SPONGENT: The Design Space of Lightweight Cryptographic Hashing

Andrey Bogdanov[1], Miroslav Knežević[1,2], Gregor Leander[3], Deniz Toz[1], Kerem Varıcı[1], and Ingrid Verbauwhede[1]

[1] Katholieke Universiteit Leuven, ESAT/COSIC and IBBT, Belgium
{andrey.bogdanov, deniz.toz, kerem.varici, ingrid.verbauwhede}@esat.kuleuven.be
[2] NXP Semiconductors, Leuven, Belgium
miroslav.knezevic@nxp.com
[3] DTU Mathematics, Technical University of Denmark
g.leander@mat.dtu.dk

Abstract. The design of secure yet efficiently implementable cryptographic algorithms is a fundamental problem of cryptography. Lately, lightweight cryptography – optimizing the algorithms to fit the most constrained environments – has received a great deal of attention, the recent research being mainly focused on building block ciphers. As opposed to that, the design of lightweight hash functions is still far from being well-investigated with only few proposals in the public domain.
In this article, we aim to address this gap by exploring the design space of lightweight hash functions based on the sponge construction instantiated with PRESENT-type permutations. The resulting family of hash functions is called SPONGENT. We propose 13 SPONGENT variants – for different levels of collision and (second) preimage resistance as well as for various implementation constraints. For each of them we provide several ASIC hardware implementations - ranging from the lowest area to the highest throughput. We make efforts to address the fairness of comparison with other designs in the field by providing an exhaustive hardware evaluation on various technologies, including an open core library. We also prove essential differential properties of SPONGENT permutations, give a security analysis in terms of collision and preimage resistance, as well as study in detail dedicated linear distinguishers.
Key words: hash function, lightweight cryptography, low-cost cryptography, low-power design, sponge construction, PRESENT, SPONGENT, RFID.

1 Introduction

1.1 Motivation

As crucial applications go pervasive, the need for security in RFID and sensor networks is dramatically increasing, which requires secure yet efficiently implementable cryptographic primitives including secret-key ciphers and hash functions. In such constrained environments, the area and power consumption of a primitive usually comes to the fore and standard algorithms are often prohibitively expensive to implement.

Once this research problem was identified, the cryptographic community designed a number of tailored lightweight cryptographic algorithms to specifically address this challenge: stream ciphers like Trivium [18,16], Grain [23,24], and Mickey [3] as well as block ciphers like SEA [46], DESL, DESXL [35], HIGHT [27], mCrypton [36], KATAN/KTANTAN [17], and PRESENT [10] — to mention only a small selection of the lightweight designs.

Rather recently, some significant work on lightweight hash functions has been also performed: [11] describes ways of using the PRESENT block cipher in hashing modes of operation and [1] and [21] take the approach of designing a dedicated lightweight hash function based on a sponge construction [15,7] resulting in two hash functions QUARK and PHOTON.

Among the most prominent security applications targeted by a lightweight hash function are (including the ones requiring preimage security only and collision security only):

- **Lightweight signature schemes:** ECC over $\mathbb{F}_{2^{163}}$ is implementable with just 11.904 GE without key storage after synthesis and around 15.000 GE on a chip [22]. For comparison, the smallest published SHA-256 implementation [32] requires 8.588 GE and the reportedly most compact SHA-3 finalists BLAKE and Grøstl need 13.560 GE [25] and 14.620 GE [47], respectively, to our best knowledge. Hence, adding a hashing engine based on one of these functions to a lightweight ECC implementation nearly doubles the footprint.
- **RFID security protocols** often rely on hash functions [2,41,49]. Some of the applications require collision resistance and some of them do not, just needing preimage security. An interesting case is constituted by keyed message authentication codes (MAC) often used in this context. Here, a lightweight hash function can require less area than a lightweight block cipher in a MAC mode at a fixed level of offline and online security. MACs can be also designed using sponge primitives [8].
- **Random number generation** in hardware is used for ephemeral key generation in public-key schemes, producing random input for cryptographic protocols, and for masking schemes in implementations with protection against side-channel attacks. This frequently needs a preimage-resistant hash function. Using a hash function for pseudorandom number generator (PRNG), given a seed, provides backward security which a block cipher based PRNG (e.g. in OFB mode) does not: Once the key is leaked e.g. through a side-channel attack, the adversary can compute the previous outputs of the block cipher based PRNG. Moreover, the postprocessing of a physical random number generator sometimes includes a preimage-resistant hash function.
- **Post-quantum signature schemes** can be built upon a hash function using Merkle trees [39], [12]. There have been several attempts to efficiently implement it [45,44]. Having a lightweight hash function allows to derive a more compact implementation of the Merkle signature scheme.

However, while for multiple block ciphers, designs have already closely approached the minimum ASIC hardware footprint theoretically attainable, it does not seem the case for some recent lightweight hash functions so far. This article proposes the family of sponge-based lightweight hash functions SPONGENT with a smaller footprint than most existing dedicated lightweight hash functions: PRESENT in hashing modes and QUARK. Its area is comparable to that of PHOTON, though most of the time being slightly more compact. However, a fair comparison in terms of area requirements is a challenging task, since the area occupation is highly dependent on the implementation, technology and tools used. To address this challenge, we provide implementation figures for SPONGENT on four different technologies. In order to make the future comparisons with our designs easier, we also provide the hardware figures based on an open core library.

For some SPONGENT variants, similarly to QUARK and PHOTON, a part of this advantage comes from a reduced level of second preimage security, while maintaining the standard level collision resistance. The other SPONGENT variants attain the standard preimage, second preimage and collision security, while having area requirements much lower than those of SHA-1, SHA-2, and SHA-3 finalists. This design subspace has not been specifically addressed by any previous concrete lightweight hash function proposal. Whereas we note that the design ideas of PRESENT in hashing modes, QUARK and PHOTON might be extended to any set of security parameters.

1.2 Design considerations for lightweight hashing

The footprint of a hash function is mainly determined by

1. the number of state bits (incl. the key schedule for block cipher based designs) as well as
2. the size of functional and control logic used in a round function.

For highly serialized implementations (usually used to attain low area and power), the logic size is normally rather small and the state size dominates the total area requirements of the design. Among the recent hash functions, QUARK, while using novel ideas of reducing the state size to minimize (1), does not appear to provide the smallest possible logic size, which is mainly due to the Boolean functions with many inputs used in its round transform. In contrast to that, SPONGENT keeps the round function very simple which reduces the logic size close to the smallest theoretically possible, thus, minimizing (2) and resulting in a significantly more compact design.

As shown in [11], using a lightweight block cipher in a hashing mode (single block length such as Davies-Meyer or double block length such as Hirose) is not necessarily an optimal choice for reducing the footprint, the major restriction being the doubling of the datapath storage requirement due to the feed-forward operation.

At the same time, no feed-forward is necessary for the sponge construction, which is the design approach of choice in this work. In a permutation-based sponge construction, let r be the *rate* (the number of bits input or output per one permutation call), c be the *capacity* (internal state bits not used for input or output), and n be the hash length in bits.

To explore the design space of lightweight hashing, we propose to instantiate the sponge construction with a PRESENT-type permutation. The resulting construction is called SPONGENT and we refer to its various parameterizations as SPONGENT-$n/c/r$ for different hash sizes n, capacities c, and rates r. SPONGENT is a hermetic sponge, i.e., we do not allow the underlying permutation to have any structural distinguishers. More precisely, for five different hash sizes of $n \in \{88, 128, 160, 224, 256\}$, covering most security applications in the field, we consider (up to) three types of preimage and second-preimage security levels:

– **Full preimage and second-preimage security.** The standard security requirements for a hash function with an n-bit output size are collision resistance of $2^{n/2}$ as well as preimage and second-preimage resistance of 2^n. For this, in SPONGENT, we set $r = n$ and $c = 2n$ to obtain SPONGENT-88/176/88, SPONGENT-128/256/128, SPONGENT-160/320/160, SPONGENT-224/448/224, and SPONGENT-256/512/256.
– **Reduced second-preimage security.** The design of [1] as well as the works [7,8,15] convincingly demonstrate that a permutation-based sponge construction can allow to almost halve the state size for $n \geq c$ and reasonably small r. In this case, the preimage and second-preimage resistances are reduced to 2^{n-r} and $2^{c/2}$, correspondingly, while the collision resistance remains at the level of $2^{c/2}$. In most embedded scenarios, where a lightweight hash function is likely to be used, the full second-preimage security is not a necessary requirement. For relatively small rate r, the loss of preimage security is limited. So we take this parametrization in the design of the smallest SPONGENT variants with $n \approx c$ for small r and obtain SPONGENT-88/80/8, SPONGENT-128/128/8, SPONGENT-160/160/16, SPONGENT-224/224/16, and SPONGENT-256/256/16. These five SPONGENT-variants were published in a shortened conference version [9] of this article.
– **Reduced preimage and second-preimage security.** In some applications, the collision security is of concern only and one can abandon the requirement of preimage security to be close to 2^n. In a permutation-based sponge, going for $c = n$ and $r = n/2$, results in the reduction

of both the preimage security and second-preimage security to $2^{n/2}$, while maintaining the full collision security of $2^{n/2}$. On the implementation side, this parametrization can yield a favorable ratio between the rate and the permutation size which reduces the time-area product. We use this approach in the design of SPONGENT-160/160/80, SPONGENT-224/224/112, and SPONGENT-256/256/128.

The group of all SPONGENT variants with the same output size of n bits is referred to as SPONGENT-n. The SPONGENT-88 functions are designed for extremely restricted scenarios and low preimage security requirements. They can be used e.g. in some RFID protocols and for PRNGs. SPONGENT-128 and SPONGENT-160 might be used in highly constrained applications with low and middle requirements for collision security. The latter also provides compatibility to the SHA-1 interfaces. The parameters of SPONGENT-224 and SPONGENT-256 correspond to those of a subset of SHA-2 and SHA-3 to make SPONGENT compatible to the standard interfaces in usual lightweight embedded scenarios.

