Hold Your Breath, PRIMATEs Are Lightweight

Danilo Šijačić[1], Andreas B. Kidmose[2,4], Bohan Yang[1],
Subhadeep Banik[3], Begül Bilgin[1], Andrey Bogdanov[2,5], Ingrid Verbauwhede[1]

[1]ESAT/COSIC, KU Leuven and iMinds, Belgium
{name.surname@esat.kuleuven.be}
[2]Technical University of Denmark, Denmark
[4]{s113242@student.dtu.dk}, [5]{anbog@dtu.dk}
[3]Temasek Labs, Nanyang Technological University, Singapore
{bsubhadeep@ntu.edu.sg}

Abstract. This work provides the first hardware implementations of PRIMATEs family of authenticated encryption algorithms. PRIMATEs are designed to be lightweight in hardware, hence we focus on designs for constrained devices. We provide several serial implementations, smallest of which requires only 1.2 kGE. Additionally, we present a variety of threshold implementations that range from 4.7 kGE to 10.3 kGE.
The second part of this work presents a design of a lightweight PRIMATEs coprocessor. It is designed to conform versatile use of the core permutation, which allows implementation of the entire PRIMATEs family, with small differences in hardware. We implement HANUMAN-80 coprocessor, adapted for a 16-bit microcontroller from the Texas Instruments MSP430 family of microcontrollers. The entire HANUMAN-80 coprocessor is tested on a Spartan-6 (XC6SLX45) development board, where it occupies 72 slices (1.06% of available resources). ASIC synthesis yields a 2 kGE implementation using 90 nm library, achieves 33 kbits/sec throughput at 100 kHz operating frequency. It dissipates 0.53 μW of power on average, resulting in energy consumption of 15.60 pJ/bit.

Keywords: PRIMATEs, CAESAR, Authenticated Encryption, Hardware Implementation, Threshold Implementation, Lightweight

1 Introduction

Motivation Emerging Internet of Things (IOT) technologies require a swarm of lightweight devices scattered in our surroundings. Various sensors, actuators, or authenticators, have to provide reliable, uninterrupted service, while protecting users' privacy and data confidentiality through encryption, and data authenticity and integrity through authentication. Since adversaries may easily gain access to these devices, protection against physical attacks must be taken into account. Moreover, all of this has to be achieved at a very low price in terms of chip area, power, and energy consumption. While exact constraints vary between different kinds of these devices, we believe that passively powered

Radio Frequency IDentification (RFID) tags—which are used for identification, access management and shipment tracking, handling payments; and are great assets in aiding medical treatment—present the worst-case in terms of area and power-budget limitations. Even though the notion of lightweightness seems subjective and application-bound, statements from industry [28] and research community [6, 13, 14] agree that the area footprint of the cryptographic algorithm must not exceed 2000 two-input NAND-gates equivalent (GE) in the selected library. Having at least 12 kbits/sec throughput at the operating frequency of 100 kHz is the only bound used by researchers [6, 14] whereas industry requires having 1 bit/cycle [28]. Unfortunately, there are no widely accepted upper and lower bounds for low power and high throughput respectively, even though the discussions suggest that it is of interest [22, 28]. Lastly, in terms of average power usage industry suggests between 1 and 10 μW/MHz, with peaks between 3 and 30 μW, respectively.

Many standardized cryptographic algorithms, such as the Advanced Encryption Standard (AES) [23] of which the smallest implementation requires 2400 GE, are unfit for the lightweight area of application especially when they are wrapped with a mode of operation to provide both encryption and authentication. This results in the increasing number of stream ciphers [20], block ciphers [6, 14, 17, 19, 30] and hash functions [5, 12, 18] together with the recent standardization of the lightweight block cipher PRESENT which can be implemented using 1000 GE [27].

Traditionally, the security goals of a system are achieved using generic compositions of cryptographic primitives providing only authentication or encryption often resulting in exploitable weaknesses [2, 7]. This problem of authenticated encryption is formalized in [26], where a set of possible generic schemes named *authenticated-encryption with associated data* (AEAD) is discussed. Advancement in lightweight cipher design, and the formalization of the problem, followed by discussions in research community lead to the Competition for Authenticated Encryption: Security, Applicability, and Robustness (CAESAR) [1] which is running since 2014 with the goal of selecting a portfolio of AEAD ciphers. Currently 29 second-round candidates are being analyzed for security, software and hardware performance. Out of several candidates that claim to be lightweight, PRIMATEs [3] family grasped our attention for their claims of versatile usability, and efficiency in hardware.

Related Work PRIMATEs [3] is a family of single-pass nonce-based AEAD schemes. All members of PRIMATEs are designed for constrained hardware. They differ slightly to provide trade-offs between security and performance. High level of granularity allows PRIMATEs to find application in a number of lightweight scenarios. Authors claim that PRIMATEs can efficiently be protected against Side-Channel Analysis (SCA), especially Differential Power Analysis (DPA) [25]. Namely, non-linear part of PRIMATEs has low algebraic degree, which results in efficient Threshold Implementations (TI) [24]. TI provides provable security against DPA for symmetric key algorithms (e.g., [11]).

Until now, only a reference software implementation of PRIMATEs is provided. The claims on efficiency in hardware, of unprotected and SCA-resistant implementations, are still to be examined.

Contribution We challenge lightweightness and versatility claims by focusing on low-end designs. Namely, we design several lightweight architectures in order to analyze area, throughput and energy trade-offs. We discuss the overall performance, and how our implementations can be used in practice. Additionally, we provide a variety of TI to examine efficiency of SCA resistant implementations. We discuss the overall performance, and how our implementations can be used in practice. Furthermore, in order to accommodate practical lightweight scenarios, we present a PRIMATEs interface, designed to minimize area and latency overhead. Lastly, we design and implement a PRIMATEs coprocessor based on the aforementioned interface.

2 Preliminaries

We inherit the notation suggested by the PRIMATEs designers [3]. Namely, calligraphic, capital and small letters represent a set, an element of the set and the bit size of the element respectively, i.e. $\mathcal{X} := \{0,1\}^x$, $X \in \mathcal{X}$ and $|X| = x$. Let $X \in \{0,1\}^x$ and $Y \in \{0,1\}^y$, then $X||Y \in \{0,1\}^{x+y}$ represents the concatenation of X and Y.

2.1 AEAD Scheme

An AEAD scheme is defined by the three tuple $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ as follows:
Key space. $\mathcal{K}$ is a non-empty set of k-bit strings, i.e. $\mathcal{K} := \{0,1\}^k$.
Encryption. $\mathcal{E}$ is a deterministic algorithm which returns a pair of strings $(C,T) = \mathcal{E}_K(N,A,M)$; where: the secret key $K \in \mathcal{K}$, the public nonce $N \in \mathcal{N} := \{0,1\}^n$, the public associated data $A \in \mathcal{A} := \{0,1\}^*$, the message $M \in \mathcal{M} := \{0,1\}^*$, the ciphertext $C \in \mathcal{C} := \{0,1\}^*$, and the tag $T \in \mathcal{T} := \{0,1\}^t$. The algorithm must work even if $|M| = 0$ and/or $|A| = 0$.
Decryption. $\mathcal{D}_K(N,A,C,T)$ is a deterministic algorithm that generates the pair (M,T'). $\mathcal{D}$ returns the value M to user if $T = T'$. Otherwise $\mathcal{D}$ returns a unique symbol $\perp$. It is possible to release the message even if the tags do not match if the AEAD scheme follows certain properties [4].

