SPONGENT: A Lightweight Hash Function*

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. This paper proposes SPONGENT – a family of lightweight hash functions with hash sizes of 88 (for preimage resistance only), 128, 160, 224, and 256 bits based on a sponge construction instantiated with a PRESENT-type permutation, following the hermetic sponge strategy. Its smallest implementations in ASIC require 738, 1060, 1329, 1728, and 1950 GE, respectively. To our best knowledge, at all security levels attained, it is the hash function with the smallest footprint in hardware published so far, the parameter being highly technology dependent. SPONGENT offers a lot of flexibility in terms of serialization degree and speed. We explore some of its numerous implementation trade-offs.

We furthermore present a security analysis of SPONGENT. Basing the design on a PRESENT-type primitive provides confidence in its security with respect to the most important attacks. Several dedicated attack approaches are also investigated.

Keywords: 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

* 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.

B. Preneel and T. Takagi (Eds.): CHES 2011, LNCS 6917, pp. 312–325, 2011.

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 [12,10], Grain [13,14], and Mickey [2] as well as block ciphers like SEA [26], DESL, DESXL [21], HIGHT [16], mCrypton [22], KATAN/KTANTAN [11], and PRESENT [5] — to mention only a small selection of the lightweight designs.

Rather recently, some significant work on lightweight hash functions has been also performed: [6] describes ways of using the PRESENT block cipher in hashing modes of operation and [1] takes the approach of designing a dedicated lightweight hash function QUARK based on a sponge construction [9,3]. However, while for the stream and block ciphers, the designs have already closely approached the minimum ASIC hardware footprint theoretically attainable, it does not seem the case for lightweight hash functions so far. This paper illustrates this point by proposing the lightweight hash function SPONGENT with a considerably smaller footprint than SHA-2, SHA-3 finalists, PRESENT in hashing modes, and QUARK. Similarly to QUARK, a part of this advantage comes from a reduced level of preimage and second preimage security, while maintaining the standard level of collision resistance.

1.2 Design Considerations for a Lightweight Hash Function

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.

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.

As shown in [6], 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.

In a permutation-based sponge construction, let r be the *rate* (the number of bits input or output per one permutation call) and c be the *capacity* (internal state bits not used for input or output). The design of [1] as well as the works [3,4,9] 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, if the underlying permutation does not have any structural distinguishers (thus, the sponge construction being *hermetic*), 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 $2^{c/2}$. As in most embedded scenarios, where a lightweight hash function is likely to be used, the full second-preimage security is not a necessary requirement, we will take this approach in the design of SPONGENT. For relatively small rate r, the loss of preimage security is limited.

However, while using this novel idea of reducing the state size to minimize (1), the QUARK hash function does not appear to provide an optimal logic size, which is mainly due to the Boolean functions with many inputs used in its round transform. 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 to the output hash size n, we opt for 5 variants of SPONGENT covering most security applications in the field. SPONGENT-88 is designed for extremely restricted scenarios and low preimage security requirements. It 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 Paper

The remainder of the paper 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

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):

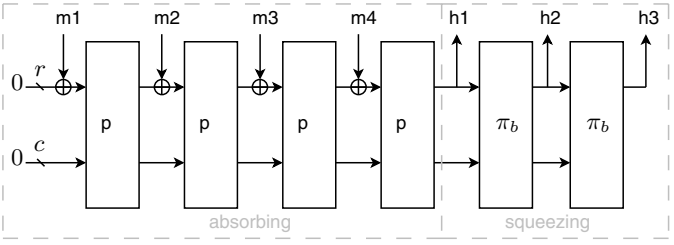

Fig. 1. Sponge construction based on a b-bit permutation π_b with capacity c bits and rate r bits. m_i are r-bit message blocks. h_i are parts of the hash value.

- **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, the hash size n equals capacity c. 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 all SPONGENT variants. Then the works [3,4,9] 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 $2^{n-r}+2^{c/2}$.

