provides full diffusion. In other words, in a key recovery attack, to compute one output bit of those functions, the attacker needs to know the value of the whole input state. Note that knowing the value for the whole input state of these functions requires knowing the whole state of the round key. This means, if the attacker wants to extend a distinguisher by appending one complete round (or rotated round) function, to do a key recovery attack, he needs to guess the whole 6ℓ bits of the key.
It is important to mention that since existence of any key-independent linear operation right before the ciphertext does not add any security to the encryption, we exclude the MC and the second SC operations from the last round.

Key Schedule: Since the main target of our design is to provide a low-latency encryption routine, and since other cost factors of the implementation such as area or energy consumption of the circuits are only secondary priorities, one can apply a key schedule built from costly operations. Yet, since we do not aim for related-key security, and since the round function has a strong diffusion, we found that using a linear key schedule is sufficient for our purposes. Besides, updating round keys by a bit-permutation function in an unrolled implementation has no latency, area or energy costs, thus we decided to use such a key schedule. Furthermore, we wanted to use a bit-permutation such that it is easy to generalize for all SPEEDY-r-6ℓ members. To do so, we chose the general affine mapping in the finite integer field of $\{0, \ldots, 6\ell - 1\}$, that the permutation P maps x, an element of this field, to $P(x) = \beta x + \gamma \mod 6\ell$. The only requirement for P being a bijection is that β and 6ℓ need to be co-prime, i.e., $\gcd(\beta, 6\ell) = 1$.

6 Security Analysis

In this section, we provide details about the cryptographic properties of the SPEEDY family of block ciphers. We start with differential, linear and algebraic properties of the S-box S and expand them over a round function of the cipher. By applying properties for the round function, we discuss the security of an r round structure of SPEEDY.

Cryptographic Properties of the S-box: The S-box S, presented in Section 3, is the heart of the SPEEDY design and it needs to be studied in detail. As described before the uniformity and linearity of S is equal to 8 and 24, respectively. This means that the maximum probability of differentials over S is $8 \cdot 2^{-6} = 2^{-3}$ and the maximum absolute correlation of linear approximations is $24 \cdot 2^{-6} = 3 \cdot 2^{-3}$ (equally means that the maximum potential of linear approximations is $(3 \cdot 2^{-3})^2 = 9 \cdot 2^{-6} \approx 2^{-2.83}$). As one important part of the Differential Distribution Table (DDT) and Linear Approximation Table (LAT), we present the 1-bit to 1-bit differentials and linear approximations in Table 7. In more detail, entry (i, j) of the 1-bit to 1-bit DDT denotes the probability that having only one active bit in the position i of the S-box inputs leads to only one active bit in the position j of the S-box output. In case of 1-bit to 1-bit LAT, entry (i, j) of the table denotes the absolute correlation value for the $x_i = y_j$ linear approximation.
Even though, one of the criteria for building the low-latency S-box was to provide full dependency of the output bits on the input bits, this is not sufficient to provide all information about algebraic properties of the function. We provide the algebraic normal form (ANF) representation of both S and S^{-1} below. As shown, not only all the input/output variables are non-linearly involved in all the output/input coordinates (i.e., the S-box provides full diffusion in both straight and inverse directions), each coordinate function is

quite dense with respect to the number of involved terms. Another interesting information is that the ANF degree for coordinates of S is 5, 3, 3, 3, 4 and 5, respectively, while in the case of S^{-1}, these numbers are 5, 4, 5, 4, 5 and 5, respectively.

$$
\begin{aligned}
y_0 &= x_3 \oplus x_5x_3 \oplus x_5x_4x_3x_2 \oplus x_5x_4x_1 \oplus x_5x_4x_3x_2x_1 \oplus x_1x_0 \oplus x_5x_4x_1x_0 \oplus x_3x_1x_0 \oplus \\
&\quad x_5x_4x_3x_1x_0 \\
y_1 &= x_3 \oplus x_4x_3 \oplus x_5x_4x_3 \oplus x_5x_3x_2 \oplus x_1 \oplus x_3x_1 \oplus x_5x_2x_0 \oplus x_1x_0 \oplus x_3x_1x_0 \\
y_2 &= 1 \oplus x_5 \oplus x_5x_2 \oplus x_4x_2 \oplus x_3x_2 \oplus x_4x_3x_2 \oplus x_0 \oplus x_5x_0 \oplus x_4x_0 \oplus x_4x_3x_0 \oplus x_2x_0 \oplus \\
&\quad x_5x_2x_0 \oplus x_3x_1x_0 \,, \\
y_3 &= x_2 \oplus x_3x_2 \oplus x_3x_1 \oplus x_5x_0 \oplus x_2x_0 \oplus x_5x_2x_0 \oplus x_4x_2x_0 \oplus x_3x_2x_0 \oplus x_3x_1x_0 \\
y_4 &= x_5x_4 \oplus x_1 \oplus x_4x_1 \oplus x_2x_1 \oplus x_4x_2x_1 \oplus x_0 \oplus x_5x_4x_0 \oplus x_4x_3x_0 \oplus x_3x_2x_0 \oplus x_4x_3x_2x_0 \oplus \\
&\quad x_1x_0 \oplus x_4x_1x_0 \oplus x_2x_1x_0 \oplus x_4x_2x_1x_0 \,, \\
y_5 &= x_4 \oplus x_5x_2 \oplus x_4x_2 \oplus x_4x_1 \oplus x_4x_2x_1 \oplus x_3x_0 \oplus x_4x_3x_0 \oplus x_5x_3x_2x_0 \oplus x_4x_3x_2x_0 \oplus \\
&\quad x_3x_1x_0 \oplus x_4x_3x_1x_0 \oplus x_2x_1x_0 \oplus x_5x_2x_1x_0 \oplus x_5x_3x_2x_1x_0 \oplus x_4x_3x_2x_1x_0 \,.
\end{aligned}
$$

