J. Cryptol. (2013) 26: 313–339
DOI: 10.1007/s00145-012-9125-6

Journal of
CRYPTOLOGY

QUARK: A Lightweight Hash*

Jean-Philippe Aumasson
NAGRA, route de Genève 22, 1033 Cheseaux, Switzerland
jeanphilippe.aumasson@gmail.com

Luca Henzen†
UBS AG, Zürich, Switzerland

Willi Meier
FHNW, Windisch, Switzerland

María Naya-Plasencia‡
University of Versailles, Versailles, France

Communicated by Mitsuru Matsui

Received 29 September 2010
Online publication 10 May 2012

Abstract. The need for lightweight (that is, compact, low-power, low-energy) cryptographic hash functions has been repeatedly expressed by professionals, notably to implement cryptographic protocols in RFID technology. At the time of writing, however, no algorithm exists that provides satisfactory security and performance. The ongoing SHA-3 Competition will not help, as it concerns general-purpose designs and focuses on software performance. This paper thus proposes a novel design philosophy for lightweight hash functions, based on the sponge construction in order to minimize memory requirements. Inspired by the stream cipher Grain and by the block cipher KATAN (amongst the lightest secure ciphers), we present the hash function family QUARK, composed of three instances: U-QUARK, D-QUARK, and S-QUARK. As a sponge construction, QUARK can be used for message authentication, stream encryption, or authenticated encryption. Our hardware evaluation shows that QUARK compares well to previous tentative lightweight hash functions. For example, our lightest instance U-QUARK conjecturally provides at least 64-bit security against all attacks (collisions, multicollisions, distinguishers, etc.), fits in 1379 gate-equivalents, and consumes on average 2.44 μW at 100 kHz in 0.18 μm ASIC. For 112-bit security, we propose S-QUARK, which can be implemented with 2296 gate-equivalents with a power consumption of 4.35 μW.

* Extended version of an article appearing at CHES 2010. The specification of QUARK given in this version differs from that in the CHES 2010 proceedings, namely, the parameter n has been increased to address a flaw in the initial analysis (as reported in [59]). This work was partially supported by European Commission through the ICT programme under contract ICT-2007-216676 ECRYPT II.

† This work was done when the second author was with ETHZ, Switzerland.

‡ This work was done when the fourth author was with FHNW, Switzerland.

Key words. Hash functions, Lightweight cryptography, Sponge functions, Cryptanalysis, Indifferentiability.

1. Introduction

Known as cryptographers' Swiss Army Knife, hash functions can serve many different purposes, within applications ranging from digital signatures and message authentication codes to secure passwords storage, key derivation, or forensics data identification. It is fair to say that any system that uses some sort of cryptography includes a hash function. These systems include resource-constrained devices implementing cryptographic functions as hardware blocks, such as RFID tags or systems-on-chip for lightweight embedded devices.

In 2006, Feldhofer and Rechberger [35] pointed out the lack of lightweight hash functions for use in RFID protocols, and gave recommendations to encourage the design of such primitives. The situation has not evolved much in four years, despite a growing demand; besides RFID protocols, lightweight hashes are indeed necessary in all applications that need to minimize the amount of hardware and the power and energy consumption.

Despite the need for lightweight hash functions, a dedicated approach to create secure and efficient algorithms remains to be found. New designs are thus of clear practical interest. In this paper, we address this problem and present a novel approach to design lightweight hashes, illustrated with the proposal of a new family of functions, called QUARK.

We expose our design philosophy in Sect. 2, before a complete specification of QUARK in Sect. 3. Then, Sect. 4 presents the rationale behind the QUARK design, and Sect. 5 reports on our preliminary security analysis. Our hardware implementation is presented in Sect. 6.