1.3 Organization of the article

The remainder of the article is organized as follows. Section 2 describes the design of SPONGENT and gives a design rationale. Section 3 presents some results of security analysis, including proven lower bounds on the number of differentially active S-boxes, best differential characteristics found, rebound attacks, and linear attacks. In Section 4, the implementation results are given for a range of trade-offs. We conclude in Section 5.

2 The design of SPONGENT

SPONGENT is a sponge construction based on a wide PRESENT-type permutation. Given a finite number of input bits, it produces an n-bit hash value. A design goal for SPONGENT is to follow the hermetic sponge strategy (no structural distinguishers for the underlying permutation are allowed).

2.1 Permutation-based sponge construction

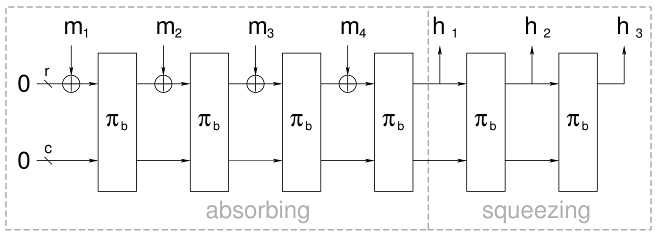

Fig. 1. Sponge construction based on a b-bit permutation π_b with capacity c bits and rate r bits. m_i are r-bit message blocks. h_i are parts of the hash value.

SPONGENT relies on a sponge construction – a simple iterated design that takes a variable-length input and can produce an output of an arbitrary length based on a permutation π_b operating on a state of a fixed number b of bits. The size of the internal state $b = r + c \geq n$ is called *width*, where r is the *rate* and c the *capacity*.

The sponge construction proceeds in three phases (see also Figure 1):

– **Initialization phase:** the message is padded by a single bit 1 followed by a necessary number of 0 bits up to a multiple of r bits (e.g., if $r = 8$, then the 1-bit message '0' is transformed to '01000000'). Then it is cut into blocks of r bits.
– **Absorbing phase:** the r-bit input message blocks are xored into the first r bits of the state, interleaved with applications of the permutation π_b.
– **Squeezing phase:** the first r bits of the state are returned as output, interleaved with applications of the permutation π_b, until n bits are returned.

In SPONGENT, the b-bit 0 is taken as the initial value before the absorbing phase. In all SPONGENT variants, except SPONGENT-88/80/8, the hash size n equals either capacity c or $2c$. The message chunks are xored into the r rightmost bit positions of the state. The same r bit positions form parts of the hash output.

Let a permutation-based sponge construction have $n \geq c$ and $c/2 > r$ which is fulfilled for the parameter choices of most of the SPONGENT variants. Then the works [7,8,15] imply the preimage security of 2^{n-r} as well as the second preimage and collision securities of $2^{c/2}$ if this construction is hermetic (that is, if the underlying permutation does not have any structural distinguishers). The best preimage attack we are aware of in this case has a computational complexity of $2^{n-r} + 2^{c/2}$. Later, this work is extended in [21] and preimage security is defined more generalized form: $min(2^{min(n,\ c+r)}, max(2^{min(n-r,\ c)}, 2^{c/2}))$.

For permutation-based sponge constructions with $n < c$ and $c/2 \leq r$ such as the remaining SPONGENT variants, it follows from the same works that the second preimage security is 2^n and collision security is $2^{c/2}$. The previous preimage attack also works for this case hence we claim that the preimage security is $min(2^n, max(2^{n-r}, 2^{c/2}))$ since $n - r < c$.

2.2 Parameters

We propose 13 variants of SPONGENT with five different hash output lengths at multiple security levels, see Table 1.

2.3 PRESENT-type permutation

The permutation $\pi_b : \mathbb{F}_2^b \rightarrow \mathbb{F}_2^b$ is an R-round transform of the input STATE of b bits that can be outlined at a top-level as:

```
for $i = 1$ to $R$ do
    STATE ← ɹǝʇunoƆl$_b(i)$ $\oplus$ STATE $\oplus$ lCounter$_b(i)$
    STATE ← sBoxLayer$_b$(STATE)
    STATE ← pLayer$_b$(STATE)
end for
```

where sBoxLayer$_b$ and pLayer$_b$ describe how the STATE evolves. For ease of design, only widths b with $4|b$ are allowed. The number R of rounds depends on block size b and can be found in Subsection 2.2 (see also Table 1). lCounter$_b(i)$ is the state of an LFSR dependent on b at time i which yields the round constant in round i and is added to the rightmost bits of STATE. ɹǝʇunoƆl$_b(i)$ is the value of lCounter$_b(i)$ with its bits in reversed order and is added to the leftmost bits of STATE.

Table 1. 13 SPONGENT variants.

	n (bit)	b (bit)	c (bit)	r (bit)	R number of rounds	security(bit) pre.	2nd pre.	col.
SPONGENT-88/80/8	88	88	80	8	45	80	40	40
SPONGENT-88/176/88	88	264	176	88	135	88	88	44
SPONGENT-128/128/8	128	136	128	8	70	120	64	64
SPONGENT-128/256/128	128	384	256	128	195	128	128	64
SPONGENT-160/160/16	160	176	160	16	90	144	80	80
SPONGENT-160/160/80	160	240	160	80	120	80	80	80
SPONGENT-160/320/160	160	480	320	160	240	160	160	80
SPONGENT-224/224/16	224	240	224	16	120	208	112	112
SPONGENT-224/224/112	224	336	224	112	170	112	112	112
SPONGENT-224/448/224	224	672	448	224	340	224	224	112
SPONGENT-256/256/16	256	272	256	16	140	240	128	128
SPONGENT-256/256/128	256	384	256	128	195	128	128	128
SPONGENT-256/512/256	256	768	512	256	385	256	256	128

The following building blocks are generalizations of the PRESENT structure to larger b-bit widths:

1. sBoxLayer$_b$: This denotes the use of a 4-bit to 4-bit S-box $S : \mathbb{F}_2^4 \to \mathbb{F}_2^4$ which is applied $b/4$ times in parallel. The action of the S-box in hexadecimal notation is given by the following table:

x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$S[x]$	E	D	B	0	2	1	4	F	7	A	8	5	9	C	3	6

2. pLayer$_b$: This is an extension of the (inverse) PRESENT bit-permutation and moves bit j of STATE to bit position $P_b(j)$, where

$$P_b(j) = \begin{cases} j \cdot b/4 \mod b-1, & \text{if } j \in \{0,\dots,b-2\} \\ b-1, & \text{if } j = b-1. \end{cases}$$

and can be seen in Figure 2.

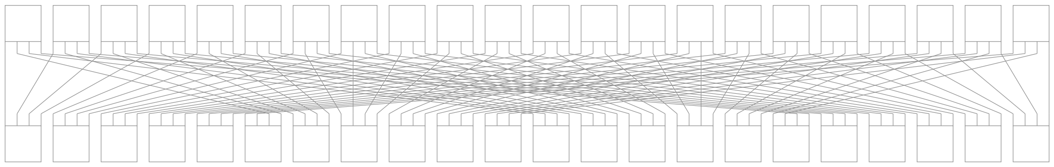

Fig. 2. The bit permutation layer of SPONGENT-88 at the example of pLayer$_{88}$.

3. lCounter$_b$: This is one of the four $\lceil \log_2 R \rceil$-bit LFSRs. The LFSR is clocked once every time its state has been used and its final value is all ones. If ζ is the root of unity in the corresponding binary finite field, the n-bit LFRSs defined by the polynomials given below are used for the SPONGENT variants.

LFSR size (bit)	Primitive Polynomial
6	$\zeta^6 + \zeta^5 + 1$
7	$\zeta^7 + \zeta + 1$
8	$\zeta^8 + \zeta^4 + \zeta^3 + \zeta^2 + 1$
9	$\zeta^9 + \zeta^4 + 1$

Table 2 provides sizes and initial values of all the LFSRs.

Table 2. Initial values of lCounter_b for all SPONGENT variants.

	LFSR size (bit)	Initial Value (hex)
SPONGENT-88/80/8	6	05
SPONGENT-88/176/88	8	D2
SPONGENT-128/128/8	7	7A
SPONGENT-128/256/128	8	FB
SPONGENT-160/160/16	7	45
SPONGENT-160/160/80	7	01
SPONGENT-160/320/160	8	A7
SPONGENT-224/224/16	7	01
SPONGENT-224/224/112	8	52
SPONGENT-224/448/224	9	105
SPONGENT-256/256/16	8	9E
SPONGENT-256/256/128	8	FB
SPONGENT-256/512/256	9	015

2.4 Design rationale

The overall design approach for SPONGENT is to target low area while favoring simplicity.

The 4-bit S-box is the major block of functional logic in a serial low-area implementation of SPONGENT. It fulfills the PRESENT design criteria in terms of differential and linear properties [10]. Moreover, any linear approximation over the S-box involving only single bits both in the input and output masks is unbiased. This aims to restrict the linear hull effect discovered in round-reduced PRESENT.

The function of the bit permutation pLayer is to provide good diffusion, by acting together with the S-box, while having a limited impact on the area requirements. This is its main design goal, while a bit permutation may occupy additional space in silicon. The counters lCounter and ɿɘƚnuoƆl are mainly aimed to prevent sliding properties and make prospective cryptanalysis approaches using properties like invariant subspaces [34] more involving.