$$
\begin{aligned}
x_0 &= y_4 \oplus y_5y_4 \oplus y_5y_4y_2 \oplus y_5y_1 \oplus y_4y_1 \oplus y_5y_4y_3y_1 \oplus y_5y_3y_2y_1 \oplus y_4y_3y_2y_1 \oplus y_5y_4y_3y_2y_1 \oplus \\
&\quad y_5y_0 \oplus y_5y_4y_0 \oplus y_2y_0 \oplus y_4y_2y_0 \oplus y_3y_2y_0 \oplus y_4y_3y_2y_0 \oplus y_5y_1y_0 \oplus y_2y_1y_0 \,, \\
x_1 &= y_5y_3 \oplus y_5y_4y_3 \oplus y_5y_3y_2 \oplus y_5y_4y_3y_2 \oplus y_4y_1 \oplus y_5y_4y_1 \oplus y_3y_1 \oplus y_4y_3y_1 \oplus y_2y_1 \oplus \\
&\quad y_4y_2y_1 \oplus y_3y_2y_1 \oplus y_4y_3y_2y_1 \oplus y_4y_0 \oplus y_5y_4y_0 \oplus y_3y_0 \oplus y_4y_3y_0 \oplus y_5y_4y_3y_0 \oplus y_2y_0 \oplus \\
&\quad y_5y_2y_0 \oplus y_4y_2y_0 \oplus y_3y_2y_0 \oplus y_5y_3y_2y_0 \oplus y_4y_3y_2y_0 \oplus y_4y_1y_0 \oplus y_5y_4y_1y_0 \oplus y_3y_1y_0 \oplus \\
&\quad y_4y_3y_1y_0 \,, \\
x_2 &= y_5 \oplus y_5y_4 \oplus y_3 \oplus y_5y_3 \oplus y_4y_3 \oplus y_5y_2 \oplus y_4y_2 \oplus y_5y_3y_2 \oplus y_5y_3y_1 \oplus y_5y_2y_1 \oplus y_4y_2y_1 \oplus \\
&\quad y_5y_4y_2y_1 \oplus y_5y_4y_3y_2y_1 \oplus y_0 \oplus y_4y_0 \oplus y_5y_4y_0 \oplus y_3y_0 \oplus y_4y_3y_0 \oplus y_5y_4y_3y_0 \oplus y_2y_0 \oplus \\
&\quad y_5y_4y_2y_0 \oplus y_5y_3y_2y_0 \oplus y_5y_1y_0 \oplus y_5y_2y_1y_0 \oplus y_4y_2y_1y_0 \,, \\
x_3 &= y_5 \oplus y_5y_4 \oplus y_5y_2 \oplus y_5y_4y_2 \oplus y_1 \oplus y_5y_1 \oplus y_4y_1 \oplus y_3y_1 \oplus y_5y_3y_1 \oplus y_2y_1 \oplus y_4y_2y_1 \oplus \\
&\quad y_5y_4y_2y_1 \oplus y_3y_2y_1 \oplus y_4y_3y_2y_1 \oplus y_0 \oplus y_5y_0 \oplus y_4y_0 \oplus y_5y_2y_0 \oplus y_1y_0 \oplus y_5y_1y_0 \oplus \\
&\quad y_4y_1y_0 \oplus y_3y_1y_0 \oplus y_5y_3y_1y_0 \oplus y_4y_3y_1y_0 \oplus y_2y_1y_0 \oplus y_3y_2y_1y_0 \,, \\
x_4 &= y_5y_4 \oplus y_3 \oplus y_5y_3 \oplus y_4y_3 \oplus y_5y_4y_3 \oplus y_5y_2 \oplus y_5y_4y_2 \oplus y_3y_2 \oplus y_5y_4y_3y_2 \oplus y_5y_3y_1 \oplus \\
&\quad y_2y_1 \oplus y_4y_2y_1 \oplus y_5y_4y_2y_1 \oplus y_3y_2y_1 \oplus y_5y_3y_2y_1 \oplus y_0 \oplus y_4y_0 \oplus y_3y_0 \oplus y_5y_3y_0 \oplus \\
&\quad y_4y_3y_0 \oplus y_5y_4y_3y_0 \oplus y_5y_2y_0 \oplus y_4y_2y_0 \oplus y_5y_4y_2y_0 \oplus y_3y_2y_0 \oplus y_1y_0 \oplus y_2y_1y_0 \oplus \\
&\quad y_4y_2y_1y_0 \oplus y_4y_3y_2y_1y_0 \,, \\
x_5 &= 1 \oplus y_4 \oplus y_5y_4 \oplus y_3 \oplus y_5y_3 \oplus y_2 \oplus y_4y_2 \oplus y_5y_4y_2 \oplus y_3y_2 \oplus y_4y_1 \oplus y_5y_4y_1 \oplus y_4y_3y_1 \oplus \\
&\quad y_5y_4y_3y_1 \oplus y_5y_2y_1 \oplus y_4y_2y_1 \oplus y_5y_4y_2y_1 \oplus y_5y_3y_2y_1 \oplus y_0 \oplus y_4y_0 \oplus y_3y_0 \oplus y_5y_3y_0 \oplus \\
&\quad y_4y_3y_0 \oplus y_5y_4y_3y_0 \oplus y_2y_0 \oplus y_4y_2y_0 \oplus y_3y_2y_0 \oplus y_5y_4y_1y_0 \oplus y_5y_3y_1y_0 \oplus y_5y_2y_1y_0 \oplus \\
&\quad y_3y_2y_1y_0 \oplus y_4y_3y_2y_1y_0 \,.
\end{aligned}
$$

Cryptographic Properties of `SB` ∘ `SC` ∘ `SB`: Since in the round function of `SPEEDY`, two `SB` operations are connected through the `SC` operation which is a simple bit permutation, it is necessary to look at the properties of this combination. We first investigate the 1-bit

Table 7: 1-bit to 1-bit differential probabilities and linear correlations of the `SPEEDY` S-box.

differential ($\times 2^{-5}$)						
$i\backslash j$	0	1	2	3	4	5
0	-	1	3	2	1	1
1	4	3	4	4	-	-
2	1	1	3	3	1	1
3	1	3	-	2	3	-
4	2	2	4	4	2	1
5	2	4	2	4	-	2

linear ($\times 2^{-4}$)						
$i\backslash j$	0	1	2	3	4	5
0	3	-	4	-	4	4
1	6	4	4	4	2	4
2	1	-	-	4	4	6
3	6	4	4	-	6	2
4	4	4	-	4	-	3
5	4	4	4	4	4	5

to 1-bit differentials and linear approximations of $\texttt{SB} \circ \texttt{SC} \circ \texttt{SB}$. Since each input bit of the second SB operation comes from a different first-stage S-box, 1-bit to 1-bit transitions over $\texttt{SB} \circ \texttt{SC} \circ \texttt{SB}$ are possible if and only if the transitions over the first and second SB operations, both are 1-bit to 1-bit transitions. Besides, without any extra assumption (such as independency between the state bits), it can be proven that probability or correlation of this 1-bit to 1-bit transitions over $\texttt{SB} \circ \texttt{SC} \circ \texttt{SB}$ is the multiplication of probabilities or correlations over two active S-boxes (one from the first SB and another from the second SB operation).
Since SC does not change the column position of active bits, it is easily possible to compute these probabilities. Table 8 presents the 1-bit to 1-bit differential probabilities and linear correlations over $\texttt{SB} \circ \texttt{SC} \circ \texttt{SB}$ such that entry $[i, j]$ denotes the maximum possible probabilities or linear correlations that an active input bit in the column i transits to an active output bit in the column j. To compute these values, we used the following equation which T_1 and T_2 denote the Table 7 and Table 8, respectively.