Related Works The SHA-3 Competition [52] aims to develop a general-purpose hash function, and received as many as 64 original and diverse submissions. Most of them, however, cannot reasonably be called lightweight, as most need more than (say) 10 000 gate equivalents (GE). An exception is CubeHash [11], which can be implemented with 7630 GE in 0.13 µm ASIC [8] to produce digests of up to 512 bits. For comparison, Feldhofer and Wolkerstorfer [36] reported an implementation of MD5 (128-bit digests, 0.35 µm ASIC) with 8001 GE, O'Neill [53] implemented SHA-1 (160-bit digests, 0.18 µm ASIC) with 6122 GE, and the compression function MAME by Yoshida et al. [61] (256-bit digests, 0.18 µm ASIC) fits in 8100 GE. These designs, however, are still too demanding for many low-end devices.

A step towards lightweight hashing is the 2008 work by Bogdanov et al. [23], which presented constructions based on the lightweight block cipher PRESENT [22]. They proposed to instantiate the Davies–Meyer construction (i.e., $E_m(h) \oplus h$, where $E_m(h)$ denotes the encryption of h with key m by the block cipher E) with PRESENT-80, giving a hash function with 64-bit digests. This hash function, called DM-PRESENT, was implemented with 1600 GE in 0.18 µm ASIC.

Another interesting approach was taken with Shamir's SQUASH [57] keyed hash function, which processes short strings only, offers 64-bit preimage resistance, and is expected to need fewer than 1000 GE. However, SQUASH is not collision resistant—as it targets RFID authentication protocols where collision resistance is unnecessary—and so is inappropriate for applications requiring a collision-resistant hash function.

In 2010, reduced versions of the hash function KECCAK (finalist in the SHA-3 Competition) were proposed [14]. For example, a version of KECCAK returning 64-bit digests was implemented with 2520 GE in 0.13 μm ASIC [45].

After the first publication of QUARK at CHES 2010 [5], other lightweight hash designs appeared, based on the sponge construction. These include PHOTON (presented at CRYPTO 2011 [41]) and SPONGENT (presented at CHES 2011 [24]).

At the time of writing, we have not been informed of any third-party results improving on our preliminary security analysis.

2. Design Philosophy

As noted in [23, Sect. 2], designers of lightweight cryptographic algorithms or protocols have to trade off between two opposite design philosophies. The first consists in creating new schemes from scratch, whereas the second consists in reusing available schemes and adapting them to system constraints. While Bogdanov et al. [23] are more in line with the latter approach—as illustrated by their DM-PRESENT proposal—we tend more towards the former.

Although QUARK borrows components from previous works, it integrates a number of innovations that make it unique and that optimize its lightweightness. As explained in this section, QUARK combines

- A sponge construction with a capacity c equal to the digest length n,
- A core permutation inspired by previous primitives, optimized for reduced resources consumption.

We introduce this design strategy as an attempt to optimize its security-performance ratio. Subsequent proposals of lightweight hash functions followed a similar strategy, with PHOTON and SPONGENT respectively building their core permutations on AES- and SERPENT-like algorithms.

2.1. *Separating Digest Length and Security Level*

We observe that the digest length of a hash function has generally been identified with its security level, with (say) n-bit digests being equivalent to n-bit security against preimage attacks. However, this rule restricts the variety of designs, as it forces designers to exclude design paradigms that may otherwise increase usability or performance.

The notion of *capacity*, introduced in the context of sponge functions [13], was a first step towards a separation of digest length and security level, and thus towards more inventive designs (as showed, by the hash family RADIOGATÚN [12]). In particular, the necessity of n-bit (second) preimage resistance is questionable from a pragmatic

standpoint, when one needs to assume that $2^{n/2}$ is an infeasible effort, to avoid birthday collision search. Designers may thus relax the security requirements against (second) preimages—as informally suggested by several researchers in the context of the SHA-3 Competition—so as to propose more efficient algorithms. For example, in [10] the designer of the SHA-3 candidate CubeHash [11] proposed instances with suboptimal preimage resistance (i.e., below 2^n) for efficiency purposes. We believe that lightweight hashes would benefit from separating digest length and security level. For this, we use a *sponge construction* and target a single security level against all attacks, including second preimage attacks, collision attacks, and any differentiating attack (although higher, optimal resistance of 2^n is offered against preimage attacks).