2.2 PRIMATEs

PRIMATEs is a family of three modes of operation named APE, HANUMAN and GIBBON [3] with sponge-like construction [8]. Namely, they rely on a permutation which operates on a binary state $B \in \{0,1\}^b$, comprised of the *rate* $R \in \{0,1\}^r$, and the *capacity* $C \in \{0,1\}^c$ (i.e. $B = R\|C$).

Each mode of operation may provide two levels of security. The security level $s \in \{80,120\}$ defines several parameters, as described in Table 1. The input block size is $r = 40$ bits independent of the mode or the security level.

PRIMATE. PRIMATEs family is based on a set of permutations, called PRIMATE permutations. Depending on the security level s, two subsets are distinguished, PRIMATE-80 and PRIMATE-120. PRIMATE-80 (resp. PRIMATE-120) is based on P80 (P120), a 200-bit (resp. 280-bit) core permutation. Permutations are designed as substitution-permutation networks (SPNs).

In both cases states are divided into 5-bit *elements*, with big-endian encoding. Elements themselves are arranged into matrices with 5 (resp. 7) 8-element rows for PRIMATE-80 (resp. PRIMATE-120). The element in the i^{th} row and j^{th} column of this matrix is denoted by $a_{i,j}$ where $i \in \{0, \ldots, 4\}$ (resp. $i \in \{0, \ldots, 6\}$) and $j \in \{0, \ldots, 7\}$. The first row $a_{0,*}$ in the state matrix contains the rate of the state, and will henceforth be referred to as the rate row. P80, and P120 are calculated using a sequence of four transformations described as follows:

1. *SubElements* (SE) is the only non-linear transformation. It consists of an element-wise permutation $X \rightarrow S(X) : \{0,1\}^5 \rightarrow \{0,1\}^5$ (S-Box) applied to each element of a state.
2. *ShiftRows* (SR) performs cyclical shifts of each row for a different number of elements. Row i is shifted left by $s_i = \{0, 1, 2, 4, 7\}$ in P80, or by $s_i = \{0, 1, 2, 3, 4, 5, 7\}$ in P120.
3. *MixColumns* (MC) operates on a state column at a time. It is a left-hand multiplication by a 5×5 (7×7) Maximum Distance Separable (MDS) matrix [3]. The matrices are chosen in a way that allows recursive calculation of a smaller matrix five (resp. seven) times.
4. *ConstantAddition* (CA) modifies a single state element $a_{1,1}$ by bitwise XORing a 5-bit constant in each round.

Round constants are generated by a 5-bit Fibonacci LFSR [3]. Varying on the sequence of values sampled from this LFSR and the number of rounds, four permutations p_1, p_2, p_3, and p_4 are derived from the core permutation (either P80 or P120), as shown in Table 2.

PRIMATE Modes of Operation. All modes of operation are generic constructs, designed based on Sponge [8] methodology principles with slight differences in input output behavior, parameter size and used permutations. Table 1 gives an overview for the latter two differences. Please refer to [3] for details on the former difference. We only emphasize the fact that decryptions of HANUMAN and GIBBON do not require the inverse transformation of PRIMATE whereas APE does.

Table 1. PRIMATEs modes of operation.

PRIMATEs	APE-s	HANUMAN-s	GIBBON-s
k	$2s$	s	s
t	$2s$	s	s
n	s	s	s
PRIMATE	p_1	p_1, p_4	p_1, p_2, p_3

Table 2. PRIMATE permutations.

PRIMATE	p_1	p_2	p_3	p_4
#of rounds of P-s	12	6	6	12
Init. val. of the LFSR	1	24	30	24

3 Implementations of PRIMATE

Following the design rationale of the PRIMATEs family, we focus on hardware implementations for heavily constrained devices. We abstain from using power-saving techniques (e.g., clock gating). Instead we perform architectural optimizations that lead to reduced area and power consumption. Lastly, we strive towards the 12 kbit/sec throughput at the operating frequency of 100 kHz, discussed in Section 1, as the performance criterion.

As in all Sponge-based designs, the majority of implementation cost of PRIMATEs comes from core permutations. Therefore, we investigate several ways to serialize P80 and P120. Additionally, we provide round-based versions of both core permutations.

Lastly, in order to benefit from the granulated nature of the PRIMATE family we have fragmented our implementations into several hierarchical levels, thus creating a generic serial-implementation strategy independent of the permutation design. Therefore, we present the design of the control logic required for generation of p_1, p_2, p_3, and p_4 separately from the core permutations.

3.1 PRIMATE Permutations' Control

The 5-bit Fibonacci LFSR used for CA transformation is one of the lightweight features predicted by design. Firstly, depending on the selected p_i, rounds can be decoded from the values of the LFSR, thereby alleviating the need for additional counters. Secondly, p_i's defer from one another only by the sequence of Fibonacci constants. Therefore, for each of the modes of operations, regardless of the security level, control module ModeCtrl is realized using simple hardware.

Permutation p_i starts by loading a corresponding constant of the Fibonacci sequence into the LFSR. After each round—underlying permutation core must provide a RndDone signal—LFSR progresses along its sequence. Output of the LFSR is used as a round constant RCon for the underlying permutation core. Lastly, small 5-bit decoders are used to detect rounds of interest (e.g., first, last) for the control of upper layers of logic.

3.2 Core Permutations P80 and P120

In combination with the control module from Section 5, any of the implementations from this section can be used to provide encryption and decryption of HANUMAN and GIBBON as well as APE encryption. Since APE requires inverse of p_1 we have abstained from implementing this functionality. Justification for this is twofold: from the area-performance-power perspective implementation of p_1 is negligibly different from its inverse; from the usability perspective it is likely that heavily constrained devices perform only encryption, while the decryption is performed in the backend. Lastly, note that these implementations include only the functionality of the core permutation, i.e. they allow computation of only one round. Depending on the way the core is used, different overhead will be introduced in the design (e.g., a feedback multiplexer). Nevertheless, we

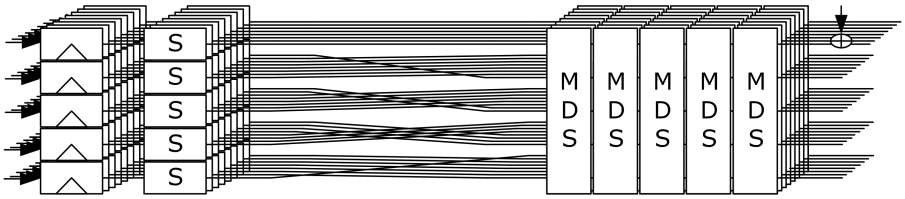

Fig. 1. Round-based architecture of the core permutation, P80-1.

find this sort of results useful, for the various use-cases that may be anticipated for these architectures. More discussion on this topic is given in Section 5.

Round-Based Implementations of P80 and P120. Figure 1, where each line represents a 5-bit element value, depicts the round-based architecture of P80, called P80-1. S-Boxes, MDS matrix multiplications, and constant addition, are implemented as combinational networks. SR transformation can be realized by rewiring of rows, hence it is free in hardware. P120 version (P120-1) is obtained using the same design approach. Lastly, these implementations include a state-sized register—for fair comparison.