2.2 Parameters

We propose five variants of SPONGENT with five different security levels:

	n (bit)	b (bit)	c (bit)	r (bit)	R number of rounds	security(bit) preimage	2nd preimage	collision
SPONGENT-88	88	88	80	8	45	80	40	40
SPONGENT-128	128	136	128	8	70	120	64	64
SPONGENT-160	160	176	160	16	90	144	80	80
SPONGENT-224	224	240	224	16	120	208	112	112
SPONGENT-256	256	272	256	16	140	240	128	128

2.3 PRESENT-type Permutation

The permutation $\pi_b : \mathbb{F}_2^b \to \mathbb{F}_2^b$ 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 $\leftarrow$ rev(lCounter$_b(i)$) $\oplus$ STATE $\oplus$ lCounter$_b(i)$
  STATE $\leftarrow$ sBoxLayer$_b$(STATE)
  STATE $\leftarrow$ 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. 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. rev(lCounter$_b(i)$) is the value of lCounter$_b(i)$ with its bits in reversed order and is added to the leftmost bits of STATE.

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 S-box fulfills the PRESENT S-box criteria [5]. 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

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

3. lCounter$_b$: This is one of the three $\lceil \log_2 R \rceil$-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 6-bit LFSR used in SPONGENT-88 is defined by the primitive trinomial $\zeta^6 + \zeta^5 + 1$ (initialized with '000101'). The 7-bit LFSR with a primitive trinomial of $\zeta^7 + \zeta^6 + 1$ is used in SPONGENT-128, SPONGENT-160, and SPONGENT-224 and respectively initialized with '1111010', '1000101', and '0000001'. SPONGENT-256 uses an 8-bit LFSR based on the pentanomial $\zeta^8 + \zeta^4 + \zeta^3 + \zeta^2 + 1$ and it is initialized with '10011110'.

2.4 Design Rationale

Permutation. The 4-bit S-box is the major block of functional logic in a serial low-area implementation of SPONGENT, the bit permutation requiring some additional space in silicon. Its simplicity and small size minimize the area and power consumption on the logic side. The structures of the bit permutation and the S-box in SPONGENT make it possible to prove

Theorem 1. *Any 5-round differential characteristic of the underlying permutation in* SPONGENT-{*88, 128, 160, 224, 256*} *has a minimum of* 10 *active S-boxes.*

Proof. The statements for SPONGENT-{88, 128, 160, 224, 256} can directly be proven by applying the same technique used in [5, Appendix III].

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 for SPONGENT-$\{88, 128, 160, 224, 256\}$, 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 the currently known cryptanalytic attacks by applying the most important state-of-the-art methods of cryptanalysis and investigating their complexity.

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 [5]. More precisely, the results on the number of differentially active S-boxes over 5 rounds will hold for all SPONGENT variants which is reflected in Theorem 1.

For all SPONGENT variants, we found that those 5-round bounds are actually tight. We present the characteristics attaining them in Table 1 as well as in Appendix A.

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., $(0 \dots 0 || \Delta m_i) \xrightarrow{\pi} (0 \dots 0 || \Delta m_{i+1})$.

Table 1. Differential characteristics with lowest numbers of differentially active S-boxes (ASN). The probabilities are calculated assuming the independency of round computations.

# of	SPONGENT-88		SPONGENT-128		SPONGENT-160		SPONGENT-224		SPONGENT-256	
rounds	ASN	Prob	ASN	Prob	ASN	Prob	ASN	Prob	ASN	Prob
5	10	2^{-21}	10	2^{-22}	10	2^{-21}	10	2^{-21}	10	2^{-20}
10	20	2^{-47}	29	2^{-68}	20	2^{-50}	20	2^{-43}	–	–
15	30	2^{-74}	-	-	30	2^{-79}	30	2^{-66}	–	–

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 (see Appendix A).

The rebound attack [24], 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.

Dedicated Rebound Attack on 6 Rounds of SPONGENT-88. Here we describe a dedicated rebound attack on SPONGENT-88. For other hash sizes, a similar method is applicable. The path for this attack is shown in Figure 2.