$$T_2[i, j] = \max_k \; T_1[i, k] \cdot T_1[k, j] \,.$$

Note that the maximum entry for differential transitions is 2^{-6} and for linear transitions it is $15 \cdot 2^{-7} \approx 2^{-3}$. We are only interested in 1-bit to 1-bit transitions, because the probability or the correlation of such transitions are among the highest ones and also because based on such transitions, we can build differential or linear characteristics with a high differential probability or linear correlation.
Again due to the fact that SC does not change the column position of the bits and each input bit of the second SB is the output of a different S-box, it is possible to compute the algebraic degree of $\texttt{SB} \circ \texttt{SC} \circ \texttt{SB}$. The degree of any output bit in the columns 0, 1, ... and 5 is equal to 19, 15, 13, 13, 13 and 20, respectively.
It is important to mention that replacing the current S-box with another bit-permutation equivalent S-box will change differential, linear and algebraic properties of $\texttt{SB} \circ \texttt{SC} \circ \texttt{SB}$. While in Section 3, we ended up with a bit-permutation equivalency class of S-boxes, we tried all the S-boxes of this class to find an S-box such that the maximum entry in Table 8 and also the number of entries with maximum value are as small as possible. Moreover, we want the minimum algebraic degree over $\texttt{SB} \circ \texttt{SC} \circ \texttt{SB}$ coordinates to be as large as possible. Note that due to the structure of the round function, since encryption with S-box $P_{out} \circ S \circ P_{in}$ is identical to encryption with S-box $P_{in} \circ P_{out} \circ S$ (up to a column permutation in the state of plaintext, ciphertext, round key and round constants), we can consider one of them to be the identity bit-permutation and only need to choose the other one.

Differential and Linear Attacks Since there are 1-bit to 1-bit differential and linear approximations over $\texttt{SB} \circ \texttt{SC} \circ \texttt{SB}$ and the corresponding probability or correlation of those transitions are quite significant, it is necessary to choose a strong MC operation. The

Table 8: 1-bit to 1-bit differential probabilities and linear correlations over $\texttt{SB} \circ \texttt{SC} \circ \texttt{SB}$.

differential ($\times 2^{-10}$)						
$i \backslash j$	0	1	2	3	4	5
0	4	6	9	9	6	3
1	12	12	12	12	12	4
2	4	9	9	9	9	3
3	12	9	12	12	6	3
4	8	12	12	12	12	4
5	16	12	16	16	12	4

linear ($\times 2^{-8}$)						
$i \backslash j$	0	1	2	3	4	5
0	16	16	16	16	16	24
1	24	16	24	16	24	24
2	24	24	24	24	24	30
3	24	24	24	24	24	24
4	24	16	16	16	24	16
5	24	20	20	20	24	25

criterion of having branch number $bn = 8$ ensures that the maximum expected differential probability (EDP) of differential trails and the maximum expected linear potential (ELP) of linear trails over two rounds of SPEEDY is equal to $(2^{-6})^8 = 2^{-48}$.
To discuss the resistance of r-round SPEEDY, we use the *higher-order* branch number bn_r defined in Equation 5 to have an overview about the minimum number of active S-boxes in differential or linear trails. Therefore, using this estimation the maximum EDP of differentials and the ELP of linear trails over r-round SPEEDY is estimated by $2^{-6 \cdot bn_r}$.
In case of SPEEDY-r-192, with the recommended α parameter, we have

$$bn_3 = 13\,, \quad bn_4 = 20\,, \quad bn_5 = 25\,, \quad bn_6 = 32\,.$$

Hence, we estimate that EDP (resp. ELP) of any differential (resp. linear) trails over 3, 4, 5 and 6 rounds is smaller than 2^{-78}, 2^{-120}, 2^{-150} and 2^{-192}. Actually, assuming that all the 1-bit to 1-bit differential or linear transitions through the S-box are possible, and by considering that there are at most 8 active words (of 6-bit) per state of operations, we searched for the minimum number of active S-boxes. We found that this number is 13, 23 and 35 for 2, 3 and 4 rounds. Assuming that all these 1-bit to 1-bit transitions occur with differential probability (or linear potential) of 2^{-3}, the EDP (resp. ELP) of any differential (resp. linear) trails over 2, 3 and 4 rounds is smaller than 2^{-39}, 2^{-69} and 2^{-105}. We emphasize that these values are an upper bound, which means that a trail with such EDP or ELP must not necessarily exist.

Higher-Order Differential, Integral and Cube Attacks SPEEDY's round function has a strong diffusion and high algebraic degree. While, we investigate these properties for one complete round precisely, for a larger number of rounds, we expect that the ANF representation would be dense with respect to the number of involved terms. Therefore, we believe that these attacks are weaker than differential and linear attacks and less of a concern.

Number of Rounds For a low-latency block cipher, a large security margin is not reasonable and is usually considered as wasteful. Since the attacker cannot add more than one round to extend a distinguisher and therefore to use the distinguisher in a key recovery attack, we believe a security margin of one round is sufficient. Therefore, we recommend to choose the number of rounds with respect to the required security level of the block cipher's application. For example, in case of the SPEEDY-r-192 instance, we recommend to use SPEEDY-6-192 and SPEEDY-7-192 for 128-bit and 192-bit security levels, respectively, while for more practical applications, such as a security level of 2^{128} time and 2^{64} data complexity, we recommend to use SPEEDY-5-192.

Impossible Differential and Zero-Correlation Linear-Hull Attacks One active bit, with respect to both differentials and linear correlations, and in both forward and backward directions can propagate to all the state bits over one (rotated) key-less SPEEDY round function and more importantly, none of this activeness is deterministic. But, it should be noted that the activeness of these bits can be related to each other if the last operation is MC. Therefore, by combining one round propagation in the forward direction and one round propagation in the backward direction, it might be possible to find impossible differentials or zero-correlation linear-hulls over two (rotated) key-less round functions. But, if we add one SB operation in the middle, we ensure that there are no such distinguishers; in other words, there are no impossible differentials or zero-correlation linear-hulls over

$$(\mathtt{SB} \circ \mathtt{SC} \circ \mathtt{SB} \circ \mathtt{SC} \circ \mathtt{MC}) \circ \mathtt{SB} \circ (\mathtt{SC} \circ \mathtt{SB} \circ \mathtt{SC} \circ \mathtt{MC} \circ \mathtt{SB})$$

or

$$(\texttt{SB} \circ \texttt{SC} \circ \texttt{MC} \circ \texttt{SB} \circ \texttt{SC}) \circ \texttt{SB} \circ (\texttt{SC} \circ \texttt{MC} \circ \texttt{SB} \circ \texttt{SC} \circ \texttt{SB}).$$

Therefore, by applying the 2-round distinguisher and extending by one round for key recovery, it might be possible to have a successful attack on 3-round SPEEDY, but we expect that more than 3 rounds are secure against those attacks.