2.2. *Working with Shift Registers*

Shift registers are a well-known construction in digital circuits, generally implemented as a simple cascade of flip-flops. In cryptography, linear or nonlinear feedback shift registers have been widely used as a building block of stream ciphers, thanks to their simplicity and efficiency of implementation (be it in terms of area or power consumption).

In the design of QUARK, we opt for an algorithm based on bit shift registers combined with (nonlinear) Boolean functions, rather than for a design based on S-boxes combined with a linear layer (as PHOTON and SPONGENT). This is motivated by the simplicity of description and of implementation, and by the close-to-optimal area requirements it induces. Indeed, the register serves both to store the internal state (mandatory in any construction) and to perform the operations bringing confusion and diffusion. The only extra circuit is devoted to the implementation of the feedback functions, which combines bits from the registers to compute the new bit fed into the register.

Since good shift register-based algorithms are known, we do not reinvent the wheel and propose a core algorithm inspired from the stream cipher family Grain [43,44] and from the block cipher family KATAN [30], which are arguably the lightest known secure stream cipher and block cipher. Although both these designs are inappropriate for direct reuse in a hash function, both contain excellent design ideas, which we integrate in our lightweight hash QUARK. A goal of this best-of-both approach is to build on solid foundations while at the same time adapting the algorithm to the attack model of a hash function.

To summarize, our approach is not to instantiate classical general-purpose constructions with lightweight components, but rather to make the whole design lightweight by optimizing all its parts: security level, construction, and core algorithm. An outcome of this design philosophy, the hash family QUARK, is described in the next section.

3. Description of the QUARK Hash Family

This section gives a complete specification of QUARK and of its three proposed instances: U-QUARK, D-QUARK, and S-QUARK. In particle physics, the u-quark is lighter than the d-quark, which itself is lighter than the s-quark; our eponym hash functions compare similarly.

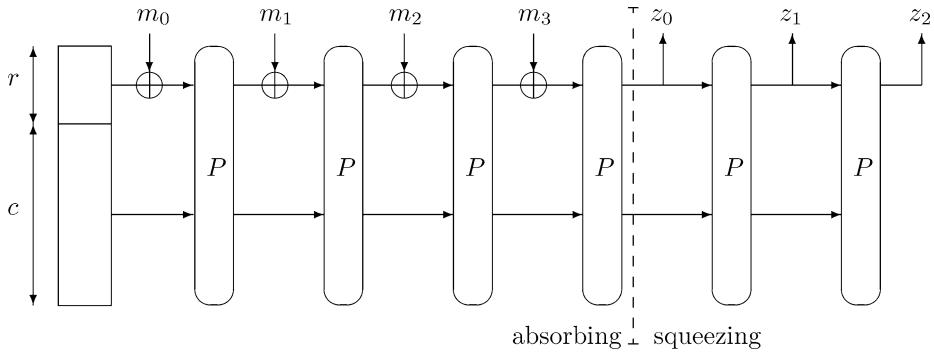

Fig. 1. The sponge construction as used by QUARK, for the example of a 4-block (padded) message.

3.1. *Sponge Construction*

QUARK uses the sponge construction, depicted in Fig. 1, and a b-bit permutation P (that is, a bijective function over $\{0,1\}^b$).

Following the notations introduced in [13], a QUARK instance is parametrized by a *rate* (or block length) r, a *capacity* c, and an *output length* n. The *width* $b = r + c$ of a sponge construction is the size of its internal state. We denote this internal state $s = (s_0, \ldots, s_{b-1})$, where s_0 is referred to as the *first* bit of the state.