- **Inbound phase:** In the forward direction, we start from the input of the second round. We generate 2^{28} structures as follows: We restrict the input of the active S-boxes to the eight values such that the difference $\Delta S : [\{\texttt{0x2}, \texttt{0x3}\} \rightarrow \texttt{0xf}]$ and to the four values such that the difference $\Delta S : [\texttt{0x4} \rightarrow \texttt{0xf}]$ and we generate the passive bits at random. For each fixed value of the passive part, we can obtain $(2^3)^2 \cdot (2^2)^2 = 2^{10}$ pairs, so we repeat the procedure 2^{18} times. At the input of round 3, we have 16 active S-boxes and 6 passive S-boxes, and it is guaranteed to have all S-boxes active at the input of round 4. In the backward direction, we start from the input of the S-boxes in round 6. Similarly, we generate 2^{28} structures by restricting the values of the active S-boxes to four values such that the difference $\Delta S^{-1} : [\texttt{0x1} \rightarrow \{\texttt{0xb}, \texttt{0xd}\}]$. Again, we can generate $(2^2)^{11} = 2^{22}$ pairs for each value of the passive part, hence we choose random 2^6 values. Then at the input of the fifth round all S-boxes are active and with high probability they will still be active after the permutation layer.
- **Merging phase:** We look for a matching input/output difference of the S-box layer in round 4 using the precomputed difference distribution table. We

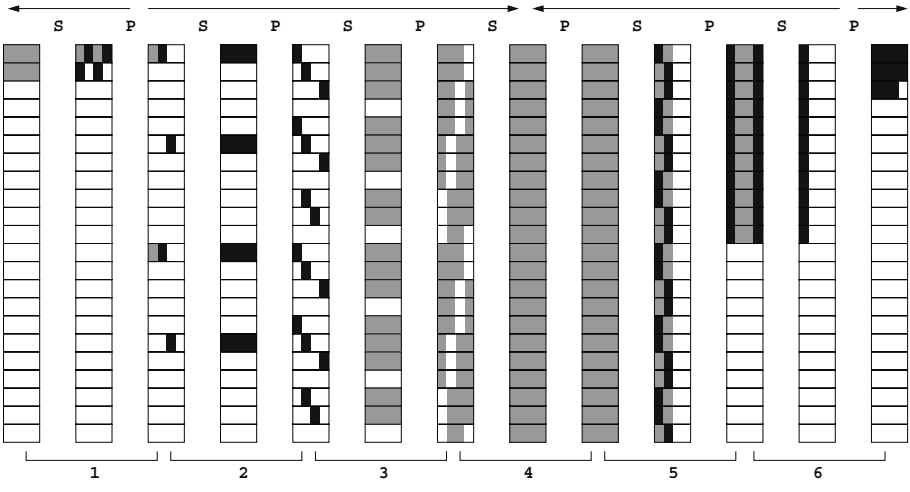

Fig. 2. Differential path for the rebound attack (black: starting S-boxes with conditions, grey: active S-boxes, S: sBoxLayer_{88}, P: pLayer_{88})

can find a match with probability of $2^{-2.4}$ for each word that has one fixed bit with zero difference and a match with probability $2^{3.6}$ for each word that has two fixed bit zero. Therefore, we can find an entire differential path with probability $(2^{-2.4})^{20} \cdot (2^{-3.6})^2 = 2^{-55.2}$. Hence we expect to find at least one solution.

– **Outbound phase:** We further extend the differential path backwards (from round 2 to round 1) and forwards (to the end of round 6) with probability 1.

3.3 Linear Attacks

The most successful attacks, the attacks that can break the highest number of rounds, for the block cipher PRESENT are attacks based on linear approximations. In particular the multi-dimensional linear attack [7] and the statistical saturation attack [8] claim to break up to 26 rounds. It was shown in [20] 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.

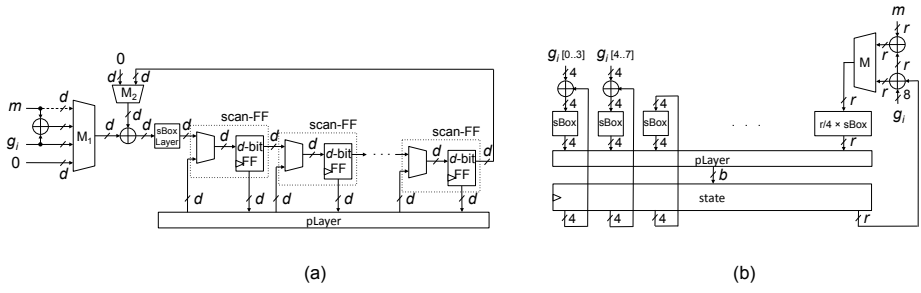

Fig. 3. Hardware architecture representing (a) serial datapath (b) parallel datapath of the SPONGENT variants

4 Hardware Implementations

Using the most serialized implementation, the hash functions SPONGENT-{88, 128, 160, 224, 256} can be implemented with 738 GE, 1060 GE, 1329 GE, 1728 GE, and 1950 GE, respectively, which is smaller than the most compact QUARK designs [1] of respective sizes. Furthermore, even the SPONGENT-256 hash function is more compact than s-QUARK having a hash output of 224 bits. 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 those of the corresponding QUARK variants.