The structures of the bit permutation and the S-box in SPONGENT make it possible to prove the following differential property (see Subsection 3.1 for the proof):

Theorem 1. *Any 5-round differential characteristic of the underlying permutation of* SPONGENT *with* $b \geq 64$ *has a minimum of* 10 *active S-boxes. Moreover, any 6-round differential characteristic of the underlying permutation of* SPONGENT *with* $b \geq 256$ *has a minimum of* 14 *active S-boxes.*

The concept of counting active S-boxes is central to the differential cryptanalysis. The minimum number of active S-boxes relates to the maximum differential characteristic probability of the construction. Since in the hash setting there are no random and independent key values added between the rounds, this relation is not exact (in fact that it is even not exact for most practical keyed block ciphers). However, differentially active S-boxes are still the major technique used to evaluate the security of SPN-based hash functions.

An important property of the SPONGENT S-box is that its maximum differential probability is 2^{-2}. This fact and the assumption of the independency of difference propagation in different rounds yield an upper bound on the differential characteristic probability of 2^{-20} over 5 rounds and of 2^{-28} over 6 rounds for $b \geq 256$ which follows from the claims of Theorem 1.

Theorem 1 is used to determine the number R of rounds in permutation π_b: R is chosen in a way that π_b provides at least b active S-boxes. Other types of analysis are performed in the next section.

3 Security Analysis

In this section, we discuss the security of SPONGENT against known cryptanalytic attacks by applying the most important state-of-the-art methods of cryptanalysis and investigating their complexity.

Table 3. Differential characteristics with lowest numbers of differentially active S-boxes (ASN). The probabilities are calculated assuming the independency of round computations.

# of	SPONGENT-88/80/8		SPONGENT-128/128/8		SPONGENT-160/160/16		SPONGENT-224/224/16		SPONGENT-160/160/80	
rounds	ASN	Prob	ASN	Prob	ASN	Prob	ASN	Prob	ASN	Prob
5	10	2^{-21}	10	2^{-22}	10	2^{-21}	10	2^{-21}	14	2^{-21}
10	20	2^{-47}	24	2^{-60}	20	2^{-50}	20	2^{-43}	32	2^{-43}
15	30	2^{-74}	40	2^{-101}	30	2^{-79}	30	2^{-66}	52	2^{-66}
# of	SPONGENT-88/176/88		SPONGENT-128/256/128		SPONGENT-160/320/160		SPONGENT-224/224/112		SPONGENT-224/448/224	
rounds	ASN	Prob	ASN	Prob	ASN	Prob	ASN	Prob	ASN	Prob
6	14	2^{-28}	14	2^{-28}	14	2^{-28}	14	2^{-28}	14	2^{-28}
12	41	2^{-96}	37	2^{-72}	39	2^{-93}	36	2^{-88}		
18	64	2^{-158}	52	2^{-119}	65	2^{-157}	66	2^{-174}		
# of	SPONGENT-256/256/16		SPONGENT-256/256/128		SPONGENT-256/512/256					
rounds	ASN	Prob	ASN	Prob	ASN	Prob				
6	14	2^{-28}	14	2^{-28}	14	2^{-28}				
12	32	2^{-73}	50	2^{-123}	34	2^{-84}				
18	52	2^{-128}	68	2^{-169}	54	2^{-128}				

3.1 Resistance against differential cryptanalysis

Here we analyze the resistance of SPONGENT against differential attacks where Theorem 1 plays a key role providing a lower bound on the number of active S-boxes in a differential characteristic. The similarities of the SPONGENT permutations and the basic PRESENT cipher allow to reuse some of the results obtained for PRESENT in [10]. More precisely, the results on the number of differentially active S-boxes over 5 and 6 rounds will hold for all respective SPONGENT variants which is reflected in Theorem 1. The proof of the Theorem 1 is as follows:

Proof. [*Theorem 1*] The statements for SPONGENT variants with $64 \leq b \leq 255$ can directly be proven by applying the same technique used in [10, Appendix III]. The proof of the 6-round bounds for

SPONGENT variants with $b \geq 256$ in Theorem 1 is based on some extended observations. Here, we will only give the proof for when the width, b, is a multiple of 64 bits, i.e., $b = 64n$. The proof for other b values can also be obtained by making use of the observations given below. Since the proof is specific to each b and hence more tedious, we do not present them here.

We obtain n groups and $4n$ subgroups by calling each four consecutive S-boxes as a *subgroup* and each sixteen consecutive S-boxes as a *group*. To be more specific: subgroup i is comprised of the S-boxes $[4(i-1) \ldots 4i-1]$ and similarly group j has the subgroups $[4(j-1) \ldots 4j-1]$. (see Figure 3). By examining the substitution and linear layers, one can make the following observations:

1. The S-box of SPONGENT is such that a difference in single input bit causes a difference in at least two output bits or vice versa.
2. The input bits to an S-box come from four distinct S-boxes of the same subgroup.
3. The input bits to a subgroup of four S-boxes come from 16 distinct S-boxes of the same group.
4. The input bits to a group of 16 S-boxes come from 64 different S-boxes.
5. The four output bits from a particular S-box enter four distinct S-boxes, each of which belongs to a distinct group of S-boxes in the subsequent round.
6. The output bits of S-boxes in distinct groups go to distinct S-boxes in distinct subgroups.
7. The output bits of S-boxes in distinct subgroups go to distinct S-boxes.

For the latter statement (SPONGENT-256), one has to deal with more cases. Consider six consecutive rounds of SPONGENT ranging from i to $i+5$ for $i \in [1 \ldots 155]$. Let D_j be the number of active S-boxes in round j. If $D_j \geq 3$, for $i \leq j \leq i+5$, then the theorem trivially holds. So let us suppose that one of D_j is equal to one first and to two then. We have the following cases:

Case $D_{i+2} = 1$. By using observation 1, we can deduce that $D_{i+1} + D_{i+3} \geq 3$ and all active S-boxes of round $i+1$ belong to the same subgroup from observation 2. Each of these active S-boxes have only a single bit difference in their output. So, according to observation 3 we have that $D_i \geq 2D_{i+1}$. Conversely, according to observation 5, all active S-boxes in round $i+3$ belong to distinct groups and have only a single bit difference in their input. So, according to observation 6, we have that $D_{i+4} \geq 2D_{i+3}$. Moreover, all active S-boxes in round $i+4$ belong to distinct subgroups and have only a single bit difference in their input. Thus, by using observation 7, we obtain that $D_{i+5} \geq 2D_{i+4}$ and can conclude that $\sum_{j=i}^{i+5} D_j \geq 1 + 3 + 2 \times 3 + 4D_{i+3} \geq 14$.

Case $D_{i+3} = 1$ If $D_{i+2} = 1$ we can refer to the first case. So, suppose that $D_{i+2} \geq 2$. According to the observation 2, all active S-boxes of round $i+2$ belong to the same subgroup and each of these active S-boxes has only a single bit difference in their output. Thus, according to observation 3, $D_{i+1} \geq 2D_{i+2} \geq 4$. Since all active S-boxes in round $i+1$ belong to distinct S-boxes of the same group and have only a single bit difference in their input, according to observation 4, we have that $D_i \geq 2D_{i+1}$. On the opposite, D_{i+4} and D_{i+5} can get one and two as a minimum value, respectively. Together this gives $\sum_{j=i}^{i+5} D_j \geq 8 + 4 + 2 + 1 + 1 + 2 \geq 18$.

Case $D_{i+1} = 1$ If $D_{i+2} = 1$, then we can refer to the first case. Thus, suppose that $D_{i+2} \geq 2$. According to observation 5, all active S-boxes in round $i+2$ belong to distinct groups and have only a single bit difference in their input. Thus, according to observation 6, we have that $D_{i+3} \geq 2D_{i+2}$. Since all active S-boxes in round $i+3$ belong to distinct subgroups and have only a single bit difference in their input. Therefore, according to observation 7, we have that $D_{i+4} \geq 2D_{i+3}$. To sum up, $\sum_{j=i}^{i+5} D_j \geq 1+1+2+4+8+D_{i+5} \geq 16+D_{i+5} \geq 17$, since $D_{i+4} > 0$ implies that $D_{i+5} \geq 1$.

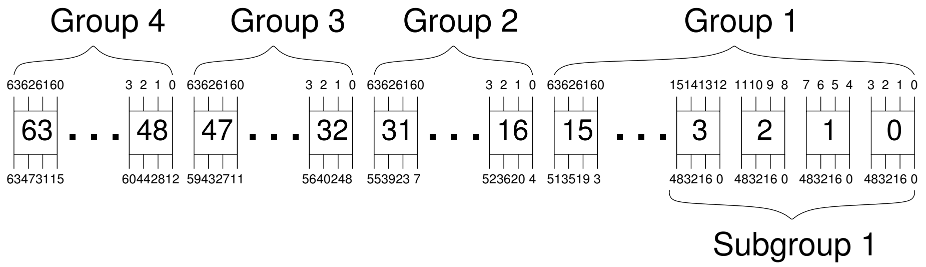

Fig. 3. The grouping and subgrouping of S-boxes for $b = 256$. The input numbers indicate the S-box origin from the previous round and the output numbers indicate the destination S-box in the following round.

Case $D_{i+4} = 1$ If $D_{i+3} = 1$, then we can refer to the second case. So, suppose that $D_{i+3} \geq 2$. According to the observation 2, all active S-boxes of round $i + 3$ belong to the same subgroup and each of those active S-boxes has only a single bit difference in their output. Therefore, according to observation 3, we have that $D_{i+2} \geq D_{i+3}$. Since, all active S-boxes in round $i + 2$ belong to distinct S-boxes of the same group and have only a single bit difference in their input, according to observation 4, we have that $D_{i+1} \geq 2D_{i+2}$. Since $D_{i+1} > 0$, $D_i \geq 1$. Thus, we can conclude that $\sum_{j=i}^{i+5} D_j \geq D_i + 8 + 4 + 2 + 1 + 1 \geq D_i + 16 \geq 17$.
Cases $D_i = 1$ and $D_{i+5} = 1$ are similar to the those for the third and fourth cases.