Serial Implementations of P80 and P120. Due to the relatively large state, and combinatorial logic designed to be efficient in hardware, major cost of a serial implementation of P80 and P120 comes from the register file used to store the state. Consequently, we aim to minimize the number of multiplexed inputs to every bit of the State Register File (SRF), and to avoid additional registers in the design. Therefore, we abstain from using additional multiplexers for controlling data flow, and design the SRF as a column-wise FIFO register. Hence, all serial implementations of P80 feature a 25-bit data path, while P120 implementations use a 35-bit data path; which correspond to the number of bits in a column of the state matrix.

Lastly, each of the permutation cores requires a dedicated Finite State Machine (FSM) for controlling the data flow, in addition to the ModeCtrl module used to iterate rounds of p_i.

P80-9 *and* P120-9. Figure 2 depicts a 9 clock cycle serial implementation (P80-9). Here, the SRF has only two modes of operation MC, and SR. When MC is active SRF is configured as a 25-bit FIFO register which feeds the data into the combinational network at its output. This mode is used for data input, as well. SR mode is always active during the first cycle of computation, during which it rewires the SRF to perform the SR transformation. P120 version (P120-9) is obtained using the same design approach.

P80-41 *and* P120-57. Based on the 9 clock cycle approach, we present another serial version by serializing the MC step. Namely, 5 matrix multiplications

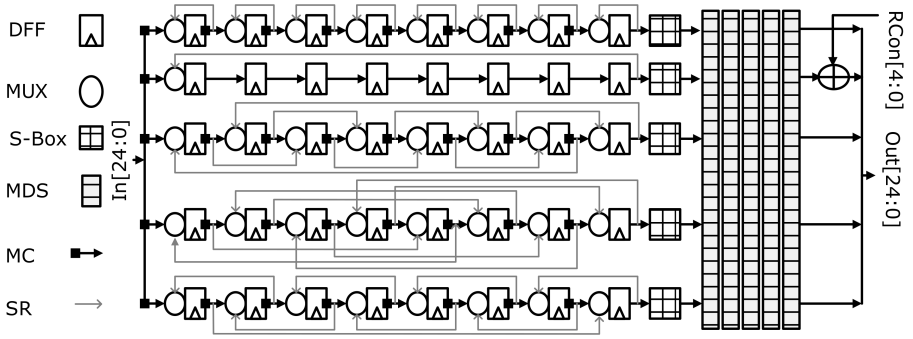

Fig. 2. Data path architecture of the P80-9 permutation core.

that are required for MC transformation of PRIMATE-80, are now performed in 5 clock cycles using 5 times less hardware. The same concept applies for PRIMATE-120 with the difference that factor of 7 applies instead of 5 (since the state of PRIMATE-120 has 7 rows). Therefore instead of 8 additional cycles 5×8 (for PRIMATE-80), or 7×8 (for PRIMATE-120) are required to perform this computation. SR transformation remains performed in a single clock cycle as before.

P80-16 *and* P120-16. Figure 3 depicts a 16 clock cycle serial implementation (P80-16). Instead of serializing MC transformation, we serialize the SR transformation. Namely, SR is performed by shifting the position of the column-wise input by the number of shifts prescribed by the SR operation for each row of SRF. After 8 clock cycles of shifting in this manner, SRF is reconfigured to perform SE and MC for another 8 clock cycles, as done previously. This way a number of 2-to-1 multiplexers around the SRF is traded for additional latency of 7 clock cycles, resulting in minimal multiplexer overhead.

On the downside, this core can not preserve state between two consecutive rounds. Namely, in both configurations SRF is written in 8 clock cycles, using 2 different patterns: regular column-wise shift (MC), and columns-wise shift with offset (SR). Consequently, if the output of one round is sequentially looped back in MC mode (as it would be done in P80-9), a number of elements in a row equal to the row offset is overwritten. Therefore, feedback path for each row should be delayed for the number of clock cycles equal to the row offset. In hardware this delay maps to introducing additional flip-flops. We estimate that the cost of this storage, and corresponding control and glue logic, increases the size of this implementation beyond feasible. P120 version (P120-16) is obtained using the same design approach.

P80-95 *and* P120-127. Lastly, we implemented single S-Box versions. These two implementations have fully serialized MC operations (requiring an additional cycle to load new column), and require 7 clock cycles to perform SR operation. Since 5-bit PRIMATEs S-Boxes are small (30–40 GE), we believe

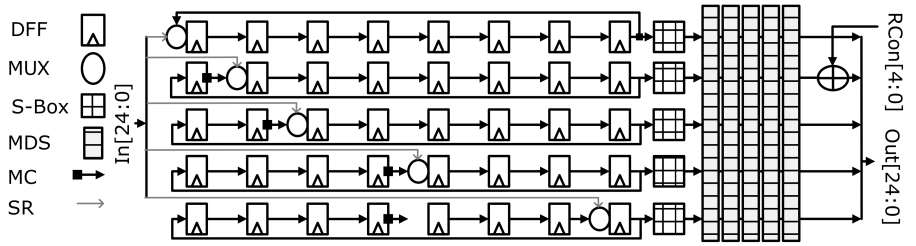

Fig. 3. Data path architecture of the P80-16 permutation core.

that area savings are not worth the performance impact when it comes to unprotected implementations. Nevertheless, this approach may lead to major area savings with TI versions; since shared S-Boxes take 246 GE and 255 GE (see 4.1).

4 Threshold Implementations

Since the application of TI on affine functions is trivial [24], we mainly focus on the sharing of the S-box. Then, we briefly discuss the shared architectures.

4.1 The Shared S-Box

The PRIMATEs S-Box is an almost bent permutation with excellent linear and differential properties and is affine equivalent to the cubic power mapping in $GF(2^5)$. As can be seen from its algebraic normal form in Equation (1) (with x_i and y_i correspond to input and output bits assuming 0 to be the index for MSB), the S-Box is quadratic which makes it suitable for efficient TI.

$$\begin{aligned} y_0 &= x_0x_2 + x_0x_3 + x_1x_4 + x_1 + x_2x_3 + x_2 + x_3 \\ y_1 &= x_0 + x_1x_2 + x_1x_3 + x_2x_3 + x_2x_4 + x_3 \\ y_2 &= x_0x_1 + x_0x_4 + x_0 + x_1 + x_2x_3 + x_2x_4 \\ y_3 &= x_0x_2 + x_0x_4 + x_0 + x_1x_2 + x_3x_4 \\ y_4 &= x_0x_3 + x_1 + x_2x_4 + x_4 + 1 \end{aligned} \tag{1}$$

In this paper, we only consider the first-order TI of the PRIMATEs S-Box. Since at least $d + 1$ shares are required to implement any function of degree d, we first implement the shared S-Box with 3 shares. We provide the component functions of y_0 for this version in Equation (2) as an example where x_i^j refers to the j-th share of x_i.

$$
\begin{aligned}
y_0^1 &= ((x_0^2 + x_0^3)(x_2^2 + x_2^3)) + ((x_0^2 + x_0^3)(x_3^2 + x_3^3)) + \\
&\quad ((x_1^2 + x_1^3)(x_4^2 + x_4^3)) + ((x_2^2 + x_2^3)(x_3^2 + x_3^3)) + x_2^2 + x_3^2 + x_1^2 \\
y_0^2 &= (x_0^1 x_2^3 + x_0^3 x_2^1 + x_0^1 x_2^1) + (x_0^1 x_3^3 + x_0^3 x_3^1 + x_0^1 x_3^1) + \\
&\quad (x_1^1 x_4^3 + x_1^3 x_4^1 + x_1^1 x_4^1) + (x_2^1 x_3^3 + x_2^3 x_3^1 + x_2^1 x_3^1) + x_1^3 + x_2^3 + x_3^3 \\
y_0^3 &= (x_0^1 x_2^2 + x_0^2 x_2^1) + (x_0^1 x_3^2 + x_0^2 x_3^1) + \\
&\quad (x_1^1 x_4^2 + x_1^2 x_4^1) + (x_2^1 x_3^2 + x_2^2 x_3^1) + x_1^1 + x_2^1 + x_3^1
\end{aligned}
\tag{2}
$$

This particular sharing fails to satisfy the uniformity property of TI. Since we were not able to find a uniform 3-share TI with our limited computational resources, we re-mask the S-box output in order to attain provable security. Remasking is performed similarly to [10] in order to reduce the fresh randomness requirement.