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: for SPONGENT-{88, 128, 160} we implement $d \in \{4, 8, 16, 32\}$, $d \in \{4, 8, 16, 68\}$, $d \in \{4, 8, 16, 44, 88\}$, respectively, while for SPONGENT-{224, 256} we implement $d \in \{4, 8, 16, 32, 64\}$. An architecture representing our serialized datapath is depicted in Fig. 3(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 registers (6.25 GE in our library), which act as a combination of two input multiplexer and an ordinary register[1]. Instead of providing a reset signal to each register separately, we use two zero inputs at the multiplexers M_1 and M_2 to correctly initialize all the registers. This additionally reduces hardware resources, as the scan registers with a reset input approximately require 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.

Additionally, we implement all the SPONGENT variants as depicted in Fig. 3(b). Every round now requires a single clock cycle, therefore resulting in faster, yet rather compact designs.

[1] Scan registers are typically used to provide scan-chain based testability of the circuit. Due to the security issues of scan-chain based testing [28], other methods such as Built-In-Self-Test (BIST) are recommended for testing the cryptographic hardware.

Table 2. 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) Pre.	Coll.	2nd Pre.	Hash (bit)	Cycles	Datapath (bit)	Process (μm)	**Area** (GE)	**Throughput** (kbps)	**Power** (μW)
SPONGENT-88	80	40	40	88	990	4	0.13	738	0.81	1.57
					45	88	0.13	1127	17.78	2.31
SPONGENT-128	120	64	64	128	2380	4	0.13	1060	0.34	2.20
					70	136	0.13	1687	11.43	3.58
SPONGENT-160	144	80	80	160	3960	4	0.13	1329	0.40	2.85
					90	176	0.13	2190	17.78	4.47
SPONGENT-224	208	112	112	224	7200	4	0.13	1728	0.22	3.73
					120	240	0.13	2903	13.33	5.97
SPONGENT-256	240	128	128	256	9520	4	0.13	1950	0.17	4.21
					140	272	0.13	3281	11.43	6.62
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 [6]	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 [6]	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 [6]	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 [6]	192	96	192	192	3338	12	0.18	4600	1.90	-
					108	192	0.18	8048	59.26	9.31
KECCAK-f[400] [17]	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] [17]	128	64	128	128	900	8	0.13	2520	8.00	5.60
					18	8	0.13	4900	400.00	27.60
SHA-1 [18]	160	80	160	160	450	32	0.25	6812	113.78	11.00
SHA-256 [19]	256	128	256	256	490	32	0.25	8588	104.48	11.20
BLAKE [15]	256	128	256	256	816	32	0.18	13575	62.79	11.16
Grøstl [27]	256	128	256	256	196	64	0.18	14622	261.14	221.00

Next, we present our hardware figures of all 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 0.13 μm CMOS process provided by Faraday Technology Corporation (fsc0h_d_tc). We provide synthesis results only. Our reasoning follows a simple design decision: we provide a large design space, focusing on multitude of design choices and discuss in detail our implementation strategy. Therefore, we rather spend our efforts by exploring a large set of hardware designs than by performing a time-consuming place and route process for each of the design separately. Moreover, the physical size of the designs (in terms of gate equivalences) is expected to remain the same even after the place and route is performed. We expect slightly worse results only with respect to the overall power consumption.