So far we have considered all paths including one active S-box in one of the rounds and obtained 14 as the minimum number of active S-boxes. But if there exists a path that has two active S-boxes in each round, then the lower bound would be 12. For this purpose, without loss of generality, assume:

$D_{i+1} = D_{i+2} = D_{i+3} = 2$ The two active S-boxes in $i + 2$ are either in the same subgroup or in different subgroups. For the former, from observations 3 and 7, we know that they have single bit of differences coming from two different subgroups of the same group in round $i + 1$. From observation 1, these two S-boxes have at least two bits of input difference, hence we obtain $D_i = 4$ by observation 2 and 3. Furthermore the two S-boxes in round $i + 2$ have two bits of output difference by observation 1. Hence, in round $i + 3$, the active S-boxes have two bits of input and they are in distinct groups by observation 5. Therefore, it is possible to have $D_{i+4} = 2$ in distinct subgroups. Hence by using observation 7, we obtain $D_{i+5} = 4$. Thus, we can conclude that $\sum_{j=i}^{i+5} D_j \geq 4 + 2 + 2 + 2 + 2 + 4 \geq 16$.

For the latter, the two active S-boxes in round $i + 1$ must have two bits of input and by observation 2 their input bits should be coming from distinct S-boxes in the same subgroup. So, the problem is reduced to the former case with one round of shift, and we can immediately say that $D_i = 2$ and $D_{i+4} = 4$. Hence by using observation 7, we obtain $D_{i+5} = 4$. Thus, we can conclude that $\sum_{j=i}^{i+5} D_j \geq 2 + 2 + 2 + 2 + 4 + 4 \geq 16$.

Based on these results, we conclude that the longest run with two active S-boxes in each round is four rounds, and the number of active S-boxes cannot be less than 14.

For all SPONGENT variants, we found that those 5- and 6-round bounds are actually tight. We present the characteristics attaining them in Table 3. Additionally, we perform a branch-and-bound search for longest characteristics with probabilities in the range of 2^{-b}. The results are given in Table 4, most of them based in iterative characteristics.

Table 4. Longest differential characteristics holding with probability in the range of 2^{-b} (under independency assumption)

	# rounds	ASN	Prob
SPONGENT-88/80/8	17	34	2^{-88}
SPONGENT-88/176/88	27	103	2^{-268}
SPONGENT-128/128/8	20	56	2^{-137}
SPONGENT-128/256/128	42	146	2^{-385}
SPONGENT-160/160/16	20	66	2^{-179}
SPONGENT-160/160/80	44	88	2^{-242}
SPONGENT-160/320/160	48	192	2^{-480}
SPONGENT-224/224/16	44	88	2^{-242}
SPONGENT-224/224/112	26	133	2^{-343}
SPONGENT-224/448/224	-	-	-
SPONGENT-256/256/16	30	108	2^{-276}
SPONGENT-256/256/128	31	150	2^{-392}
SPONGENT-256/512/256	85	256	2^{-768}

3.2 Collision attacks

A natural approach to obtain a collision for a sponge construction is to inject a difference in a message block and then cancel the propagated difference by a difference in the next message block, i.e., $(0\dots0||\Delta m_i) \overset{\pi}{\rightarrow} (0\dots0||\Delta m_{i+1})$. For this purpose, we follow a narrow trail strategy using truncated differential characteristics. We start from a given input difference (some difference restricted to S-boxes that the message block is xored into) and look for all paths that go to a fixed output difference (also located in the bitrate part of the state). Based on our experiments, even by using truncated differential characteristics, the probability of such a path is quite low and it is not possible to attack the full number of rounds.

Rebound attack The rebound attack [38], a recent technique for cryptanalysis of hash functions, is applicable to both block cipher based and permutation based hash constructions. It consists of two main steps: the inbound phase where the freedom is used to connect the middle rounds by using the match-in-the-middle technique and the outbound phase where the connected truncated differentials are calculated in both forward and backward directions. It has been mostly used to improve the results on AES-based algorithms (ECHO [6], Grøstl [20], LANE [28], Whirlpool [5]), but it has also been successfully applied to similar permutations (Luffa [30], Keccak [19]).

Compared to the other algorithms the rebound attack has been successfully applied to, the design of SPONGENT imposes some limitations. First of all, since the permutation is bit-oriented, and not byte-oriented, it might be non-trivial to find the path followed by a given input difference and to determine the number of active S-boxes after several rounds. This is mainly due to the difference propagation that strongly depends on the values of the passive part of the state. Moreover, the probability that two inbound phases match requires more detailed analysis. Below we attempt to develop rebound attacks on several SPONGENT variants. Rebound analysis applies similarly to the remaining variants.

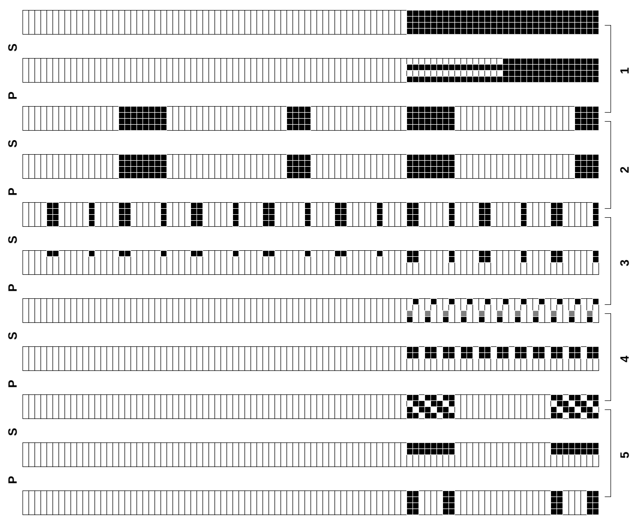

Fig. 4. Differential path for the rebound attack on SPONGENT-128/256/128 (S: sBoxLayer$_{384}$, P: pLayer$_{384}$).

For SPONGENT-88/80/8, we looked for characteristics that match in the middle with the available degrees of freedom coming from the message bits. For 5 and more rounds, when the whole state is active in the matching phase, we would not be able to generate enough pairs by using only a difference in the message bits. Since the expected probability of matching the inbound phases is $2^{-b/4}$ (where $b/4$ is the number of S-boxes) and the available degree of freedom is only 2^{2r}, this argument is also valid for SPONGENT-128/128/8, SPONGENT-160/160/16, SPONGENT-224/224/16, and SPONGENT-256/256/16. For other SPONGENT variants there exist enough degrees of freedom and we decided to explore it with one of the SPONGENT variants.

It is trivial to find one round inbound phase in SPONGENT and then by applying the outbound phase for several rounds, which technically yields a differential characteristic. Since, one third of the state is xored with the message value for the variants whose rate is different from 8 or 16, we have enough flexibility to diffuse the difference through forward and backward direction. But then, merging these differential characteristics seems difficult due to the limited number of pairs generated in the inbound phase.

In our example which is given in Figure 4, we focused on SPONGENT-128/256/128 and found a five-round trail by following the strategy outlined above. In our attack, we fix the input and output differences of sBoxLayer in the fourth round. For a half of the differences, we fix the difference to $1_x \rightarrow 3_x$ and for the other half it is possible to fix the difference to either $4_x \rightarrow 3_x$ or $8_x \rightarrow 3_x$, but not both together. Then, we let the differences diffuse for three rounds in the backward direction and for one round in the forward direction. All possible positions of the active bits are shown in black in Figure 4. Note that in round 5, we impose a restriction on the outputs of the SBoxLayer such that the differences occur only in the bitrate part.

It is possible to generate $4^{11} \cdot 2^{11} = 2^{33}$ pairs in the inbound phase and a pair can satisfy the desired differential trail with a probability of $Pr[B_x \rightarrow \{1_x, 2_x\}]^6 \cdot Pr[D_x \rightarrow \{1_x, 2_x, 3_x\}]^6 \cdot Pr[6_x \rightarrow \{1_x, 2_x\}]^4 = 2^{-26.15}$. Therefore, in total, we expect to have $2^{6.85}$ valid pairs that satisfy the given path.

Bound considerations for the rebound attack The adversary might try to find a way to attack by using multiple inbounds with a sparse differential. Therefore, to explore the security against multiple inbound phases, we put the adversary into a best-case scenario as follows.

We know that there exists no differential characteristic over five rounds with the number of active S-boxes less than 10 for all SPONGENT variants. We can also deduce lower bounds on the number of active S-boxes for $1, 2, 3$, and 4 rounds as $1, 2, 4$ and 6, respectively. Then a bound on the minimum number of active S-boxes, hence the probability of a differential characteristic, for any number of rounds can be approximated by combining these bounds.[4]

The desired bit security level for a sponge construction with respect to collision attacks is $c/2$. From now on we assume that the complexity of each inbound phase is equal to $c/2$ and at least one active S-box matches between two inbound phases (with probability 2^{-8}). Let n_{in} be the number of inbound phases then we have to generate $n_{elm} = 2^{8\cdot(n_{in}-1)/n_{in}}$ elements for each inbound phase. Let p denote the probability of each inbound phase, then p can be at least $2^{-(c/2-\lceil log_2(n_{elm})\rceil)}$ and we can compute the number of rounds in each inbound phase by using the given bounds above.

Under these assumptions, the maximum number of rounds per inbound phase and the percentage of the total number of rounds attacked is given in Table 5.

Table 5. Bounds for rebound attack.