Additionally, we implement a 4-share uniform TI which is provided for y_0 in Equation (3), as an example. Even though this implementation has bigger area compared to its 3-share counterpart, it does not require fresh randomness after the initial sharing. This may lead to significant savings once a random number generator is included in the design.

$$
\begin{aligned}
y_0^1 &= ((x_0^2 + x_0^3 + x_0^4)(x_2^2 + x_2^3 + x_2^4)) + ((x_0^2 + x_0^3 + x_0^4)(x_3^2 + x_3^3 + x_3^4)) + \\
&\quad ((x_1^2 + x_1^3 + x_1^4)(x_4^2 + x_4^3 + x_4^4)) + ((x_2^2 + x_2^3 + x_2^4)(x_3^2 + x_3^3 + x_3^4)) + \\
&\quad x_1^2 + x_2^2 + x_3^2 \\
y_0^2 &= ((x_0^1(x_2^3 + x_2^4)) + (x_2^1(x_0^3 + x_0^4)) + (x_0^1 x_2^1)) + \\
&\quad ((x_0^1(x_3^3 + x_3^4)) + (x_3^1(x_0^3 + x_0^4)) + (x_0^1 x_3^1)) + ((x_1^1(x_4^3 + x_4^4)) + (x_4^1(x_1^3 + x_1^4)) + \\
&\quad (x_1^1 x_4^1)) + x_1^3 + ((x_2^1(x_3^3 + x_3^4)) + (x_3^1(x_2^3 + x_2^4)) + (x_2^1 x_3^1)) + x_2^3 + x_3^3 \\
y_0^3 &= ((x_0^1 x_2^2) + (x_0^2 x_2^1)) + ((x_0^1 x_3^2) + (x_0^2 x_3^1)) + ((x_1^1 x_4^2) + (x_1^2 x_4^1)) + \\
&\quad x_1^4 + ((x_2^1 x_3^2) + (x_2^2 x_3^1)) + x_2^4 + x_3^4 \\
y_0^4 &= x_1^1 + x_2^1 + x_3^1
\end{aligned}
\tag{3}
$$

4.2 Architectures

We implement a total of 6 threshold implementations of PRIMATEs. Firstly, implementations vary in number of shares. Secondly, we utilize different degrees of serialization to ensure tradeoffs between cost and performance. For convenience we name them P80-9^3, P80-9^4, P120-9^4, P80-95^3, P80-95^4, P120-127^4, where the superscript numbers indicate the number of shares.

Figure 4 depicts the datapath of P80-9^3. For each of the S-Box shares a copy of the MC circuit, and an additional SRF, needs to be added to the design to maintain the masked state. The additional control logic required to implement

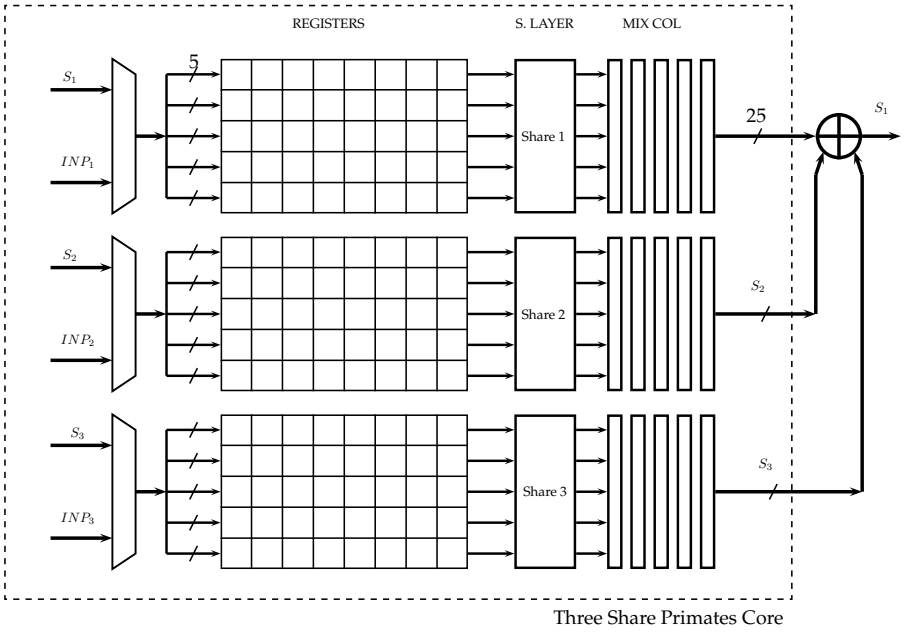

Fig. 4. Data path of the 3-share P80-9 permutation core.

the circuit is minimal. This is in contrast to the serialized threshold implementation of AES [23], where the S-Box needed to be implemented using pipelined stages. The structure of the Primates S-Box obviates the need for such pipelining.

5 Implementation Results

All implementations are synthesized from RTL code written in VHDL. We use Synopsys Design Compiler v2015.06 to synthesize each design. Furthermore, we use Synopsys PrimeTime v2015.06 with PX add-on to perform more accurate static timing analysis and switching activity based power estimation.

We provide synthesis results using 2 standard-cell libraries: Faraday UMC 90 nm, generic core in Low-K RVT process, 1.2 V power supply (UMC 90), and NangateOpenCellLibrary 45 nm, PDKv1_3_v2010_12 (NAN 45), in Table 3.

Along the maximum frequency and area we provide performance figures at the operating frequency of 100 kHz. These include throughput, implementation efficiency (throughput per unit of area), dynamic and static power consumption, and energy efficiency (energy per state bit in every round). We use these figures to benchmark P80 and P120 permutations, in order to identify the most suitable lightweight scenario for each of the permutation cores.

Firstly, we observe that the control logic ModeCtrl discussed in Section utilizes negligible amount of resources—under 100 GE. All control modules

Table 3. Post-synthesis hardware implementation results.