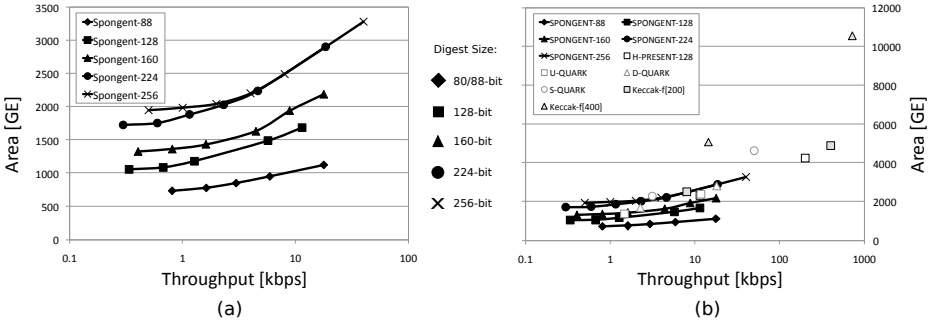

Fig. 4. (a) Area versus throughput trade-off of the SPONGENT hash family. (b) Comparison with state-of-the-art lightweight hash functions.

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 2 reports hardware figures obtained using the aforementioned methodology. Besides having a very small footprint, another remarkable result is that the most serialized versions of SPONGENT-{88, 128, 160, 224, 256} are built of 89.3%, 92.5%, 93.8%, 95%, and 96% sequential logic, respectively. 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 4(a) illustrates the wide spectrum of our explored design space, where a typical trade-off between speed and area is scrutinized. Using the same metrics, we compare our design with state-of-the-art lightweight hash functions (Fig. 4(b)). For the same level of security, the SPONGENT family tends to require much smaller area than its counterparts.

5 Conclusion

In this paper, we have proposed the family of lightweight hash functions SPONGENT with hash sizes 88, 128, 160, 224, and 256 bits. Its serialized implementations in ASIC hardware require 738, 1060, 1329, 1728, and 1950 GE, respectively. Thus, SPONGENT has the smallest footprint among all hash functions published so far at all security levels it attains, though area requirements are highly dependent on technology used.

References

1. Aumasson, J.P., Henzen, L., Meier, W., Naya-Plasencia, M.: Quark: A lightweight hash. In: Mangard and Standaert [23], pp. 1–15
2. Babbage, S., Dodd, M.: The MICKEY Stream Ciphers. In: Robshaw and Billet [25], pp. 191–209
3. Bertoni, G., Daemen, J., Peeters, M., Assche, G.V.: On the Indifferentiability of the Sponge Construction. In: Smart, N.P. (ed.) EUROCRYPT 2008. LNCS, vol. 4965, pp. 181–197. Springer, Heidelberg (2008)
4. Bertoni, G., Daemen, J., Peeters, M., Assche, G.V.: Sponge-Based Pseudo-Random Number Generators. In: Mangard and Standaert [23], pp. 33–47
5. 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 2007. LNCS, vol. 4727, pp. 450–466. Springer, Heidelberg (2007)
6. 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 2008. LNCS, vol. 5154, pp. 283–299. Springer, Heidelberg (2008)
7. Cho, J.Y.: Linear Cryptanalysis of Reduced-Round PRESENT. In: Pieprzyk, J. (ed.) CT-RSA 2010. LNCS, vol. 5985, pp. 302–317. Springer, Heidelberg (2010)
8. Collard, B., Standaert, F.-X.: A Statistical Saturation Attack against the Block Cipher PRESENT. In: Fischlin, M. (ed.) CT-RSA 2009. LNCS, vol. 5473, pp. 195–210. Springer, Heidelberg (2009)
9. Daemen, J., Peeters, M., Assche, G.V.: Sponge Functions. In: Ecrypt Hash Workshop 2007 (2007), `http://www.csrc.nist.gov/pki/HashWorkshop/PublicComments/2007May.html`
10. De Cannière, C.: TRIVIUM: A Stream Cipher Construction Inspired by Block Cipher Design Principles. In: Katsikas, S.K., López, J., Backes, M., Gritzalis, S., Preneel, B. (eds.) ISC 2006. LNCS, vol. 4176, pp. 171–186. Springer, Heidelberg (2006)
11. 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 2009. LNCS, vol. 5747, pp. 272–288. Springer, Heidelberg (2009)
12. De Cannière, C., Preneel, B.: Trivium. In: Robshaw and Billet [25], pp. 244–266
13. Hell, M., Johansson, T., Maximov, A., Meier, W.: The Grain Family of Stream Ciphers. In: Robshaw and Billet [25], pp. 179–190
14. Hell, M., Johansson, T., Meier, W.: Grain: a stream cipher for constrained environments. IJWMC 2(1), 86–93 (2007)
15. Henzen, L., Aumasson, J.P., Meier, W., Phan, R.C.W.: LSI Characterization of the Cryptographic Hash Function BLAKE (2010), `http://131002.net/data/papers/HAMP10.pdf`
16. 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 2006. LNCS, vol. 4249, pp. 46–59. Springer, Heidelberg (2006)
17. Kavun, E., Yalcin, T.: A Lightweight Implementation of Keccak Hash Function for Radio-Frequency Identification Applications. In: Ors Yalcin, S.B. (ed.) RFIDSec 2010. LNCS, vol. 6370, pp. 258–269. Springer, Heidelberg (2010)