Meet-in-the-Middle Attack The maximum number of attacked rounds using meet-in-the-middle technique can be evaluated considering the maximum length of three features: partial-matching, initial structure and splice-and-cut. For partial-matching, the number of rounds in both forward and backward directions cannot reach the full diffusion rounds which for SPEEDY in both directions is smaller than one round. The condition for the initial structure is that the key differential trails in both forward and backward directions do not share active non-linear components. As any key differential in SPEEDY affects the whole state after one complete round in both directions, there is no such differential which shares active S-box(es) in more than one round. Therefore, it only works up to one round. Splice-and-cut may extend the number of attacked rounds up to the number of full diffusion rounds, i.e., again one round. Thus, it is not possible for the attacker to mount a successful meet-in-the-middle attack on a (2+1+1) = 4-round SPEEDY.

Implementation Attacks The protection of SPEEDY against implementation attacks like timing, power analysis or fault injection attacks is not a focus of this work. Clearly, a straightforward and unprotected implementation of SPEEDY will be susceptible to adversaries who are capable of observing the characteristics of the implementation during its execution. Although this attacker model traditionally requires physical access to the executing device and therefore is typically considered to be less of a concern for desktop and server CPUs (the targeted application area for SPEEDY) there have been more and more successful remote power analysis attacks on such devices recently, most notably the PLATYPUS attack [LKO+21]. Thus, even in such contexts, physical adversaries can no longer be ignored and protecting SPEEDY against said attacks is a great direction for future research.
In that regard, a recent work has pointed out that, although it is hardly feasible to apply hardware masking to unrolled low-latency cryptography without sacrificing a large portion of its performance due to the necessary inclusion of register stages, simple reset methods (i.e., randomly pre-charging the combinatorial circuit) deliver very promising results against passive side-channel attacks if applied properly [Moo20]. The parallelism, speed and asynchronicity of SPEEDY are assumed to be even higher than for the investigated PRINCE instance. Thus, we believe that this kind of protection mechanism can most reasonably be applied to unrolled SPEEDY in hardware without causing a large performance penalty. According to [Moo20], the cost of this countermeasure is either that the throughput is halved, or that the area is doubled when instantiating the unrolled cipher twice and alternating between pre-charging or encrypting with each circuit. Additionally, the cost for the Random Number Generator (RNG) has to be considered.

7 Hardware Implementation

In this section, we analyze the minimum achievable latency of fully-unrolled SPEEDY hardware implementations as well as the area required for the time-constrained circuits and compare them to a number of other cryptographic primitives that have been suggested for high-speed single-cycle encryption in literature. Implementing SPEEDY in hardware is rather straightforward since almost all round operations which require any logic and may not be realized through wiring alone are already chosen as circuit representations. In detail, Figure 3 shows the hardware circuitry for the 6-bit high-speed S-box while Figure 4

Table 9: Minimum latency of fully-unrolled encryption-only circuits of different cryptographic primitives.

	Minimum Latency [ns]					
	Commercial Foundry				NanGate OCL	
Cipher	90 nm LP	65 nm LP	40 nm LP	28 nm HPC	45 nm	15 nm
`Gimli E-M`	4.532467	3.330192	2.794736	1.178424	4.537304	0.435069
$\texttt{MANTIS}_6$	4.625529	3.405490	2.891383	1.278725	4.479773	0.437595
$\texttt{MANTIS}_7$	5.201681	3.722473	3.234409	1.421365	5.074452	0.492703
$\texttt{MANTIS}_8$	5.823127	4.233543	3.631438	1.594997	5.739020	0.552384
`Midori`	5.061255	3.582221	3.142355	1.362237	4.934847	0.481522
`Orthros`	3.862139	2.678637	2.401275	1.087139	3.774836	0.369497
`PRINCE`	4.101177	2.866749	2.521302	1.108886	4.059997	0.389144
`PRINCEv2`	4.047311	2.944367	2.509131	1.103273	4.077636	0.387146
$\texttt{QARMA}_5\text{-64-}\sigma_0$	4.075846	2.920377	2.498908	1.134901	4.014516	0.385281
$\texttt{QARMA}_6\text{-64-}\sigma_0$	4.770325	3.418600	2.951308	1.308331	4.554445	0.448931
$\texttt{QARMA}_7\text{-64-}\sigma_0$	5.449707	3.909138	3.389576	1.538606	5.336362	0.517093
$\texttt{QARMA}_8\text{-64-}\sigma_0$	6.103768	4.396543	3.814078	1.697027	5.966323	0.575525
$\texttt{QARMA}_5\text{-64-}\sigma_1$	4.515514	3.284252	2.815788	1.219624	4.367899	0.408580
$\texttt{QARMA}_6\text{-64-}\sigma_1$	5.297867	3.808675	3.271455	1.388353	4.944635	0.472798
$\texttt{QARMA}_7\text{-64-}\sigma_1$	6.014477	4.371963	3.745959	1.601572	5.800633	0.542712
$\texttt{QARMA}_8\text{-64-}\sigma_1$	6.720944	4.904521	4.202632	1.797539	6.498429	0.608985
`SPEEDY-5-192`	2.994643	2.178075	1.867064	0.847761	3.187368	0.300466
`SPEEDY-6-192`	3.637978	2.639186	2.277422	1.032206	3.848132	0.366762
`SPEEDY-7-192`	4.261928	3.087257	2.663004	1.217946	4.515505	0.431032
`SPEEDY-5-192` *	2.941130	2.121748	1.820950	0.826217	2.817971	0.290961
`SPEEDY-6-192` *	3.559981	2.573561	2.223863	1.011173	3.382270	0.353391
`SPEEDY-7-192` *	4.174183	3.029217	2.620612	1.186598	3.995325	0.413950

* = Optimized HDL code with direct instantiation of library cells based on Figures 3 and 4.

depicts the logic circuit that implements the combined $\texttt{A}_{k_{r+1}} \circ \texttt{A}_{c_r} \circ \texttt{MC}$ function. The `ShiftColumns` operation does not require any logic, which means that only the initial and the final `AddRoundKey` functions remain. Obviously these are implemented with a single stage of regular XOR gates.

Table 9 presents the minimum latency results achieved for different instances of `Gimli`, `MANTIS`, `Midori`, `Orthros`, `PRINCE`, `PRINCEv2`, `QARMA`, and `SPEEDY` (in alphabetical order). All results have been obtained by synthesizing the fully-unrolled cipher circuits between two register stages for minimum clock period using the *Synopsys Design Compiler Version O-2018.06-SP4* software while executing four stages of the `compile_ultra` command (three incremental). We have repeated the analysis with 6 different standard cell libraries, 4 of which are manufacturable cell libraries from a commercial foundry, while the remaining 2 are open-source libraries which are not manufacturable but can be used for producing universally comparable and reproducible synthesis results. Please note that `Gimli` is a key-less permutation. Therefore, in order to create an encryption circuit from the primitive we have realized it in Even-Mansour scheme [EM97] with two different keys at the beginning and end. With respect to our `SPEEDY` implementations we distinguish between results that are achieved when giving the regular behavioral (or dataflow) description of the cipher to the synthesis tool and those results we have obtained by optimizing the code and instantiating the desired standard cells directly in the HDL code (according to the gate-level descriptions shown in Figures 3 and 4). It is obvious that this optimization has a significant impact on the performance in NanGate libraries, but less of an impact in the commercial technologies. In order to force the synthesizer to use our suggested gate-level structures for `MC` and `SB` we

Table 10: Area consumption of fully-unrolled encryption-only circuits of different cryptographic primitives when synthesized for minimum latency.