Design	Library	Max. Freq.	Area	@ 100 kHz				
				T'put	Impl. Eff'cy	D. Pwr.	S. Pwr.	E. Eff'cy
		[MHz]	[kGE]	$[\frac{\text{Mbit}}{\text{s}}]$	$[\frac{\text{Mbit}}{\text{kGE}\cdot\text{s}}]$	[μW]	[nW]	$[\frac{\text{pJ}}{\text{bit}}]$
ApeCtrl	UMC90	361.58	0.06	—	—	0.01	1.15	—
	NAN45	606.76	0.09	—	—	0.01	1.37	—
HanCtrl	UMC90	487.02	0.05	—	—	0.01	1.02	—
	NAN45	683.06	0.08	—	—	0.01	1.11	—
GibCtrl	UMC90	599.23	0.05	—	—	0.01	1.04	—
	NAN45	749.29	0.07	—	—	0.01	1.23	—
P80-1	UMC90	179.60	3.68	20.00	5.43	2.32	74.00	0.12
	NAN45	341.53	4.72		4.24	1.63	83.30	0.09
P80-9	UMC90	256.74	1.43	2.22	1.56	0.74	29.80	0.35
	NAN45	439.77	2.05		1.08	0.78	32.80	0.37
P80-16	UMC90	509.50	1.20	1.25	1.04	0.68	25.20	0.57
	NAN45	896.38	1.78		0.70	0.42	27.60	0.36
P80-41	UMC90	204.18	1.32	0.49	0.37	0.46	26.70	0.99
	NAN45	267.61	1.98		0.25	0.30	31.80	0.68
P120-1	UMC90	142.27	6.32	28.00	4.42	4.61	137.00	0.17
	NAN45	281.31	8.23		3.51	3.65	159.00	0.14
P120-9	UMC90	183.69	2.17	3.11	1.43	1.26	46.00	0.42
	NAN45	490.17	3.10		1.00	1.17	165.00	0.43
P120-16	UMC90	447.21	1.82	1.75	0.96	1.13	38.60	0.67
	NAN45	722.33	2.69		0.65	0.80	42.60	0.48
P120-57	UMC90	114.32	1.87	0.49	0.26	0.63	36.80	1.37
	NAN45	239.24	2.79		0.18	0.40	44.80	0.91
P80-9^3	UMC90	162.60	5.18	2.22	0.428	0.81	1.04	0.36
	NAN45	251.25	7.20		0.308	0.49	65.90	0.25
P80-95^3	UMC90	151.74	4.72	0.21	0.044	0.60	0.86	2.55
	NAN45	315.45	6.33		0.033	0.35	54.40	1.92
P80-9^4	UMC90	133.15	6.15	2.22	0.360	0.87	1.07	0.39
	NAN45	249.33	9.24		0.240	0.53	86.20	0.28
P120-9^4	UMC90	79.05	10.30	3.11	0.302	1.41	2.34	0.45
	NAN45	104.20	13.84		0.225	0.81	125.00	0.30
P80-95^4	UMC90	181.49	6.19	0.21	0.033	0.78	1.12	3.71
	NAN45	298.50	8.31		0.025	0.50	70.40	2.71
P120-127^4	UMC90	204.45	8.60	0.22	0.025	1.04	1.60	4.72
	NAN45	300.00	11.51		0.019	0.70	96.50	3.61

have shorter critical paths than any of the permutation cores, therefore they can not pose as a computational bottleneck. Secondly, we observe that areas of P80 and P120 scale in a linear fashion with respect to the state-size. Therefore, we focus discussion on the P80, for simplicity.

On the one hand, P80-1 is dominant in throughput and energy efficiency. Area costs of computing the entire round in parallel make P80-1 large for most

resource-constrained devices (e.g., RFID tags). Therefore P80-1 is better suited for battery powered devices, and applications where high throughput and long battery life is of greater interest (e.g., wireless sensor nodes).

On the other hand, serial implementations seem very suited for constrained devices. P80-16 is the smallest implementation, which requires only 1.2 kGE. Unfortunately, this low resource cost comes at the requirement of storing value of the state between rounds externally. Hence, feasibility of this implementation strongly depends on the specifics of the application and resources of the platform that relies on PRIMATEs-80. Also, this implementation has maximum frequency considerably higher than the rest. Second largest implementation, P80-41 removes the problem of external storage requirement, and has the lowest power consumption. Decrease in power consumption is due to the decreased size of combinatorial logic used for MC transformation. Nevertheless, it is not followed by area decrease, since SRF of P80-41 is more costly. Namely, SRF size is increased by additional multiplexers[1] required for performing SR transformation in one clock cycle, as well as additional control that makes rest of the SRF idle while MC transformation is performed on a column. Still, the pitfall of this implementation is the heavily reduced throughput due to the high latency. Lastly, P80-9 is 50% (320%) more efficient than P80-16 (P80-41), at the price of 20% (8%) area, and 9% (61%) power, increase.

When it comes to TI, we see that due to the efficient design of the S-Box, there is no need for pipelining S-Boxes. Therefore, the circuit size is increased approximately linearly with respect to the number of shares.

6 Usability, Comparison, and Discussion

Implementation results presented in Section 5 serve the purpose of benchmarking the core permutation. Here we discuss how these results fit real-world applications. Figure 5 gives estimated encryption throughput of PRIMATEs based on different serial implementations, with respect to the size of authenticated data and plaintext in bytes; assuming 100 kHz operating frequency. Throughput is estimated based on the latency of encryption in all 3 modes of operation, APE, GIBBON, HANUMAN. Due to the fact that PRIMATEs may be used for applications that require encryption and (or) authentication of very short messages, we include the latency of initialization for each mode. Lastly, note that Figure 5 does not take any interface overhead into account, other than assuming that input of data into state (e.g., initialization of key and nonce), and XOR-ing of data into state (e.g., for tag generation) introduces latency of one round of the core permutation.

Since GIBBON employs p_2, and p_3 which use only 6 rounds, it is asymptotically twice as fast as the other two modes, allowing throughput up to 70 kbits/sec for GIBBON using P80-9. Therefore it is preferred when performance

[1] Note that further area decrease SRF can be done by replacing flip-flop-multiplexer pairs with scan flip-flops. This has no practical significance as scan flip-flops are intended for test inputs in the during production.

takes precedence over slightly lowered security. Furthermore, APE is slightly slower than HANUMAN, due to the initial processing of the nonce (cf. [3]) and the highest level of security that follows. Taking the 12 kbits/sec throughput at 100 kHz operating frequency into account, serializing MC transformation in P80-41, and P120-57 versions makes them suitable for devices where their low power consumption outweighs low throughput. Namely, as p_1, and p_4 permutations which are effectively used for encryption (authentication) of each block require 12 rounds, this results in 492 clock cycles latency for P80-41 (684 for P120-57). On the other hand, 9 (108 cycles per block in APE, HANUMAN; 54 cycles per block in GIBBON), and 16 (144 cycles per block in APE, HANUMAN; 72 cycles per block in GIBBON) clock cycle version satisfies these requirements with a significant margin; and we deem them very usable for most constrained application even with significant interface overhead. Moreover, since it requires no external storage, we recommend P80-9, and P120-9 as the most sound choices.