Given a predefined initial state of b bits (specified for each instance of QUARK in Appendix A), the sponge construction processes a message m in three steps:

1. **Initialization**: the message is padded by appending a '1' bit followed by the minimal (possibly zero) number of '0' bits to reach a length that is a multiple of r.
2. **Absorbing phase**: the r-bit message blocks are XOR'd with the last r bits of the state (that is, $s_{b-r}, \ldots, s_{b-2}, s_{b-1}$), interleaved with applications of the permutation P. The absorbing phase starts with an XOR between the first block and the state, and it finishes with a call to the permutation P.
3. **Squeezing phase**: the last r bits of the state are returned as output, interleaved with applications of the permutation P, until n bits are returned. The squeezing phase starts with the extraction of r bits, and also finishes with the extraction of r bits.

3.2. *Permutation*

QUARK uses a permutation denoted P, inspired by the stream cipher Grain and by the block cipher KATAN (see Sect. 4.3 for details).

As depicted in Fig. 2, the internal state of P is viewed as three feedback shift registers (FSRs) two nonlinear ones (NFSRs) of $b/2$ bits each, and a linear one (LFSR) of $\lceil \log 4b \rceil$ bits. The state at epoch $t \geq 0$ is thus composed of

- An NFSR X of $b/2$ bits, denoted $X^t = (X_0^t, \ldots, X_{b/2-1}^t)$;
- An NFSR Y of $b/2$ bits, denoted $Y^t = (Y_0^t, \ldots, Y_{b/2-1}^t)$;
- An LFSR L of $\lceil \log 4b \rceil$ bits, denoted $L^t = (L_0^t, \ldots, L_{\lceil \log 4b \rceil - 1}^t)$.

Given a b-bit input, P proceeds in three stages, as described below.

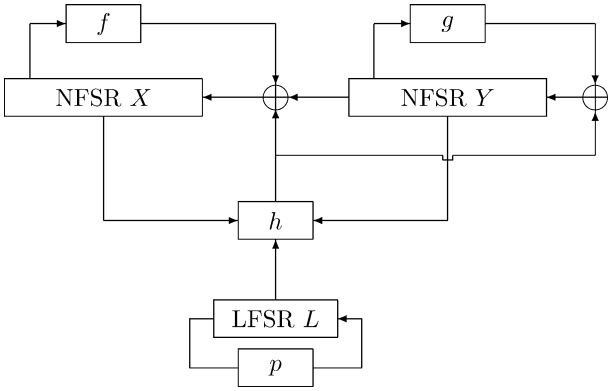

Fig. 2. Diagram of the permutation of QUARK.

3.2.1. *Initialization*

Upon input of the b-bit internal state of the sponge construction $s = (s_0, \ldots, s_{b-1})$, P initializes its internal state as follows:

- X is initialized with the first $b/2$ input bits: $(X_0^0, \ldots, X_{b/2-1}^0) := (s_0, \ldots, s_{b/2-1})$.
- Y is initialized with the last $b/2$ input bits: $(Y_0^0, \ldots, Y_{b/2-1}^0) := (s_{b/2}, \ldots, s_{b-1})$.
- L is initialized to the all-one string: $(L_0^0, \ldots, L_{\lceil \log 4b \rceil - 1}^0) := (1, \ldots, 1)$.

3.2.2. *State Update*

From an internal state (X^t, Y^t, L^t), the next state $(X^{t+1}, Y^{t+1}, L^{t+1})$ is determined by clocking the internal mechanism as follows:

1. The function h is evaluated upon input bits from X^t, Y^t, and L^t, and the result is written h^t:

$$h^t := h\big(X^t, Y^t, L^t\big).$$

2. X is clocked using Y_0^t, the function f, and h^t:

$$\big(X_0^{t+1}, \ldots, X_{b/2-1}^{t+1}\big) := \big(X_1^t, \ldots, X_{b/2-1}^t, Y_0^t + f\big(X^t\big) + h^t\big).$$

3. Y is clocked using the function g and h^t:

$$\big(Y_0^{t+1}, \ldots, Y_{b/2-1}^{t+1}\big) := \big(Y_1^t, \ldots, Y_{b/2-1}^t, g\big(Y^t\big) + h^t\big).$$

4. L is clocked using the function p:

$$\big(L_0^{t+1}, \ldots, L_{\lceil \log 4b \rceil}^{t+1}\big) := \big(L_1^t, \ldots, L_{\lceil \log 4b \rceil - 1}^t, p\big(L^t\big)\big).$$

Table 1. Parameters of the proposed instances of QUARK.