	Area [GE]					
	Commercial Foundry				NanGate OCL	
Cipher	90 nm LP	65 nm LP	40 nm LP	28 nm HPC	45 nm	15 nm
`Gimli E-M`	72644.00	82781.00	63100.50	144036.33	52038.67	57551.25
$\texttt{MANTIS}_6$	21045.75	23264.50	20448.25	36073.33	12660.67	15954.00
$\texttt{MANTIS}_7$	23229.25	26385.75	23192.50	43220.33	14225.67	17522.50
$\texttt{MANTIS}_8$	26365.75	30316.75	25429.75	50793.00	15663.33	19707.50
`Midori`	18678.50	21964.00	17562.25	41450.67	10675.33	13927.25
`Orthros`	49639.75	61657.00	44715.75	74384.67	31317.33	39165.00
`PRINCE`	16244.25	19877.75	17177.00	38145.33	9873.33	13291.00
`PRINCEv2`	17661.25	18798.25	16556.50	33470.33	10332.00	13069.50
$\texttt{QARMA}_5$-64-σ_0	19590.75	21706.75	20255.00	31703.00	11824.67	14880.75
$\texttt{QARMA}_6$-64-σ_0	22624.25	25349.50	22689.00	38813.67	14165.67	17621.75
$\texttt{QARMA}_7$-64-σ_0	25614.00	29323.00	24656.25	40494.33	15769.33	19770.25
$\texttt{QARMA}_8$-64-σ_0	28813.75	32780.75	28262.75	47952.33	17908.00	22074.00
$\texttt{QARMA}_5$-64-σ_1	20264.75	23753.00	20202.25	34302.00	12350.33	15588.75
$\texttt{QARMA}_6$-64-σ_1	23162.25	26941.25	23333.75	45419.00	15066.00	18164.00
$\texttt{QARMA}_7$-64-σ_1	26563.75	31495.00	27059.50	52108.00	16641.00	20670.25
$\texttt{QARMA}_8$-64-σ_1	30534.50	35787.75	29116.50	54967.00	18963.67	22761.75
`SPEEDY-5-192`	47364.00	53856.00	47528.50	74467.00	27903.33	34649.00
`SPEEDY-6-192`	57322.00	64438.25	56816.00	88932.00	34085.00	41443.25
`SPEEDY-7-192`	68370.00	75273.00	65422.00	95235.67	39853.33	48727.75
`SPEEDY-5-192` *	49902.00	58796.25	55846.75	80313.33	29839.00	38075.25
`SPEEDY-6-192` *	59688.00	70653.00	66553.00	98950.00	36523.33	46266.50
`SPEEDY-7-192` *	73397.75	84745.00	77519.75	111754.33	42813.33	54193.25

* = Optimized HDL code with direct instantiation of library cells based on Figures 3 and 4.

set a `size-only` attribute on the relevant cells in *Synopsys Design Compiler* before the first `compile_ultra` command. The synthesizer then only scales the drive strengths of these cells. In a next step three `compile_ultra -incremental` commands are executed without `size-only` attribute, so that all optimizations are allowed again. With that technique the highest quality of results is achieved and the majority of manually-instantiated cells still remain unchanged.

It is obvious from Table 9 that `SPEEDY-5-192` and `SPEEDY-6-192` produce the smallest latencies among all implementations. The next fastest primitives are `Orthros` and `PRINCE`/`PRINCEv2`. `Gimli`, performs respectably well given its large state (384 bit) and number of rounds (24). Yet, the claim that it outperforms `PRINCE` by a significant margin, made in [GKD20], is very doubtful considering our results. Please note that for all ciphers except `Midori` we have used hardware implementations written by the original authors of the corresponding papers (`Qameleon` authors for `QARMA`).

Table 10 shows the corresponding area consumption for the fully-unrolled and highly latency constrained circuits. Clearly, `SPEEDY` requires a larger circuit area compared to all other ciphers except `Gimli`. However, this is mainly caused by its 192-bit state (which is larger than for all other ciphers in the table except `Gimli`). In more detail, when multiplying the area of the 64-bit ciphers by 3 (to encrypt 192 bit at once) many of them require a larger area than `SPEEDY-5-192` and all `MANTIS` and `QARMA` instances even exceed the area of `SPEEDY-6-192`. Thus, we believe that for their block widths and the high security and performance levels that the `SPEEDY` instances provide, their area consumption is acceptable. Power consumption figures for all circuits are given in Appendix A, Table 12.

Table 11: Comparison of pre-layout and post-layout latencies in a commercial 65 nm CMOS technology.

	Minimum Latency [ns]		
	65 nm LP		
Cipher	Pre-Layout	Post-Layout	Overhead
Gimli E-M	3.330192	3.902397	17.18 %
$\texttt{MANTIS}_6$	3.405490	3.810253	11.89 %
$\texttt{MANTIS}_7$	3.722473	4.225445	13.51 %
$\texttt{MANTIS}_8$	4.233543	4.785156	13.03 %
Midori	3.582221	4.005088	11.80 %
Orthros	2.678637	3.166256	18.20 %
PRINCE	2.866749	3.236980	12.91 %
PRINCEv2	2.944367	3.324928	12.93 %
$\texttt{QARMA}_5$-64-σ_0	2.920377	3.302898	13.10 %
$\texttt{QARMA}_6$-64-σ_0	3.418600	3.869228	13.18 %
$\texttt{QARMA}_7$-64-σ_0	3.909138	4.432907	13.40 %
$\texttt{QARMA}_8$-64-σ_0	4.396543	5.078354	15.51 %
$\texttt{QARMA}_5$-64-σ_1	3.284252	3.696785	12.56 %
$\texttt{QARMA}_6$-64-σ_1	3.808675	4.294109	12.75 %
$\texttt{QARMA}_7$-64-σ_1	4.371963	4.929371	12.75 %
$\texttt{QARMA}_8$-64-σ_1	4.904521	5.519027	12.53 %
SPEEDY-5-192	2.178075	2.612023	19.92 %
SPEEDY-6-192	2.639186	3.142331	19.06 %
SPEEDY-7-192	3.087257	3.717537	20.42 %
SPEEDY-5-192 *	2.121748	2.572030	21.22 %
SPEEDY-6-192 *	2.573561	3.136378	21.87 %
SPEEDY-7-192 *	3.029217	3.696695	22.03 %

* = Optimized HDL code with direct instantiation of library cells based on Figures 3 and 4.