18. Kim, M., Ryou, J.: Power Efficient Hardware Architecture of SHA-1 Algorithm for Trusted Mobile Computing. In: Qing, S., Imai, H., Wang, G. (eds.) ICICS 2007. LNCS, vol. 4861, pp. 375–385. Springer, Heidelberg (2007)
19. 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 2008. LNCS, vol. 5487, pp. 240–252. Springer, Heidelberg (2009)
20. Leander, G.: On Linear Hulls, Statistical Saturation Attacks, PRESENT and a Cryptanalysis of PUFFIN (to appear, 2011)
21. Leander, G., Paar, C., Poschmann, A., Schramm, K.: New Lightweight DES Variants. In: Biryukov, A. (ed.) FSE 2007. LNCS, vol. 4593, pp. 196–210. Springer, Heidelberg (2007)
22. Lim, C.H., Korkishko, T.: mCrypton – A Lightweight Block Cipher for Security of Low-Cost RFID Tags and Sensors. In: Song, J.-S., Kwon, T., Yung, M. (eds.) WISA 2005. LNCS, vol. 3786, pp. 243–258. Springer, Heidelberg (2006)
23. Mangard, S., Standaert, F.-X. (eds.): CHES 2010. LNCS, vol. 6225. Springer, Heidelberg (2010)
24. Mendel, F., Rechberger, C., Schläffer, M., Thomsen, S.S.: The Rebound Attack: Cryptanalysis of Reduced Whirlpool and `Grøstl`. In: Dunkelman, O. (ed.) FSE 2009. LNCS, vol. 5665, pp. 260–276. Springer, Heidelberg (2009)
25. Robshaw, M.J.B., Billet, O. (eds.): New Stream Cipher Designs. LNCS, vol. 4986. Springer, Heidelberg (2008)
26. 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)
27. 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), `http://eprint.iacr.org/2009/349`
28. Yang, B., Wu, K., Karri, R.: Scan Based Side Channel Attack on Dedicated Hardware Implementations of Data Encryption Standard. In: International Test Conference, pp. 339–344 (2004)

A Some Differential Paths

Table 3. Sample differential paths for SPONGENT−{88, 128}

	SPONGENT-88		SPONGENT-128	
Round	Difference	Prob	Difference	Prob
0	0500000000000009000000	2^{-21}	00000000000000009000900000000000000	2^{-22}
5	00000004008000000000		00000000000080004000000000000000000	
0	900090000000000000000	2^{-47}	00000005500000000000033000000000000	2^{-68}
10	20000000000080000A0000		00000400400000001000000000000001010	
0	900090000000000000000	2^{-74}		
15	00000000801000000000			

Table 4. Sample differential paths for SPONGENT−{160, 224, 256}

SPONGENT-160		
Round	Difference	Prob
0	060000000600000000000000000000000000000000	2^{-21}
5	000000000000000004008000000000000000000000	
0	900000000009000000000000000000000000000000	2^{-50}
10	008000000000000000000002000000000A00000000	
0	000000000000000000000000000030003000000000	2^{-79}
15	000000000000000008010000000000000000000000	
SPONGENT-224		
Round	Difference	Prob
0	090000000000900000000000000000000000000000000000000000000000	2^{-21}
5	000000020004000000000000000000000000000000000000000000000000	
0	000000000000000000000000000000009000000000090000000000	2^{-43}
10	008000000000000000000000000000010000000000000090000000000	
0	000000001001000000000000000000000000000000000000000000000000	2^{-66}
15	008000000000000000000000000000010000000000000090000000000	
SPONGENT-256		
Round	Difference	Prob
0	000000000000000000000000000000000000000000000000000660000000000660	2^{-20}
5	000000002000100000000000000000000000000000000000000000000020001000	