	2 Inbounds		3 Inbounds	
	rounds /inbound	attacked rounds(%)	rounds /inbound	attacked rounds(%)
SPONGENT-88/80/8	9	40.00	9	60.00
SPONGENT-88/176/88	10	14.81	9	20.00
SPONGENT-128/128/8	15	42.86	14	60.00
SPONGENT-128/256/128	14	14.36	13	20.00
SPONGENT-160/160/16	19	42.22	19	63.33
SPONGENT-160/160/80	19	31.67	19	47.50
SPONGENT-160/320/160	17	14.17	16	20.00
SPONGENT-224/224/16	28	46.67	27	67.50
SPONGENT-224/224/112	23	27.06	23	40.59
SPONGENT-224/448/224	23	13.53	23	20.29
SPONGENT-256/256/16	28	40.00	27	57.86
SPONGENT-256/256/128	28	28.72	27	41.54
SPONGENT-256/512/256	28	14.55	27	21.04

3.3 Preimage resistance

Here we apply a meet-in-the-middle approach to obtain preimages on SPONGENT. The attack has two main steps: pre-computation and matching phase. Complexity of the attack is dominated by pre-computation phase.

Since the hash size is n bits, and the data is extracted in r bit chunks, there exists n/r rounds in the squeezing phase. To be able to compute the data backwards in the absorbing phase, we need to know not only h_i's but also d_i values to obtain the input value of the permutation π, where h_i denotes the part of the hash value and d_i is the concatenated part to h_i. The algorithm is as follows:

[4] Note that, Table 3 shows that these bounds might be optimistic.

1. **Pre-computation:** We know that $\pi^{-1}(h_{i+1}, d_{i+1}) = (h_i, d_i)$ for each i in the squeezing phase. Since h_i (r-bits) is already fixed, the probability of finding such d_i is 2^{-r}. Therefore, we start with $2^{((n/r)-1)\cdot r} = 2^{n-r}$ different $d_{n/r}$ values to have a solution for d_1.
2. **Match-in-the-middle:** Choose k such that $k \cdot r \geq c/2$. Then
   – Generate $2^{c/2}$ elements in the backward direction by using (h_1, d_1) and possible values for $m_{k+2}, \ldots, m_{2k+1}$ and store them in a table.
   – Generate $2^{c/2}$ elements in the forward direction by using possible values for $m_1, \ldots, m_k$ and compare with list in the previous step to find a match of c bits (corresponding to capacity) in the middle.
   – Obtain m_{k+1} by xor-ing the r bits (corresponding to bitrate) for the matching elements.

In the pre-computation part, we obtain the required value d_1 to compute the data backwards in the absorbing phase by 2^{n-r} computations. We need $2^{c/2}$ memory to store the elements generated in the second step and $2^{c/2}$ computations are needed to find a full match. These complexities are exactly given in [50] which extends the bounds given in [15] for $c > n$. We have derived those once again here for completeness. The preimage attack complexities together with the parameter k are given in Table 6.

Table 6. Meet-in-the-middle attack results for SPONGENT.

	k	Time Complexity $max(2^{n-r}, 2^{c/2})$	Memory Complexity $(2^{c/2})$
SPONGENT-88/80/8	5	2^{80}	2^{40}
SPONGENT-88/176/88	1	2^{88}	2^{88}
SPONGENT-128/128/8	8	2^{120}	2^{64}
SPONGENT-128/256/128	1	2^{128}	2^{128}
SPONGENT-160/160/16	5	2^{144}	2^{80}
SPONGENT-160/160/80	1	2^{80}	2^{80}
SPONGENT-160/320/160	1	2^{160}	2^{160}
SPONGENT-224/224/16	7	2^{208}	2^{112}
SPONGENT-224/224/112	1	2^{112}	2^{112}
SPONGENT-224/448/224	1	2^{224}	2^{224}
SPONGENT-256/256/16	8	2^{240}	2^{128}
SPONGENT-256/256/128	1	2^{128}	2^{128}
SPONGENT-256/512/256	1	2^{256}	2^{256}

Note that, if $c \leq n - r$, it is sufficient to try all possible 2^c values to construct the whole state in order to obtain a preimage, hence it provides an upper bound for the preimage resistance. If we combine the results we obtain $max(2^{min(n-r,c)}, 2^{c/2})$ and it can be generalized into the form: $min(2^{min(n,\ c+r)}, max(2^{min(n-r,\ c)}, 2^{c/2}))$. Here, $2^{min(n,\ c+r)}$ computations will be necessary depending on the permutation size when the generic attack, defined above, fails.

3.4 Linear attacks

The most successful attacks, the attacks that can break the highest number of rounds, for the block cipher PRESENT are those based on linear approximations. In particular the multi-dimensional linear attack [13] and the statistical saturation attack [14] claim to break up to 26 rounds. It was shown in

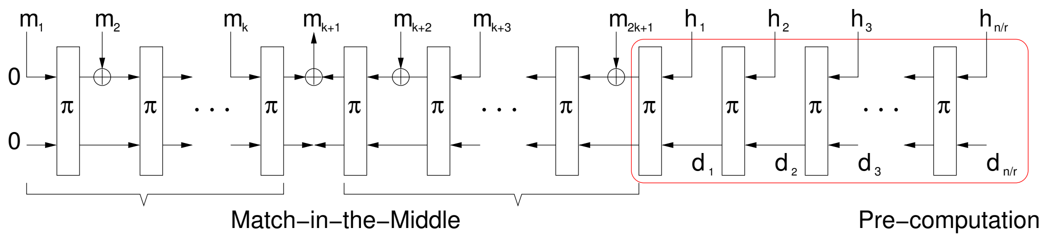

Fig. 5. Meet-in-the-middle attack against sponge construction.

[33] that both attacks are closely related. Moreover, the main reason why these attacks are the most successful attacks on PRESENT so far, is the existence of many linear trails with only one active S-box in each round. It is not immediately clear how linear distinguishers on the SPONGENT permutation π_b could be transferred into collision or (second) pre-image attacks on the hash function. However, as we claim that SPONGENT is a hermetic sponge construction, the existence of such distinguishers has to be excluded. So the SPONGENT S-box was chosen in a way that allows for at most one trail with this property given a linear approximation.

Unlike for the block cipher PRESENT, where the key determines the actual linear correlation between an input and an output mask, for the permutation π_b we can compute the actual linear trail contribution for all trails with only one active S-box in every round. Each such trail over w rounds has a correlation of $\pm 2^{-2w}$ and for each trail determining the sign is easy. More concretely, one can easily compute a $b \times b$ matrix M_t over the rationals such that the entry at position i, j is the correlation coefficient for round t for the linear trail with input mask e_i and output mask e_j. Here e_i (resp. e_j) is the unit vector with a single 1 at position i (resp. j). Note that the matrices M_t are sparse and all very similar, the only difference is caused by the round constant, which induces sign changes at a few positions only.

Given those matrices, it is now possible to compute the maximal linear correlation contribution for those one bit intermediate masks for all one bit input and output masks. For w rounds we simply compute $M^{(w)} = \prod_{i=1}^{w} M_i$ and the maximal correlation is given by $c_w := \max_{i,j} |M_{ij}^{(w)}|$. We compute this value for all SPONGENT variants. Table 7 summarizes those results. Most importantly, this table shows the maximal number of rounds w where the trail contributions is still larger than or equal to $2^{-b/2}$. Beyond this number of rounds, it seems unlikely that distinguishers based on linear approximations exist. For most SPONGENT variants, the best linear hull based on single-bit masks has exactly one linear trail.

4 Hardware Implementations

In this section we provide a wide range of hardware figures by evaluating all of the 13 SPONGENT variants in detail. Not only a comprehensive hardware evaluation is of our primary interest, we also further elaborate on the importance of having the unified benchmarking platform for comparing different lightweight designs. To further stress on the latter issue, we provide the results using four different CMOS technologies. For a thorough evaluation of area, throughput, maximum frequency, and power consumption, we use the UMC 130 nm CMOS generic process (UMC130) provided

Table 7. Results of linear trail correlation based on one bit masks.

	b	max w with $c_w \geq 2^{-b/2}$	R	$\log_2 c_R$
SPONGENT-88/80/8	88	22	45	−90
SPONGENT-88/176/88	264	66	135	−270
SPONGENT-128/128/8	136	34	70	−140
SPONGENT-128/256/128	384	96	195	−388.4
SPONGENT-160/160/16	176	44	90	−180
SPONGENT-160/160/80	240	60	120	−240
SPONGENT-160/320/160	480	122	240	−473.7
SPONGENT-224/224/16	240	60	120	−240
SPONGENT-224/224/112	336	84	170	−340
SPONGENT-224/448/224	673	169	340	−675.3
SPONGENT-256/256/16	272	68	140	−280
SPONGENT-256/256/128	384	96	195	−388.4
SPONGENT-256/512/256	768	192	385	−770

by the Faraday corporation[5]. Moreover, we provide the estimates of the circuit area using three other libraries: UMC 180 nm CMOS generic process (UMC180), an open source NANGATE 45 nm CMOS technology (NANGATE45) [40] as well as the advanced 90 nm CMOS standard cell library provided by NXP Semiconductors (NXP90).

In order to provide very compact implementations, we first focus on serialized designs. We explore different datapath sizes (d) for each of the SPONGENT variants and we focus on $d \in \{4, 8, \frac{b}{2}, b\}$. An architecture representing our serialized datapath is depicted in Fig. 6(a). The control logic consists of a single counter for the cycle count and some extra combinational logic to drive the select signals of the multiplexers. In order to further reduce the area we use so-called scan flip-flops, which act as a combination of two input multiplexer and an ordinary D flip-flop[6]. Instead of providing a reset signal to each flip-flop separately, we use two zero inputs at the multiplexers M_1 and M_2 to correctly initialize all the flip-flops. This additionally reduces hardware resources, as the scan flip-flops with a reset input approximately require an additional GE per bit of storage. With g_i we denote the value of $\mathrm{lCounter}_b(i)$ in round i. $\mathrm{lCounter}_b(i)$ is implemented as an LFSR as explained in Subsection 2.3. The input of the message block m, denoted with dashed line, is omitted in some cases, i.e. $d \geq r$. The pLayer module requires no additional logic except some extra wiring.