Because synthesis results disregard the impact of wire capacitances on the latency of hardware circuits, we have exemplarily taken all netlists generated for the 65 nm technology through a Place and Route (PnR) process in order to estimate the post-layout latencies. These are given in comparison to the pre-layout values in Table 11. Naturally, the overhead introduced by the physical layout is greater for the circuits that have a larger area footprint, e.g., Gimli, Orthros and SPEEDY, because connected cells might be wider apart from each other and longer wire lengths are required to connect them (also because metal utilization increases and wires have to be routed on higher, thicker metal layers). However, despite the slightly larger overhead SPEEDY-5-192 and SPEEDY-6-192 are still the fastest encryption primitives after PnR.

7.1 Decryption

For the most part of this work we have ignored the SPEEDY decryption. SPEEDY is primarily designed to offer ultra fast encryption of data with a high level of security. As discussed by the authors of the Orthros low-latency PRF, it is sufficient for many use cases to have a one directional primitive [BIL+21]. Among these use cases are several popular block cipher modes of operation, such as CTR, CMAC and GCM, which all require no decryption routine, as well as applications such as pointer authentication and memory encryption schemes based on Merkle trees [BIL+21]. According to [BIL+21] even a memory encryption scheme applied inside Intel's Software Guard Extensions (SGX) uses adapted variants of GMAC and GCM without requiring the underlying primitive to be invertible. However, since SPEEDY does not lack invertibility like Orthros does, it can also be used in

application scenarios where invertibility and decryption are indeed required, but where it is acceptable that only one direction is extremely efficient. In Appendix B, Table 13 implementation results (latency, area, power) are presented for the SPEEDY decryption. Although it is not nearly as efficient as the encryption, the SPEEDY-5-192 decryption is faster than the Midori encryption and many others (cf. Table 9) and the SPEEDY-6-192 decryption is still faster than the QARMA$_7$-64-σ_1 encryption and a few more (cf. Table 9).

7.2 Code and Reproducibility

A reference software implementation in C and hardware implementations of SPEEDY-r-192 encryption and decryption in VHDL, along with synthesized netlists in NanGate libraries and associated synthesis scripts, are all available in our GitHub repository found here: https://github.com/Chair-for-Security-Engineering/SPEEDY.

8 Conclusion

In this work we have introduced SPEEDY, a family of ultra low-latency block ciphers developed for extremely high execution speed in CMOS hardware and dedicated to semi-custom, i.e., standard-cell-based, integrated circuit design. The primary targets for SPEEDY are security architectures in high-end CPUs which require ultra low-latency encryption, such as secure caches, dedicated hardware extensions, memory encryption, pointer authentication and many more. SPEEDY achieves higher performance than any competitor because of hardware-specific gate- and transistor-level observations that have been exploited in its design to make it extremely performant in CMOS hardware. While SPEEDY can be instantiated with different block and key sizes, the default is 192 bit. Based on our analysis, we are confident that 7 rounds provide full security, while 5 rounds already provide a higher security level than PRINCE or PRINCEv2 for example. Our extensive evaluation of hardware implementations demonstrates that both SPEEDY-5-192 and SPEEDY-6-192 are faster than any proposed version of PRINCE, PRINCEv2, MANTIS, QARMA, Midori, Gimli and Orthros. Thus, SPEEDY is a significant upgrade over the state of the art for any application where area and energy are secondary design goals while high performance is the number one priority.

Acknowledgments

The work described in this paper has been supported in part by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy - EXC 2092 CASA - 390781972 and through the project 406956718 "SymmetriC CiphEr design with inherent phySical Security (SuCCESS)". Besides, S. Rasoolzadeh is supported by the Netherlands Organisation for Scientific Research (NWO) under TOP grant TOP1.18.002 SCALAR.

References

[ABP+18] Victor Arribas, Begül Bilgin, George Petrides, Svetla Nikova, and Vincent Rijmen. Rhythmic keccak: SCA security and low latency in HW. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2018(1):269–290, 2018.

[Ava17] Roberto Avanzi. The QARMA block cipher family. almost MDS matrices over rings with zero divisors, nearly symmetric even-mansour constructions with non-involutory central rounds, and search heuristics for low-latency s-boxes. *IACR Trans. Symmetric Cryptol.*, 2017(1):4–44, 2017.

[BBI+15] Subhadeep Banik, Andrey Bogdanov, Takanori Isobe, Kyoji Shibutani, Harunaga Hiwatari, Toru Akishita, and Francesco Regazzoni. Midori: A block cipher for low energy. In Tetsu Iwata and Jung Hee Cheon, editors, *Advances in Cryptology - ASIACRYPT 2015 - 21st International Conference on the Theory and Application of Cryptology and Information Security, Auckland, New Zealand, November 29 - December 3, 2015, Proceedings, Part II*, volume 9453 of *Lecture Notes in Computer Science*, pages 411–436. Springer, 2015.

[BCG+12] Julia Borghoff, Anne Canteaut, Tim Güneysu, Elif Bilge Kavun, Miroslav Knezevic, Lars R. Knudsen, Gregor Leander, Ventzislav Nikov, Christof Paar, Christian Rechberger, Peter Rombouts, Søren S. Thomsen, and Tolga Yalçin. PRINCE - A low-latency block cipher for pervasive computing applications - extended abstract. In Xiaoyun Wang and Kazue Sako, editors, *Advances in Cryptology - ASIACRYPT 2012 - 18th International Conference on the Theory and Application of Cryptology and Information Security, Beijing, China, December 2-6, 2012. Proceedings*, volume 7658 of *Lecture Notes in Computer Science*, pages 208–225. Springer, 2012.

[BEK+20] Dušan Božilov, Maria Eichlseder, Miroslav Knezevic, Baptiste Lambin, Gregor Leander, Thorben Moos, Ventzislav Nikov, Shahram Rasoolzadeh, Yosuke Todo, and Friedrich Wiemer. Princev2 - more security for (almost) no overhead. In *Selected Areas in Cryptography - SAC 2020*, Lecture Notes in Computer Science, 2020.

[BFP19] Joan Boyar, Magnus Gausdal Find, and René Peralta. Small low-depth circuits for cryptographic applications. *Cryptogr. Commun.*, 11(1):109–127, 2019.

[BIL+21] Subhadeep Banik, Takanori Isobe, Fukang Liu, Kazuhiko Minematsu, and Kosei Sakamoto. Orthros: A low-latency PRF. *IACR Trans. Symmetric Cryptol.*, 2021(1):37–77, 2021.

[BJK+16] Christof Beierle, Jérémy Jean, Stefan Kölbl, Gregor Leander, Amir Moradi, Thomas Peyrin, Yu Sasaki, Pascal Sasdrich, and Siang Meng Sim. The SKINNY family of block ciphers and its low-latency variant MANTIS. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part II*, volume 9815 of *Lecture Notes in Computer Science*, pages 123–153. Springer, 2016.

[BKL+17] Daniel J. Bernstein, Stefan Kölbl, Stefan Lucks, Pedro Maat Costa Massolino, Florian Mendel, Kashif Nawaz, Tobias Schneider, Peter Schwabe, François-Xavier Standaert, Yosuke Todo, and Benoît Viguier. Gimli : A cross-platform permutation. 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 299–320. Springer, 2017.