Lastly, we look into some of the industrial standards, devised for lightweight devices (e.g., smartcards). For example, EPCGlobal Gen2 and ISO/IEC 18000-63 passive UHF RFID air interface standards discussed in [28] prescribes the following constraints: clock frequency (1.5–2.5 MHz) and response latency (39.06–187.50 μs). These constraints allow RFID devices between 58 and 468 clock cycles to respond. Considering this type of constraints, and the 12 kbits/sec at

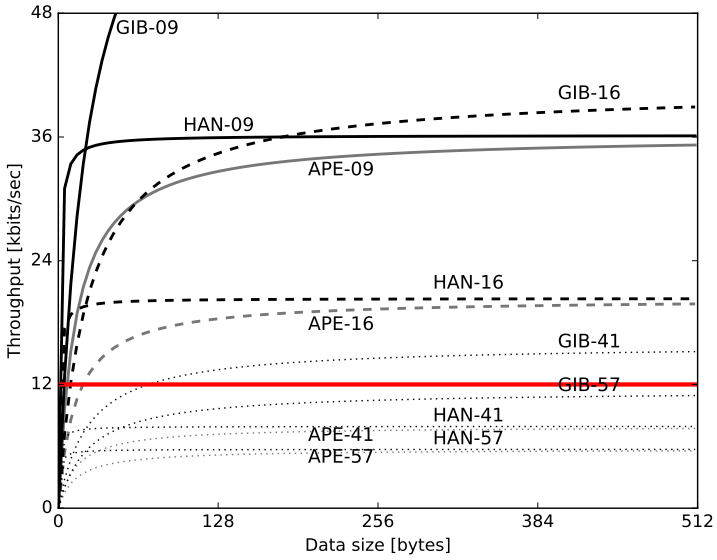

Fig. 5. Estimated encryption throughput at 100 kHz operating frequency

.

100 kHz requirement, we believe that lightweight ciphers can be fairly compared based on the metric presented in Table 4. Assuming a fixed operating frequency, and the corresponding throughput, performance constraints of implementations can be compared solely based on the block size. We believe this is a practical usability metric, which can be easily used in conjecture with area, power, and energy constraints.

Table 4. Maximal cycle latency per block assuming 12 kbits/sec, 100 kHz.

Block size [bit]	16	32	40	64	80	96	128	256
Max. number of cycles	132	261	326	521	651	782	1042	2084

7 PRIMATEs Coprocessor

In this section we present a coprocessor architecture which can be used for all PRIMATEs. It is designed to be compatible with 8-, and 16-bit microprocessors; and features an interface to PRIMATEs cores, efficient in terms of latency and hardware overheads. Moreover, this approach can be applied for other sponge-based ciphers (e.g., [9, 15]).

The key to the interface design is depicted in Figure 6. Namely, instead of mapping the entire SRF into microprocessor memory space, we introduced a single row-sized InterFace (IF) register. IF is treated as a number of memory mapped registers with 8-, or 16-bit parallel input, by the microprocessor. Alternatively, IF provides element-wise shift capability; which allows it to communicate with each row of the underlying SRF via circular shifts. This way of accessing SRF (row-wise) has multiple benefits: IF allows data to be written to the permutation core in block-pipeline fashion, effectively introducing clock cycle latency overhead equal to the number of elements in a row; it provides translation from the microprocessor word to element-sized word without any precomputation; element-sized data path conforms FIFO construction of each row, hence it results in zero area overhead; allows implementation of all PRIMATEs. Namely, all steps required for PRIMATEs schemes (cf. [3]) can be divided into two groups: computationally expensive p_i permutations, and computationally feasible data flow operations (e.g., data parsing and writing to the core, etc.). Therefore, we believe that the best design strategy is to leave data flow operations to a microprocessor (or an upper level FSM), and dedicate coprocessor to performing p_i. All required read, write, and XOR operations can be achieved using three types of row interfaces depicted in Figure 6.

Namely, Rc encapsulates a row of the SRF with all of its logic, can only be written to, and introduces no hardware overhead. Rb is a row which can also be read-written via circular shifts, where reading introduces overhead of 5-bit multiplexer entry at the output (5-bit AND2 is 6.25 GE in UMC 90), and slightly more control logic. Ra allows the data from IF register to be circularly

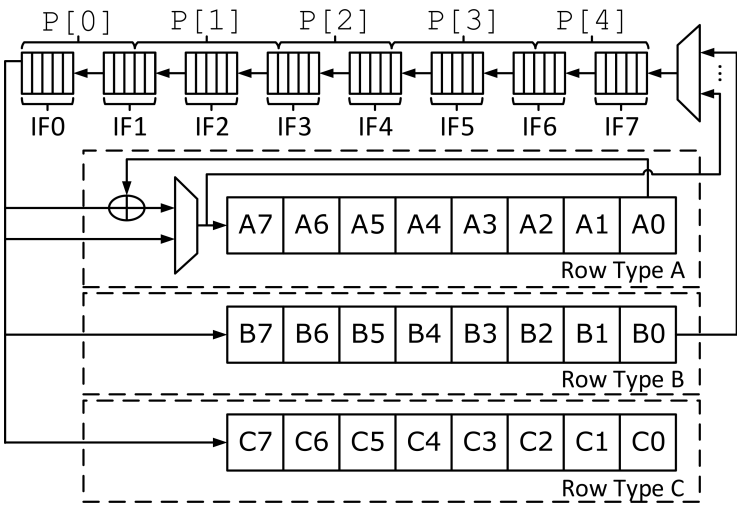

Fig. 6. Coprocessor interface and the XOR instruction support.

shifted as is (same as RB), or circularly XOR-ed to the value stored in the row, depending on the value of the XOR bit of the instruction (see Table 5). This approach requires no additional latency for the encryption of blocks, as they are XOR-ed to the rate as they are being written to SRF. Lastly, RA row can be read without changing its content by a circular XOR of 0^{40}. The cost of each 5-bit XOR-multiplexer required to support XOR is 23.75 GE in UMC 90.

In the particular case of PRIMATEs IF register is 40-bits wide, hence it can be mapped to 5 8-bit registers, or 3 16-bit registers. Additionally, an 8-bit Instruction Register (IR) is required for PRIMATEs instructions, which can be mapped to the remaining byte of one of the 3 16-bit registers.

7.1 HANUMAN-80 Coprocessor

As an example, we design and implement a HANUMAN-80 coprocessor, based on the preferred P80-9 core. Subset of PRIMATEs micro-instructions, required for HANUMAN-80 encryption and decryption (cf. [3]), is given in Table 5. Top-level architecture, depicted in Figure 7, is adapted to MSP430 microcontroller family [21].

We use Spartan-6 FPGA (XC6SLX45-3CSG324) to implement and test our design, next to an OpenMSP430 implementation [16]. On this platform coprocessor fits in a total of 72 (1.06%) slices (206 FFs and 278 LUTs.) In ASIC, using UMC 90 standard-cell library from Section 5, the entire coprocessor requires 2 kGE. Note that HANUMAN-80 compliant P80-9 (with the data path arhictecture from Figure 2) requires 1.69 kGE. Overhead of 0.26 kGE (18.68% larger than the raw P80-9 core of 1.43 kGE) includes all the glue logic; and entire control logic, including HANCTRL and the FSM of the coprocessor for fetching,

decoding, and executing micro-instructions. Since each row has separate enable signal, area as well as power savings can easily be achieved by gating the clock instead of using flip-flops with enable. Furthermore overhead of 0.31 kGE is introduced for the 8-bit instruction unit and the 40-bit IF register , which enables circular access to SRF in a block-pipeline manner, allowing to almost negligible interface overhead. Alternatively, this register can be removed, and the SRF redesigned to be accessible to the microprocessor. This leads to area decrease; but also increases the latency, and makes it heavily dependent on the latency of the write cycle of the microprocessor, since pipeline feature is absent. Average dynamic power at the operating frequency of 100 kHz is 0.49 μW, while the 39.90 nW of power are dissipated statically; consuming 5.3 μW/MHz in total,

Table 5. Instruction Set of the HANUMAN-80 coprocessor.