Using the most serialized implementation, the smallest variant of the SPONGENT family, SPONGENT-88/80/8, can be implemented using only 738 GE. Even the largest member of the family, SPONGENT-256/512/256, consumes only 5.1 kGE, while providing 256 bits of preimage and second preimage security, and 128 bits of collision resistance. Though some of this advantage is at the expense of a performance reduction, also less serialized (and, thus, faster) implementations result in area requirements significantly lower than 10 kGE. To demonstrate this, we implement all the SPONGENT variants as depicted in Fig. 6(b). Every round now requires a single clock cycle, therefore resulting in faster, yet rather compact designs.

[5] The choice of the UMC130 library for our hardware implementation is driven by the size of a single scan flip-flop. One scan flip-flop in our UMC180 is 6.67 GE large, while in UMC130 it consists of 6.25 GE. In [21], for example, a scan flip-flop of only 6 GE has been reported.

[6] Scan flip-flops are typically used to provide scan-chain based testability of the circuit. Due to the security issues of scan-chain based testing [51], other methods such as Built-In-Self-Test (BIST) are recommended for testing the cryptographic hardware.

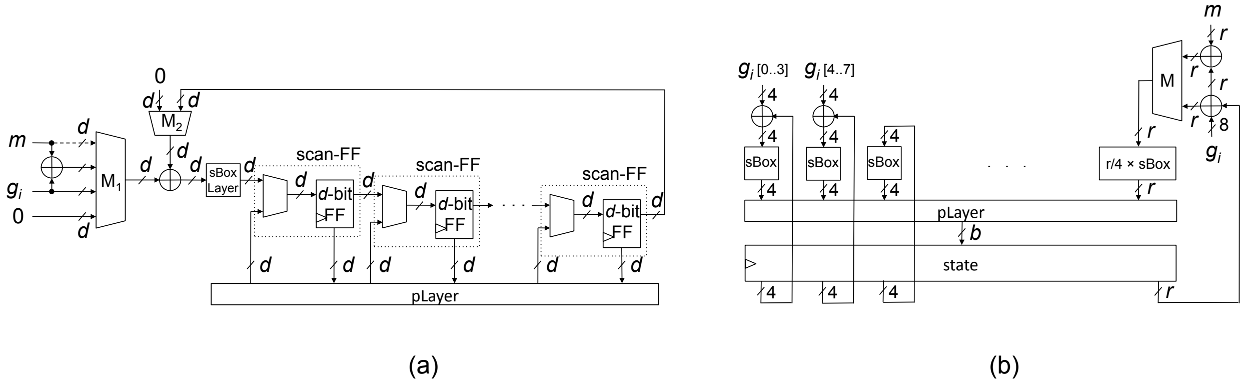

Fig. 6. Hardware architecture representing (a) serial datapath (b) parallel datapath of the SPONGENT variants.

Another courtesy of our proposal is the result of 5 times unrolled design of SPONGENT variants which, all running at the maximum frequency of about 600 MHz, provide a throughput between 360 Mbps and 2 Gbps (depending on the variant) and consume between 5 kGE and 48 kGE.

Next, we present the obtained hardware figures for all of the SPONGENT variants. For the purpose of extensive hardware evaluation we use Synopsys Design Compiler version D-2010.03-SP4 and target the High-Speed UMC 130 nm CMOS generic process provided by Faraday Technology Corporation (fsc0h_d_tc).

The power is estimated by observing the internal switching activity of the complete design. Using Mentor Graphics ModelSim version 10.0 SE, we simulate the circuits' behavior for very long messages and generate the VCD (Value Change Dump) files. The VCD files are then converted to the backward SAIF (Switching Activity Interchange Format) files and used within Synopsys Design Compiler for the accurate estimation of the mean power consumption. A typical frequency of 100 kHz is used for all measurements.

Table 8 reports hardware figures obtained using the aforementioned methodology. For the sake of comparison, we include figures for several state-of-the-art lightweight hash functions. We also include two out of five SHA-3 finalists for which the data of compact hardware implementations is publicly available. We do not compare our design with software-like solutions that benefit from using an external memory for storing the intermediate data. Figure 7 illustrates the wide spectrum of our explored design space, where a typical trade-off between speed and area is scrutinized.

4.1 A Fair Comparison – Mission (Im)possible

A fair comparison of hardware performance between different designs has already been discussed in the literature [17,4]. It is rather obvious that such comparison is only possible once the highly optimized designs are implemented on the same hardware platform, using the same standard cell library and the same synthesis tools (including the design flow scripts). And this all, finally repeated over many different instances (libraries, tools, scripts, etc). However, mainly due to the licensing issues and the designer's preference to use a certain software package, this becomes a very difficult task in practice.

Table 8. Hardware performance of the SPONGENT family and comparison with state-of-the-art lightweight hash designs. The nominal frequency of 100 kHz is assumed in all cases and the power consumption is therefore adjusted accordingly.

Hash function	Security (bit)			Hash	Cycles	Datapath	Process	**Area**	**Throughput**	**Power***
	Pre.	Coll.	2nd Pre.	(bit)		(bit)	(μm)	(GE)	(kbps)	(μW)
SPONGENT-88/80/8	80	40	40	88	990	4	0.13	738	0.81	1.57
					45	88	0.13	1127	17.78	2.31
SPONGENT-88/176/88	88	44	88	88	8910	4	0.13	1912	0.99	3.4
					135	264	0.13	3450	65.19	7.5
SPONGENT-128/128/8	120	64	64	128	2380	4	0.13	1060	0.34	2.20
					70	136	0.13	1687	11.43	3.58
SPONGENT-128/256/128	128	64	128	128	18720	4	0.13	2641	0.68	6.1
					195	384	0.13	5011	65.64	10.9
SPONGENT-160/160/16	144	80	80	160	3960	4	0.13	1329	0.40	2.85
					90	176	0.13	2190	17.78	4.47
SPONGENT-160/160/80	80	80	80	160	7200	4	0.13	1730	1.11	3.4
					120	240	0.13	3139	66.67	6.8
SPONGENT-160/320/160	160	80	160	160	28800	4	0.13	3264	0.56	8.2
					240	480	0.13	6237	66.67	13.6
SPONGENT-224/224/16	208	112	112	224	7200	4	0.13	1728	0.22	3.73
					120	240	0.13	2903	13.33	5.97
SPONGENT-224/224/112	112	112	112	224	14280	4	0.13	2371	0.78	5.0
					170	336	0.13	4406	65.88	9.6
SPONGENT-224/448/224	224	112	224	224	57120	4	0.13	4519	0.39	11.5
					340	672	0.13	8726	65.88	19.2
SPONGENT-256/256/16	240	128	128	256	9520	4	0.13	1950	0.17	4.21
					140	272	0.13	3281	11.43	6.62
SPONGENT-256/256/128	128	128	128	256	18720	4	0.13	2641	0.68	6.1
					195	384	0.13	5011	65.64	10.9
SPONGENT-256/512/256	256	128	256	256	73920	4	0.13	5110	0.35	12.8
					385	768	0.13	9944	66.49	21.9
PHOTON-80/20/16 [21]	64	40	40	80	708	4	0.18	865	2.82	1.59
					132	20	0.18	1168	12.15	2.70
PHOTON-128/16/16 [21]	112	64	64	128	996	4	0.18	1122	1.61	2.29
					156	24	0.18	1708	10.26	3.45
PHOTON-160/36/36 [21]	124	80	80	160	1332	4	0.18	1396	2.70	2.74
					180	28	0.18	2117	20.00	4.35
PHOTON-224/32/32 [21]	192	112	112	224	1716	4	0.18	1735	1.86	4.01
					204	32	0.18	2786	15.69	6.50
PHOTON-256/32/32 [21]	224	128	128	256	996	8	0.18	2177	3.21	4.55
					156	48	0.18	4362	20.51	8.38
U-QUARK [1]	120	64	64	128	544	1	0.18	1379	1.47	2.44
					68	8	0.18	2392	11.76	4.07
D-QUARK [1]	144	80	80	160	704	1	0.18	1702	2.27	3.10
					88	8	0.18	2819	18.18	4.76
S-QUARK [1]	192	112	112	224	1024	1	0.18	2296	3.13	4.35
					64	16	0.18	4640	50.00	8.39
DM-PRESENT-80 [11]	64	32	64	64	547	4	0.18	1600	14.63	1.83
					33	64	0.18	2213	242.42	6.28
DM-PRESENT-128 [11]	64	32	64	64	559	4	0.18	1886	22.90	2.94
					33	128	0.18	2530	387.88	7.49
H-PRESENT-128 [11]	128	64	64	128	559	8	0.18	2330	11.45	6.44
					32	128	0.18	4256	200.00	8.09
C-PRESENT-192 [11]	192	96	192	192	3338	12	0.18	4600	1.90	-
					108	192	0.18	8048	59.26	9.31
KECCAK-f[400] [29]	160	80	160	160	1000	16	0.13	5090	14.40	11.50
					20	16	0.13	10560	720.00	78.10
KECCAK-f[200] [29]	128	64	128	128	900	8	0.13	2520	8.00	5.60
					18	8	0.13	4900	400.00	27.60
SHA-1 [31]	160	80	160	160	450	32	0.25	6812	113.78	11.00
SHA-256 [32]	256	128	256	256	490	32	0.25	8588	104.48	11.20
BLAKE [26]	256	128	256	256	816	32	0.18	13575	62.79	11.16
Grøstl [48]	256	128	256	256	196	64	0.18	14622	261.14	221.00

* The power figures rather serve an illustration purpose. A comparison between different technologies is difficult.