[BKN19] Dusan Bozilov, Miroslav Knezevic, and Ventzislav Nikov. Optimized threshold implementations: Minimizing the latency of secure cryptographic accelerators. In Sonia Belaïd and Tim Güneysu, editors, *Smart Card Research and Advanced Applications - 18th International Conference, CARDIS 2019, Prague, Czech Republic, November 11-13, 2019, Revised Selected Papers*, volume 11833 of *Lecture Notes in Computer Science*, pages 20–39. Springer, 2019.

[BMD+20] Begül Bilgin, Lauren De Meyer, Sébastien Duval, Itamar Levi, and François-Xavier Standaert. Low AND depth and efficient inverses: a guide on s-boxes

for low-latency masking. *IACR Trans. Symmetric Cryptol.*, 2020(1):144–184, 2020.

[DEMS19] Christoph Dobraunig, Maria Eichlseder, Florian Mendel, and Martin Schläffer. Ascon v1.2 submission to nist. https://csrc.nist.gov/CSRC/media/Projects/lightweight-cryptography/documents/round-2/spec-doc-rnd2/ascon-spec-round2.pdf, 2019. Accessed: 2021-07-02.

[DXS19] Shuwen Deng, Wenjie Xiong, and Jakub Szefer. Analysis of secure caches using a three-step model for timing-based attacks. *J. Hardware and Systems Security*, 3(4):397–425, 2019.

[EM97] Shimon Even and Yishay Mansour. A construction of a cipher from a single pseudorandom permutation. *J. Cryptol.*, 10(3):151–162, 1997.

[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.

[GKD20] Santosh Ghosh, Michael E. Kounavis, and Sergej Deutsch. Gimli encryption in 715.9 psec. *IACR Cryptol. ePrint Arch.*, 2020:336, 2020.

[KHF+19] Paul Kocher, Jann Horn, Anders Fogh, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz, and Yuval Yarom. Spectre attacks: Exploiting speculative execution. In *2019 IEEE Symposium on Security and Privacy, SP 2019, San Francisco, CA, USA, May 19-23, 2019*, pages 1–19. IEEE, 2019.

[KNR12] Miroslav Knezevic, Ventzislav Nikov, and Peter Rombouts. Low-latency encryption - is "lightweight = light + wait"? In Emmanuel Prouff and Patrick Schaumont, editors, *Cryptographic Hardware and Embedded Systems - CHES 2012 - 14th International Workshop, Leuven, Belgium, September 9-12, 2012. Proceedings*, volume 7428 of *Lecture Notes in Computer Science*, pages 426–446. Springer, 2012.

[LKO+21] Moritz Lipp, Andreas Kogler, David Oswald, Michael Schwarz, Catherine Easdon, Claudio Canella, and Daniel Gruss. PLATYPUS: Software-based Power Side-Channel Attacks on x86. In *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2021.

[LP07] Gregor Leander and Axel Poschmann. On the classification of 4 bit s-boxes. In Claude Carlet and Berk Sunar, editors, *Arithmetic of Finite Fields, First International Workshop, WAIFI 2007, Madrid, Spain, June 21-22, 2007, Proceedings*, volume 4547 of *Lecture Notes in Computer Science*, pages 159–176. Springer, 2007.

[LSG+18] Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Anders Fogh, Jann Horn, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, and Mike Hamburg. Meltdown: Reading kernel memory from user space. In William Enck and Adrienne Porter Felt, editors, *27th USENIX Security Symposium, USENIX Security 2018, Baltimore, MD, USA, August 15-17, 2018*, pages 973–990. USENIX Association, 2018.

[LSL+19] Shun Li, Siwei Sun, Chaoyun Li, Zihao Wei, and Lei Hu. Constructing low-latency involutory MDS matrices with lightweight circuits. *IACR Trans. Symmetric Cryptol.*, 2019(1):84–117, 2019.

[Moo65] Gordon E. Moore. Cramming more components onto integrated circuits. *Electronics*, 38(8), April 1965.

[Moo20] Thorben Moos. Unrolled cryptography on silicon A physical security analysis. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2020(4):416–442, 2020.

[MS16] Amir Moradi and Tobias Schneider. Side-channel analysis protection and low-latency in action - - case study of PRINCE and midori -. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *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*, volume 10031 of *Lecture Notes in Computer Science*, pages 517–547, 2016.

[oST79] National Institute of Standards and Technology. Fips-46: Data encryption standard (des). http://csrc.nist.gov/publications/fips/fips46-3/fips46-3.pdf, 1979. Accessed: 2021-07-02.

[oST01] National Institute of Standards and Technology. Fips-197: Advanced encryption standard (aes). https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf, 2001. Accessed: 2021-07-02.

[Qur18] Moinuddin K. Qureshi. CEASER: mitigating conflict-based cache attacks via encrypted-address and remapping. In *51st Annual IEEE/ACM International Symposium on Microarchitecture, MICRO 2018, Fukuoka, Japan, October 20-24, 2018*, pages 775–787. IEEE Computer Society, 2018.

[RCN04] Jan M. Rabaey, Anantha Chandrakasan, and Borivoje Nikolic. *Digital integrated circuits- A design perspective.* Prentice Hall, 2ed edition, 2004.

[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.

[WUG+19] Mario Werner, Thomas Unterluggauer, Lukas Giner, Michael Schwarz, Daniel Gruss, and Stefan Mangard. Scattercache: Thwarting cache attacks via cache set randomization. In Nadia Heninger and Patrick Traynor, editors, *28th USENIX Security Symposium, USENIX Security 2019, Santa Clara, CA, USA, August 14-16, 2019*, pages 675–692. USENIX Association, 2019.

A Power Consumption

Table 12: Estimated power consumption of fully-unrolled encryption-only circuits of different cryptographic primitives when synthesized for minimum latency. Estimated for 100 MHz operation.