Mnemonic	Code	Description
RESET	`0-------`	Perform software reset.
WAIT	`1000-000`	Put coprocessor in a idle state.
P1	`1----001`	Perform p_1 permutation.
P1S	`1----101`	Perform p_1 permutation with padding spill into capacity.
P4	`1----001`	Perform p_4 permutation.
RATEX	`10011111`	XOR in to rate.
RATES	`10010111`	Shift in to rate.
RDRATE	`10011111`	XOR in 0^{40} to rate; emulated rate read.
CAP1S	`10100111`	Shift in to capacity row 1, R/W.
CAP2S	`10110111`	Shift in to capacity row 2, R/W.
CAP3S	`11000111`	Shift in to capacity row 3, W.
CAP4S	`11010111`	Shift in to capacity row 4, W.

Dash, "-", can be replaced by either zero or one.

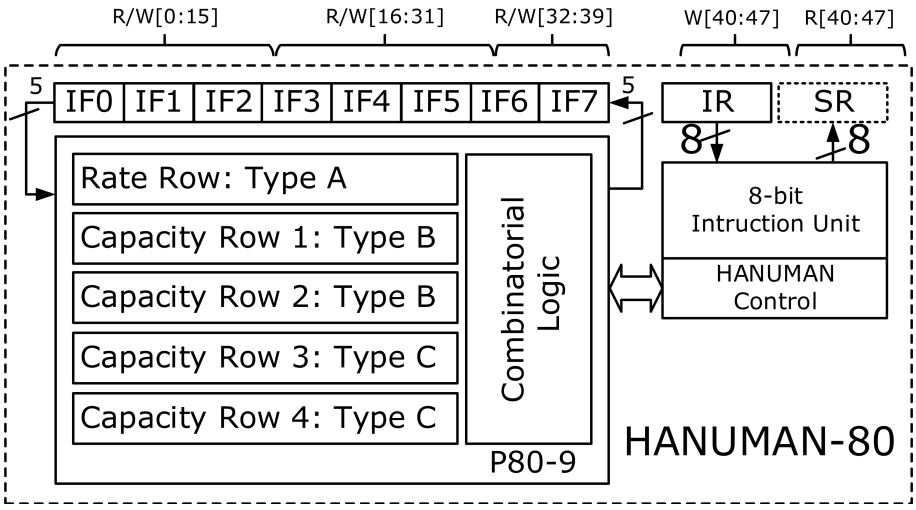

Fig. 7. HANUMAN-80 coprocessor architecture.

Table 6. Implementation comparison.

Design	Tech.	Area [kGE]	Block size [bit]	# TCLK	$\frac{\#\,\text{TCLK}}{\text{Table 4}} \cdot 100$ [%]
AES♠ [23]	UMC 180	2.4	128	226	21.69
HANUMAN-80	UMC 90	2.00	40	118	36.20
GIBBON-80	UMC 90	≈2.00◇	40	64	19.63
Minalpher [29]	NAN 45	2.81♣	256	304	14.59
Ascon-64 [15]	UMC 90	5.86	64	354	67.95
Ascon-x-low-area [15]	UMC 90	3.75	64	3072	589.63

♠ UMCL18G212T3 based on a UMC 180 nm library. ◇ GIBBON-80 coprocessor estimated area. ♣ Authors state that no optimization is performed.

which fits requirements of the industry [28]. Throughput estimated based on the 118 clock cycles (12×9 for p_i, 8 for circular shift, and 2 for instruction fetch and decode) latency per data block (asymptotically) is 33 kbits/sec. This is a valid assumption, since no additional storage is required for pre-computing and storing the initialization phase, while tag generation is simply the XOR operation. Under these assumptions, estimated energy consumption is 15.60 pJ/bit.

Implementation Comparison. Performing a fair evaluation of different candidates is a difficult task for several reasons. Firstly, it requires a common interface, equally suitable for all candidates. Secondly, broad area of use cases, ranging from RFID chips to high-end hardware accelerators, might not make use of a single interface objective enough. Thirdly, implementers present their results in different technological libraries and processes of each library; which makes area and power comparison more difficult. Consistently with the lightweight tone of this work, and assumed real-world limitations, we use Table 6 to benchmark several implementations of second-round CAESAR candidates, against the smallest implementation of AES [23]. Coherently to the discussion from Section 5, we use area, clock cycle latency(# TCLK), and block size as main comparison parameters. Additionally, we present how well does each implementation fit the constraints setting from Table 4 (lower percentage is better). Only two candidates are chosen at this time for the lack of lightweight ASIC implementations of others.

8 Conclusions and Future Work

Based on the hardware implementations of PRIMATEs family of authenticated ciphers, and adjacent discussion we find PRIMATEs to be very suitable for constrained devices. Namely, uninterfaced implementations of the permutation that lies in the heart of PRIMATEs takes only 60–72% of the 2 kGE lightweightness criteria. As shown by example of the HANUMAN-80 coprocessor, this leaves plenty of space for the implementation of interface and control logic. Furthermore, without any circuit-level optimizations (e.g., clock gating, power gating),

or picking technology library for low-power application, our coprocessor fits the all of the commonly accepted criteria in practice; in terms of throughput, area, and average power consumption proposed in [28]. Additionally, presented variety of TI shows that securing PRIMATEs against first order DPA can be achieved using as little as 4.3 kGE.

Additionally, by looking at the PRIMATEs AEAD schemes in [3], and the design of our interface, we observe that by simply using different row interfaces depicted in Figure 6 coprocessor can be turned into GIBBON-80, and APE-80. Similarly, by using P120-9 instead of P80-9 all 3 modes of operation can be satisfied for the increased security level, with minor changes in hardware, conforming the same architecture. Therefore, both security levels, for all modes of operation can be achieved on the same chip—or any reasonable combination tailored for the specific application—with very little hardware overhead.

Further evaluation of this family requires a tapeout of a versatile PRIMATEs chip, which would allow detailed assessment of SCA security. This study would also allow us to study how efficiently can different modes of operation and security levels coexist on a single chip. Furthermore, we plan to study TI of PRIMATEs in order to achieve same levels of security using less randomness and resources, as well as higher-order DPA security.

Acknowledgments. This work has started during a short-term research mission, COST Action IC1306: Cryptography for Secure Digital Interaction. In addition, this work is supported in part by the Research Council KU Leuven (C16/15/058), by the Flemish Government (G.00130.13N and FWO G.0876.14N), by the Flemish iMinds projects, by the Hercules Foundation (AKUL/11/19), and by the European Commission through the Horizon 2020 research and innovation programme under contract No H2020-ICT-2014-644371 WITDOM, H2020-ICT-2014-644209 HEAT, H2020-MSCA-ITN-2014-643161 ECRYPT-NET, and under grant agreement 644052 HECTOR. D. Šijačić is a beneficiary of a Marie Curie-Sklodowska research grant. B. Bilgin is a postdoctoral researcher of the Research Foundation—Flanders (FWO). B. Yang is supported by the Scholarship from China Scholarship Council (No.201206210295).