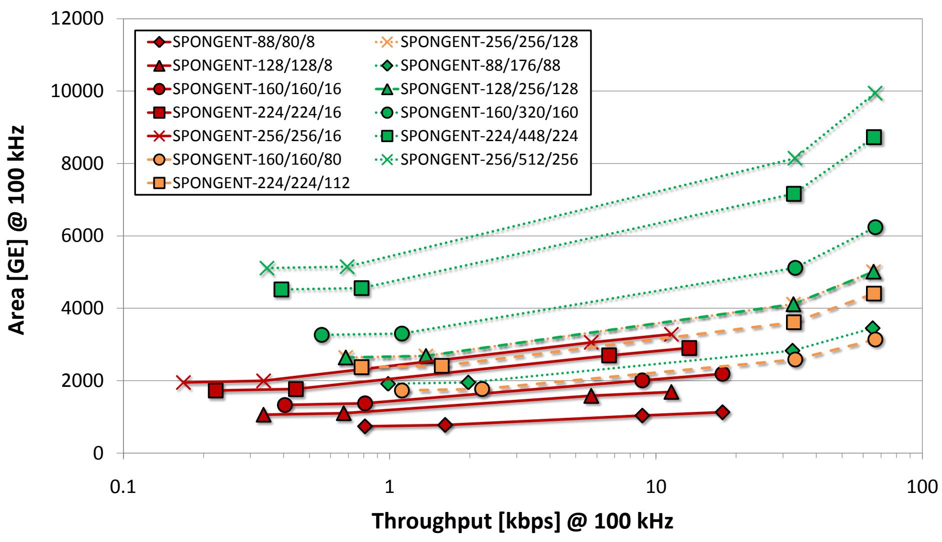

Fig. 7. Area versus throughput trade-off of the SPONGENT hash family.

To partially address this issue and in order to avoid any ambiguity we provide Table 9 with area requirements of the basic building cells from our UMC130 library. The library contains many other cells and we only outline ones that are of particular interest to us. Several special cells acting as a combination of two or more basic gates (e.g. AO is a combination of AND and OR) are also used very often and are appropriate for reducing the physical size of the design. The size of these cells varies, mainly depending on the driving strength of the cell and its input capacitance. The final design provided by the synthesis tool will therefore be driven by many internal factors, e.g. speed constraints, physical area constraints, fan-in, fan-out, length of the wires, and many others.

Moreover, we provide Table 10 where the same SPONGENT RTL designs were synthesized using four different libraries. Compared to our UMC130 library, the overhead of UMC180 and NANGATE45 libraries ranges up to 13 % and 20 %, respectively, while the NXP90 library results in smaller area up to 32 %, which represents a significant margin (the size is compared using gate equivalences).

The main cause of the above described variance is a different cells' size, which is directly related to the library type. A single scan flip-flop consumes at least 6.25 GE and 6.67 GE in UMC130 and UMC180, respectively. The NXP90 library has significantly smaller flip-flops which are the main area consumers in the case of SPONGENT family. On the other hand, NANGATE45 (with a scan flip-flop of 7.67 GE) is an open core library and seems to be a good candidate for accurate comparison between different lightweight designs.

5 Conclusion

In this work, we have explored the design space of lightweight cryptographic hashing by proposing the family of new hash functions SPONGENT tailored for resource-constrained applications. We consider 5 hash sizes for SPONGENT – ranging from the ones offering mainly preimage resistance only to those complying to (a subset of) SHA-2 and SHA-3 parameters. For each parameter set,

Table 9. Area requirements of selected standard cells in our UMC 130 nm library.

Standard cell	Number of inputs	Area [μm^2]	Area [GE]
D flip-flop	1	20 – 40	5 – 10
Scan flip-flop	1	25 – 47	6.25 – 11.75
NOT	1	3 – 28	0.75 – 7
NAND	2	4 – 23	1 – 5.75
	3	6 – 14	1.5 – 3.5
	4	12 – 18	3 – 4.5
NOR	2	4 – 40	1 – 10
	3	6 – 13	1.5 – 3.25
	4	11 – 19	2.75 – 4.75
AND	2	5 – 19	1.25 – 4.75
	3	7 – 16	1.75 – 4
	4	10 – 33	2.5 – 8.25
OR	2	5 – 25	1.25 – 6.25
	3	7 – 26	1.75 – 6.5
XOR	2	11 – 16	2.75 – 4
	3	22 – 26	5.5 – 6.5
	4	30 – 31	7.5 – 7.75
AO, AN	6	6 – 17	1.5 – 4.25
	4	6 – 21	1.5 – 5.25
	6	10 – 25	2.5 – 6.25
	8	15 – 18	3.75 – 4.5
OA, NA	6	5 – 21	1.25 – 5.25
	4	6 – 21	1.5 – 5.25
	6	9 – 18	2.25 – 4.5
	8	15 – 18	3.75 – 4.5
MUX	2	9 – 28	2.25 – 7
	3	16 – 27	4 – 6.75
	4	25 – 35	6.25 – 8.75

AO = AND and OR, AN = AND and NOR,
OA = OR and AND, NA = NOR and AND.

we instantiate SPONGENT using up to three competing security paradigms (all of them offering full collision security): reduced second-preimage security, reduced preimage and second-preimage security, as well as full preimage and second-preimage security. Each parametrization accounts for its unique implementation properties in terms of ASIC hardware footprint, performance and time-area product, which are analyzed in the article. We also perform security analysis in terms of differential properties, linear distinguishers, and rebound attacks.

Acknowledgements. Andrey Bogdanov is a postdoctoral fellow of the Fund for Scientific Research - Flanders (FWO). This work is supported in part by the IAP Programme P6/26 BCRYPT of the Belgian State, by FWO project G.0300.07, by the European Commission under contract number ICT-2007-216676 ECRYPT NoE phase II, by K.U.Leuven-BOF (OT/08/027 and OT/06/40), and by the Research Council K.U.Leuven: GOA TENSE. We would like to thank the reviewers of CHES'11 and LC'11 for their comments.

Table 10. Area of the SPONGENT family compared using four different standard cell libraries.

	Datapath (bit)	Area (GE) UMC 130 nm	UMC 180 nm	NANGATE 45 nm	NXP 90 nm
SPONGENT-88/80/8	4	738	759	869	521
	88	1127	1232	1237	883
SPONGENT-88/176/88	4	1912	1965	2264	1308
	264	3450	3847	3633	2553
SPONGENT-128/128/8	4	1060	1103	1257	737
	136	1687	1855	1831	1279
SPONGENT-128/256/128	4	2641	2724	3183	1813
	384	5011	5581	5715	4167
SPONGENT-160/160/16	4	1329	1367	1572	918
	176	2190	2241	2406	1752
SPONGENT-160/160/80	4	1730	1769	2066	1192
	240	3139	3434	3612	2650
SPONGENT-160/320/160	4	3264	3340	3931	2232
	480	6237	6949	7163	5262
SPONGENT-224/224/16	4	1728	1768	2070	1192
	240	2903	3203	3220	2334
SPONGENT-224/224/112	4	2371	2422	2827	1621
	336	4406	4900	4611	3197
SPONGENT-224/448/224	4	4519	4625	5430	3069
	672	8726	9696	9751	6932
SPONGENT-256/256/16	4	1950	2012	2323	1340
	272	3281	3721	3639	2612
SPONGENT-256/256/128	4	2641	2724	3183	1813
	384	5011	5581	5713	4213
SPONGENT-256/512/256	4	5110	5232	6163	3471
	768	9944	11054	10778	7426