	Power [mW]					
	Commercial Foundry				NanGate OCL	
Cipher	90 nm LP	65 nm LP	40 nm LP	28 nm HPC	45 nm	15 nm
Gimli E-M	16.3489	12.4244	4.1035	8.5614	9.4797	2.7762
$\texttt{MANTIS}_6$	0.2848	0.2108	0.0889	0.3755	0.3680	0.2101
$\texttt{MANTIS}_7$	0.3140	0.2409	0.0986	0.4509	0.4107	0.2318
$\texttt{MANTIS}_8$	0.3503	0.2806	0.1072	0.5269	0.4479	0.2605
Midori	0.2652	0.2104	0.0798	0.4512	0.3131	0.1848
Orthros	0.6626	0.5814	0.1935	0.7978	0.8711	0.4959
PRINCE	0.2162	0.1856	0.0756	0.4079	0.2930	0.1759
PRINCEv2	0.2390	0.1827	0.0721	0.3629	0.3041	0.1708
$\texttt{QARMA}_5$-64-σ_0	0.2652	0.2044	0.0867	0.3285	0.3448	0.1997
$\texttt{QARMA}_6$-64-σ_0	0.2993	0.2364	0.0973	0.3973	0.4099	0.2332
$\texttt{QARMA}_7$-64-σ_0	0.3367	0.2640	0.1054	0.4087	0.4529	0.2614
$\texttt{QARMA}_8$-64-σ_0	0.3846	0.2964	0.1205	0.4935	0.5121	0.2896
$\texttt{QARMA}_5$-64-σ_1	0.2669	0.2187	0.0872	0.3672	0.3607	0.2059
$\texttt{QARMA}_6$-64-σ_1	0.3052	0.2443	0.1004	0.4879	0.4350	0.2385
$\texttt{QARMA}_7$-64-σ_1	0.3544	0.2795	0.1161	0.5599	0.4769	0.2700
$\texttt{QARMA}_8$-64-σ_1	0.3903	0.3246	0.1263	0.5906	0.5418	0.2946
SPEEDY-5-192	11.6227	7.9766	3.0922	3.9246	4.9508	1.7998
SPEEDY-6-192	14.2678	9.7228	3.7569	4.7595	6.1494	2.1764
SPEEDY-7-192	17.2552	11.5149	4.4061	5.1270	7.2578	2.5978
SPEEDY-5-192 *	11.7005	8.6807	3.6014	5.8412	5.3485	2.0160
SPEEDY-6-192 *	14.2010	10.6287	4.3671	5.1269	6.6413	2.4959
SPEEDY-7-192 *	17.8889	12.9823	5.1331	5.8412	7.8866	2.9508

* = Optimized HDL code with direct instantiation of library cells based on Figures 3 and 4.

B SPEEDY Decryption Implementation Results

Table 13: Estimated latency, area, and power consumption of the SPEEDY decryption routine.

	Minimum Latency [ns]					
	Commercial Foundry				NanGate OCL	
Cipher	90 nm LP	65 nm LP	40 nm LP	28 nm HPC	45 nm	15 nm
SPEEDY-5-192	4.827471	3.469787	2.953934	1.387975	5.088359	0.471568
SPEEDY-6-192	5.845453	4.197634	3.586378	1.680402	6.174353	0.572912
SPEEDY-7-192	6.887968	4.937893	4.240692	1.987920	7.259925	0.672681
	Area [GE]					
SPEEDY-5-192	101401.50	118295.50	107298.50	123458.67	70771.33	86302.50
SPEEDY-6-192	120336.75	138823.50	127010.00	146688.00	83632.67	102160.50
SPEEDY-7-192	138292.50	161802.50	142642.25	163059.67	97923.33	117827.25
	Power [mW]					
SPEEDY-5-192	21.6051	15.7708	6.4204	5.7405	11.6600	4.1493
SPEEDY-6-192	26.2426	18.7986	7.7360	6.9424	14.0370	4.9956
SPEEDY-7-192	30.3541	22.0906	8.6553	7.7193	16.5390	5.8020

C Test Vectors for SPEEDY-r-192

SPEEDY-5-192

K	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
P	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
C	E0	D5	6F	BD	95	56	A8	71	CA	49	35	7A	82	2D	04	81	A8	50	2D	DD	16	FE	CE	0F
K	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
P	01	23	45	67	89	AB	CD	EF	01	23	45	67	89	AB	CD	EF	01	23	45	67	89	AB	CD	EF
C	12	3A	5D	7A	D4	5D	E4	4A	27	64	0B	EF	01	F4	8D	42	01	7C	FA	D0	F2	22	3C	3C
K	01	23	45	67	89	AB	CD	EF	01	23	45	67	89	AB	CD	EF	01	23	45	67	89	AB	CD	EF
P	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
C	FC	FB	8E	9C	23	0A	07	81	B0	63	30	76	FD	62	BF	7D	CE	F4	98	BA	2C	2B	29	6C
K	76	4C	4F	62	54	E1	BF	F2	08	E9	58	62	42	8F	AE	D0	15	84	F4	20	7A	7E	84	77
P	A1	3A	63	24	51	07	0E	43	82	A2	7F	26	A4	06	82	F3	FE	9F	F6	80	28	D2	4F	DB
C	01	DA	25	A9	3D	1C	FC	5E	4C	0B	74	F6	77	EB	74	6C	28	1A	26	01	93	B7	75	5A

SPEEDY-6-192

K	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
P	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
C	A6	D5	18	A2	E5	73	75	15	15	93	11	0A	16	1E	D7	C6	27	8A	BC	D0	31	CB	E8	6C
K	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
P	01	23	45	67	89	AB	CD	EF	01	23	45	67	89	AB	CD	EF	01	23	45	67	89	AB	CD	EF
C	CB	44	11	34	1F	FF	B3	00	03	00	1A	8C	1F	06	FE	D8	7F	F6	89	C5	2D	1E	AB	65
K	01	23	45	67	89	AB	CD	EF	01	23	45	67	89	AB	CD	EF	01	23	45	67	89	AB	CD	EF
P	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
C	4B	F4	3B	6A	64	8E	81	6A	EF	4F	C9	88	A9	4C	76	7F	A8	36	BA	25	A8	D2	A3	EF
K	76	4C	4F	62	54	E1	BF	F2	08	E9	58	62	42	8F	AE	D0	15	84	F4	20	7A	7E	84	77
P	A1	3A	63	24	51	07	0E	43	82	A2	7F	26	A4	06	82	F3	FE	9F	F6	80	28	D2	4F	DB
C	88	BF	D3	DC	14	0F	38	BC	53	A6	66	87	F5	30	78	60	56	0E	BE	C4	11	00	66	2D

SPEEDY-7-192

K	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
P	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
C	24	7D	30	80	D2	63	F7	4C	B0	3D	DE	6E	57	5C	68	EE	68	EE	E9	57	E1	C2	9C	50
K	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
P	01	23	45	67	89	AB	CD	EF	01	23	45	67	89	AB	CD	EF	01	23	45	67	89	AB	CD	EF
C	B4	8F	32	16	AB	33	AE	01	99	14	2F	6A	07	43	E8	48	1B	FC	37	62	5C	BB	DC	4F
K	01	23	45	67	89	AB	CD	EF	01	23	45	67	89	AB	CD	EF	01	23	45	67	89	AB	CD	EF
P	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
C	55	65	08	92	98	E4	C1	34	CE	03	12	B2	7E	75	BA	21	A6	8C	0B	4F	46	33	7F	2D
K	76	4C	4F	62	54	E1	BF	F2	08	E9	58	62	42	8F	AE	D0	15	84	F4	20	7A	7E	84	77
P	A1	3A	63	24	51	07	0E	43	82	A2	7F	26	A4	06	82	F3	FE	9F	F6	80	28	D2	4F	DB
C	ED	3D	0E	A1	1C	42	7B	D3	25	70	DF	41	C6	FD	66	EB	BF	49	16	E7	60	ED	09	43