References

1. Cryptographic Competitions. http://competitions.cr.yp.to/index.html.
2. N. Al Fardan and K. Paterson. Lucky Thirteen: Breaking the TLS and DTLS Record Protocols. In *IEEE Symposium on Security and Privacy (SP) 2013*, pages 526–540, May 2013.
3. E. Andreeva, B. Bilgin, A. Bogdanov, A. Luykx, F. Mendel, B. Mennink, N. Mouha, Q. Wang, and K. Yasuda. PRIMATEs v1.02, Submission to the CAESAR Competition. http://primates.ae/wp-content/uploads/primatesv1.02.pdf.
4. E. Andreeva, A. Bogdanov, A. Luykx, B. Mennink, N. Mouha, and K. Yasuda. How to securely release unverified plaintext in authenticated encryption. In P. Sarkar and T. Iwata, editors, *Advances in Cryptology – ASIACRYPT 2014*, volume 8873 of *LNCS*, pages 105–125. Springer Berlin Heidelberg, 2014.

5. J.-P. Aumasson, L. Henzen, W. Meier, and M. Naya-Plasencia. Quark: A Lightweight Hash. In S. Mangard and F.-X. Standaert, editors, *Cryptographic Hardware and Embedded Systems, CHES 2010*, volume 6225 of *LNCS*, pages 1–15. Springer Berlin Heidelberg, 2010.
6. R. Beaulieu, D. Shors, J. Smith, S. Treatman-Clark, B. Weeks, and L. Wingers. The simon and speck families of lightweight block ciphers. Cryptology ePrint Archive, Report 2013/404, 2013. http://eprint.iacr.org/.
7. M. Bellare and C. Namprempre. Authenticated Encryption: Relations among notions and analysis of the generic composition paradigm. *IACR ePrint Archive*, 2000:25, 2000.
8. G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche. Cryptographic Sponge Functions. http://sponge.noekeon.org/CSF-0.1.pdf.
9. G. Bertoni, J. Daemen, M. Peeters, G. Van Assche, and R. Keer. Caesar submission: Ketje v1. http://competitions.cr.yp.to/round1/ketjev11.pdf.
10. B. Bilgin, J. Daemen, V. Nikov, S. Nikova, V. Rijmen, and G. Van Assche. Efficient and first-order dpa resistant implementations of keccak. In A. Francillon and P. Rohatgi, editors, *Smart Card Research and Advanced Applications*, LNCS, pages 187–199. Springer International Publishing, 2014.
11. B. Bilgin, B. Gierlichs, S. Nikova, V. Nikov, and V. Rijmen. *A More Efficient AES Threshold Implementation*, pages 267–284. Springer International Publishing, Cham, 2014.
12. A. Bogdanov, M. Knežević, G. Leander, D. Toz, K. Varc, and I. Verbauwhede. Spongent: A lightweight hash function. In B. Preneel and T. Takagi, editors, *Cryptographic Hardware and Embedded Systems CHES 2011*, volume 6917 of *LNCS*, pages 312–325. Springer Berlin Heidelberg, 2011.
13. A. Bogdanov, L. Knudsen, G. Leander, C. Paar, A. Poschmann, M. Robshaw, Y. Seurin, and C. Vikkelsoe. PRESENT: An Ultra-Lightweight Block Cipher. In P. Paillier and I. Verbauwhede, editors, *Cryptographic Hardware and Embedded Systems – CHES 2007*, volume 4727 of *LNCS*, pages 450–466. Springer Berlin Heidelberg, 2007.
14. C. De Cannire, O. Dunkelman, and M. Knežević. KATAN and KTANTAN – a family of small and efficient hardware-oriented block ciphers. In C. Clavier and K. Gaj, editors, *Cryptographic Hardware and Embedded Systems - CHES 2009*, volume 5747 of *LNCS*, pages 272–288. Springer Berlin Heidelberg, 2009.
15. C. Dobraunig, M. Eichlseder, F. Mendel, and M. Schläffer. Ascon a family of authenticated encryption algorithms. http://ascon.iaik.tugraz.at/index.html.
16. O. Girard. *openMSP430*, rev. 1.15, may 19, 2015. http://opencores.org.
17. Z. Gong, S. Nikova, and Y. Law. KLEIN: A New Family of Lightweight Block Ciphers. In A. Juels and C. Paar, editors, *RFID. Security and Privacy*, volume 7055 of *LNCS*, pages 1–18. Springer Berlin Heidelberg, 2012.
18. J. Guo, T. Peyrin, and A. Poschmann. The PHOTON Family of Lightweight Hash Functions. In P. Rogaway, editor, *Advances in Cryptology CRYPTO 2011*, volume 6841 of *LNCS*, pages 222–239. Springer Berlin Heidelberg, 2011.
19. J. Guo, T. Peyrin, A. Poschmann, and M. Robshaw. The LED Block Cipher. In B. Preneel and T. Takagi, editors, *Cryptographic Hardware and Embedded Systems CHES 2011*, volume 6917 of *LNCS*, pages 326–341. Springer Berlin Heidelberg, 2011.
20. M. Hell, T. Johansson, A. Maximov, and W. Meier. The Grain Family of Stream Ciphers. In M. Robshaw and O. Billet, editors, *New Stream Cipher Designs*, volume 4986 of *LNCS*, pages 179–190. Springer Berlin Heidelberg, 2008.

21. T. Instruments. Msp430x1xx family user's guide. http://www.ti.com/lit/ug/slau049f/slau049f.pdf.
22. M. Knezevic, V. Nikov, and P. Rombouts. Low-Latency Encryption Is Lightweight = Light + Wait? In E. Prouff and P. Schaumont, editors, *Cryptographic Hardware and Embedded Systems CHES 2012*, volume 7428 of *LNCS*, pages 426–446. Springer Berlin Heidelberg, 2012.
23. A. Moradi, A. Poschman, S. Ling, C. Paar, and H. Wang. Pushing the limits: A very compact and a threshold implementation of aes. volume 6632 of *LNCS*, pages 69–88. Springer, 2011.
24. S. Nikova, C. Rechberger, and V. Rijmen. Threshold implementations against side-channel attacks and glitches. In *Proceedings of Information and Communications Security, 8th International Conference, ICICS 2006, number 4307 in LNCS*, pages 529–545. Springer-Verlag, 2006.
25. B. J. Paul C. Kocher, Joshua Jaffe. Differential power analysis. In *CRYPTO '99 Proceedings of the 19th Annual International Cryptology*, LNCS, pages 388–397. Springer-Verlag London, 1999.
26. P. Rogaway. Authenticated-encryption with Associated-data. In *Proceedings of the 9th ACM Conference on Computer and Communications Security*, CCS '02, pages 98–107, New York, NY, USA, 2002. ACM.
27. C. Rolfes, A. Poschmann, G. Leander, and C. Paar. *Ultra-Lightweight Implementations for Smart Devices – Security for 1000 Gate Equivalents*, pages 89–103. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.
28. M.-J. O. Saarinen and D. Engels. A Do-It-All-Cipher for RFID: Design Requirements (Extended Abstract) . IACR ePrint Archive, 2012. http://eprint.iacr.org/.
29. Y. Sasaki, Y. Todo, K. Aoki, Y. Naito, T. Sugaware, Y. Murakami, M. M., and H. S. Minalpher v1. https://competitions.cr.yp.to/round1/minalpherv1.pdf.
30. W. Wu and L. Zhang. LBlock: A Lightweight Block Cipher. In J. Lopez and G. Tsudik, editors, *Applied Cryptography and Network Security*, volume 6715 of *LNCS*, pages 327–344. Springer Berlin Heidelberg, 2011.