References

1. Aumasson, J.P., Henzen, L., Meier, W., Naya-Plasencia, M.: Quark: A Lightweight Hash. In: Mangard and Standaert [37], pp. 1–15
2. Avoine, G., Oechslin, P.: A Scalable and Provably Secure Hash-Based RFID Protocol. In: PerCom Workshops. pp. 110–114. IEEE Computer Society (2005)
3. Babbage, S., Dodd, M.: The MICKEY Stream Ciphers. In: Robshaw and Billet [42], pp. 191–209
4. Badel, S., Dagtekin, N., Nakahara, J., Ouafi, K., Reffé, N., Sepehrdad, P., Susil, P., Vaudenay, S.: ARMADILLO: A Multi-purpose Cryptographic Primitive Dedicated to Hardware. In: Mangard and Standaert [37], pp. 398–412
5. Barreto, P.S.L.M., Rijmen, V.: The Whirlpool hashing function. In: Proceedings of the 1st NESSIE Workshop. p. 15. Leuven,B (2000)
6. Benadjila, R., Billet, O., Gilbert, H., Macario-Rat, G., Peyrin, T., Robshaw, M., Seurin, Y.: SHA-3 Proposal: ECHO. Submission to NIST (updated) (2009), `http://crypto.rd.francetelecom.com/echo/doc/echo_description_1-5.pdf`
7. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G.: On the Indifferentiability of the Sponge Construction. In: Smart, N.P. (ed.) EUROCRYPT'08. LNCS, vol. 4965, pp. 181–197. Springer (2008)
8. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G.: Sponge-Based Pseudo-Random Number Generators. In: Mangard and Standaert [37], pp. 33–47
9. Bogdanov, A., Knezevic, M., Leander, G., Toz, D., Varici, K., Verbauwhede, I.: SPONGENT: A Lightweight Hash Function. In: Preneel, B., Takagi, T. (eds.) CHES'11. LNCS, vol. 6917, pp. 312–325. Springer (2011)
10. Bogdanov, A., Knudsen, L.R., Leander, G., Paar, C., Poschmann, A., Robshaw, M.J.B., Seurin, Y., Vikkelsoe, C.: PRESENT: An Ultra-Lightweight Block Cipher. In: Paillier, P., Verbauwhede, I. (eds.) CHES'07. LNCS, vol. 4727, pp. 450–466. Springer (2007)
11. Bogdanov, A., Leander, G., Paar, C., Poschmann, A., Robshaw, M.J.B., Seurin, Y.: Hash Functions and RFID Tags: Mind the Gap. In: Oswald, E., Rohatgi, P. (eds.) CHES'08. LNCS, vol. 5154, pp. 283–299. Springer (2008)
12. Buchmann, J., García, L.C.C., Dahmen, E., Döring, M., Klintsevich, E.: CMSS - An Improved Merkle Signature Scheme. In: Barua, R., Lange, T. (eds.) INDOCRYPT'06. LNCS, vol. 4329, pp. 349–363. Springer (2006)
13. Cho, J.Y.: Linear Cryptanalysis of Reduced-Round PRESENT. In: Pieprzyk, J. (ed.) CT-RSA'10. LNCS, vol. 5985, pp. 302–317. Springer (2010)
14. Collard, B., Standaert, F.X.: A Statistical Saturation Attack against the Block Cipher PRESENT. In: Fischlin, M. (ed.) CT-RSA'09. LNCS, vol. 5473, pp. 195–210. Springer (2009)
15. Daemen, J., Peeters, M., Van Assche, G.: Sponge Functions. Ecrypt Hash Workshop 2007 (2007), `http://www.csrc.nist.gov/pki/HashWorkshop/PublicComments/2007May.html`
16. De Cannière, C.: Trivium: A Stream Cipher Construction Inspired by Block Cipher Design Principles. In: Katsikas, S.K., Lopez, J., Backes, M., Gritzalis, S., Preneel, B. (eds.) ISC'06. LNCS, vol. 4176, pp. 171–186. Springer (2006)
17. De Cannière, C., Dunkelman, O., Knežević, M.: KATAN and KTANTAN - A Family of Small and Efficient Hardware-Oriented Block Ciphers. In: Clavier, C., Gaj, K. (eds.) CHES'09. LNCS, vol. 5747, pp. 272–288. Springer (2009)
18. De Cannière, C., Preneel, B.: Trivium. In: Robshaw and Billet [42], pp. 244–266
19. Duc, A., Guo, J., Peyrin, T., Wei, L.: Unaligned Rebound Attack - Application to Keccak. Cryptology ePrint Archive, Report 2011/420 (2011), `http://eprint.iacr.org/2011/420`
20. Gauravaram, P., Knudsen, L.R., Matusiewicz, K., Mendel, F., Rechberger, C., Schläffer, M., Thomsen, S.S.: Grøstl – a SHA-3 candidate. Submission to NIST (Round 3) (2011), `http://www.groestl.info/Groestl.pdf`
21. Guo, J., Peyrin, T., Poschmann, A.: The PHOTON Family of Lightweight Hash Functions. In: Rogaway [43], pp. 222–239
22. Hein, D.M., Wolkerstorfer, J., Felber, N.: ECC Is Ready for RFID - A Proof in Silicon. In: Avanzi, R.M., Keliher, L., Sica, F. (eds.) SAC'08. LNCS, vol. 5381, pp. 401–413. Springer (2008)
23. Hell, M., Johansson, T., Maximov, A., Meier, W.: The Grain Family of Stream Ciphers. In: Robshaw and Billet [42], pp. 179–190
24. Hell, M., Johansson, T., Meier, W.: Grain: a stream cipher for constrained environments. IJWMC 2(1), 86–93 (2007)
25. Henzen, L., Aumasson, J.P., Meier, W., Phan, R.C.W.: VLSI Characterization of the Cryptographic Hash Function BLAKE. `http://131002.net/data/papers/HAMP10.pdf` (2010)
26. Henzen, L., Aumasson, J.P., Meier, W., Phan., R.C.W.: VLSI Characterization of the Cryptographic Hash Function BLAKE (2010), available at `http://131002.net/data/papers/HAMP10.pdf`

27. Hong, D., Sung, J., Hong, S., Lim, J., Lee, S., Koo, B., Lee, C., Chang, D., Lee, J., Jeong, K., Kim, H., Kim, J., Chee, S.: HIGHT: A New Block Cipher Suitable for Low-Resource Device. In: Goubin, L., Matsui, M. (eds.) CHES'06. LNCS, vol. 4249, pp. 46–59. Springer (2006)
28. Indesteege, S.: The LANE hash function. Submission to NIST (2008), `http://www.cosic.esat.kuleuven.be/publications/article-1181.pdf`
29. Kavun, E., Yalcin, T.: A Lightweight Implementation of Keccak Hash Function for Radio-Frequency Identification Applications. In: Ors Yalcin, S. (ed.) Radio Frequency Identification: Security and Privacy Issues, LNCS, vol. 6370, pp. 258–269. Springer Berlin / Heidelberg (2010)
30. Khovratovich, D., Naya-Plasencia, M., Röck, A., Schläffer, M.: Cryptanalysis of *Luffa* v2 Components. In: Biryukov, A., Gong, G., Stinson, D.R. (eds.) Selected Areas in Cryptography. LNCS, vol. 6544, pp. 388–409. Springer (2010)
31. Kim, M., Ryou, J.: Power Efficient Hardware Architecture of SHA-1 Algorithm for Trusted Mobile Computing. In: Proceedings of the 9th international conference on Information and communications security. pp. 375–385. ICICS'07, Springer (2007)
32. Kim, M., Ryou, J., Jun, S.: Efficient Hardware Architecture of SHA-256 Algorithm for Trusted Mobile Computing. In: Yung, M., Liu, P., Lin, D. (eds.) Inscrypt. LNCS, vol. 5487, pp. 240–252. Springer (2008)
33. Leander, G.: On linear hulls, statistical saturation attacks, present and a cryptanalysis of puffin. In: Paterson, K.G. (ed.) EUROCRYPT'11. LNCS, vol. 6632, pp. 303–322. Springer (2011)
34. Leander, G., Abdelraheem, M.A., AlKhzaimi, H., Zenner, E.: A Cryptanalysis of PRINTcipher: The Invariant Subspace Attack. In: Rogaway [43], pp. 206–221
35. Leander, G., Paar, C., Poschmann, A., Schramm, K.: New Lightweight DES Variants. In: Biryukov, A. (ed.) FSE'07. LNCS, vol. 4593, pp. 196–210. Springer (2007)
36. Lim, C.H., Korkishko, T.: mCrypton - A Lightweight Block Cipher for Security of Low-Cost RFID Tags and Sensors. In: Song, J., Kwon, T., Yung, M. (eds.) WISA'05. LNCS, vol. 3786, pp. 243–258. Springer (2005)
37. Mangard, S., Standaert, F.X. (eds.): Cryptographic Hardware and Embedded Systems, CHES 2010, 12th International Workshop, Santa Barbara, CA, USA, August 17-20, 2010. Proceedings, LNCS, vol. 6225. Springer (2010)
38. Mendel, F., Rechberger, C., Schläffer, M., Thomsen, S.S.: The Rebound Attack: Cryptanalysis of Reduced Whirlpool and Grøstl. In: Dunkelman, O. (ed.) FSE'09. LNCS, vol. 5665, pp. 260–276. Springer (2009)
39. Merkle, R.: Secrecy, authentication and public key systems / A certified digital signature. Ph.D. thesis, Dept. of Electrical Engineering, Stanford University (1979)
40. NANGATE: The NanGate 45nm Open Cell Library, available at `http://www.nangate.com`
41. Osaka, K., Takagi, T., Yamazaki, K., Takahashi, O.: An Efficient and Secure RFID Security Method with Ownership Transfer. In: Wang, Y., Cheung, Y., Liu, H. (eds.) CIS. LNCS, vol. 4456, pp. 778–787. Springer (2006)
42. Robshaw, M.J.B., Billet, O. (eds.): New Stream Cipher Designs - The eSTREAM Finalists, LNCS, vol. 4986. Springer (2008)
43. Rogaway, P. (ed.): Advances in Cryptology - CRYPTO 2011 - 31st Annual Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2011. Proceedings, LNCS, vol. 6841. Springer (2011)
44. Rohde, S., Eisenbarth, T., Dahmen, E., Buchmann, J., Paar, C.: Fast Hash-Based Signatures on Constrained Devices. In: Grimaud, G., Standaert, F.X. (eds.) CARDIS'08. LNCS, vol. 5189, pp. 104–117. Springer (2008)
45. Shoufan, A.: An FPGA Accelerator for Hash Tree Generation in the Merkle Signature Scheme. In: Sirisuk, P., Morgan, F., El-Ghazawi, T.A., Amano, H. (eds.) ARC'10. LNCS, vol. 5992, pp. 145–156. Springer (2010)
46. Standaert, F.X., Piret, G., Gershenfeld, N., Quisquater, J.J.: SEA: A Scalable Encryption Algorithm for Small Embedded Applications. Presented at the Workshop on RFID and Light-Weight Crypto in Graz, Austria (2005)
47. Tillich, S., Feldhofer, M., Issovits, W., Kern, T., Kureck, H., Muehlberghuber, M., Neubauer, G., Reiter, A., Koefler, A., Mayrhofer, M.: Compact Hardware Implementations of the SHA-3 Candidates ARIRANG, BLAKE, Grøstl, and Skein. Cryptology ePrint Archive, Report 2009/349 (2009)
48. Tillich, S., Feldhofer, M., Issovits, W., Kern, T., Kureck, H., Mühlberghuber, M., Neubauer, G., Reiter, A., Köfler, A., Mayrhofer, M.: Compact Hardware Implementations of the SHA-3 Candidates ARIRANG, BLAKE, Grøstl, and Skein. Cryptology ePrint Archive, Report 2009/349 (2009), available at `http://eprint.iacr.org/2009/349`
49. Tsudik, G.: YA-TRAP: Yet Another Trivial RFID Authentication Protocol. In: PerCom Workshops. pp. 640–643. IEEE Computer Society (2006)
50. Van Assche, G.: Errata for Keccak presentation. E-mail sent to the NIST SHA-3 mailing list on Feb 7 2011, on behalf of the Keccak team (2011)
51. Yang, B., Wu, K., Karri, R.: Scan Based Side Channel Attack on Dedicated Hardware Implementations of Data Encryption Standard. International Test Conference pp. 339–344 (2004)