Instance	Rate (r)	Capacity (c)	Width (b)	Rounds ($4b$)	Digest (n)
U-QUARK	8	128	136	544	136
D-QUARK	16	160	176	704	176
S-QUARK	32	224	256	1024	256

3.2.3. *Computation of the Output*

Once initialized, the state of QUARK is updated $4b$ times. The output is defined as the final value of the NFSRs X and Y, using the same bit ordering as for the initialization. That is, the new internal state of the sponge construction is set to

$$s = (s_0, \ldots, s_{b-1}) = \left(X_0^{4b}, X_1^{4b}, \ldots, Y_{b/2-2}^{4b}, Y_{b/2-1}^{4b}\right).$$

3.3. *Proposed Instances*

We propose three different flavors of QUARK: U-QUARK, D-QUARK, and S-QUARK. For each, we give its rate r, capacity c, width b, digest length n, and its functions f, g, and h. For all flavors of QUARK, we have $\lceil \log 4b \rceil = 10$, thus the data-independent LFSR L is of 10 bits. The function p, used by L, is the same for all three instances: given a register L, p returns $L_0 + L_3$.

Table 1 summarizes the parameters of the three instances proposed.

U-QUARK is the lightest flavor of QUARK. It was designed to provide 128-bit preimage resistance and at least 64-bit security against all other attacks, and to admit a parallelization degree of 8. It has parameters $r = 8, c = 128, b = 136, n = 136$.

Function f Given a 68-bit register X, f returns

$$\begin{aligned} &X_0 + X_9 + X_{14} + X_{21} + X_{28} + X_{33} + X_{37} + X_{45} + X_{50} + X_{52} + X_{55} \\ &\quad + X_{55}X_{59} + X_{33}X_{37} + X_9X_{15} + X_{45}X_{52}X_{55} + X_{21}X_{28}X_{33} \\ &\quad + X_9X_{28}X_{45}X_{59} + X_{33}X_{37}X_{52}X_{55} + X_{15}X_{21}X_{55}X_{59} \\ &\quad + X_{37}X_{45}X_{52}X_{55}X_{59} + X_9X_{15}X_{21}X_{28}X_{33} + X_{21}X_{28}X_{33}X_{37}X_{45}X_{52}. \end{aligned}$$

Function g Given a 68-bit register Y, g returns

$$\begin{aligned} &Y_0 + Y_7 + Y_{16} + Y_{20} + Y_{30} + Y_{35} + Y_{37} + Y_{42} + Y_{49} + Y_{51} + Y_{54} \\ &\quad + Y_{54}Y_{58} + Y_{35}Y_{37} + Y_7Y_{15} + Y_{42}Y_{51}Y_{54} + Y_{20}Y_{30}Y_{35} \\ &\quad + Y_7Y_{30}Y_{42}Y_{58} + Y_{35}Y_{37}Y_{51}Y_{54} + Y_{15}Y_{20}Y_{54}Y_{58} \\ &\quad + Y_{37}Y_{42}Y_{51}Y_{54}Y_{58} + Y_7Y_{15}Y_{20}Y_{30}Y_{35} + Y_{20}Y_{30}Y_{35}Y_{37}Y_{42}Y_{51}. \end{aligned}$$

Function h Given 68-bit registers X and Y, and a 10-bit register L, h returns

$$\begin{aligned} &L_0 + X_1 + Y_2 + X_4 + Y_{10} + X_{25} + X_{31} + Y_{43} + X_{56} + Y_{59} \\ &\quad + Y_3X_{55} + X_{46}X_{55} + X_{55}Y_{59} + Y_3X_{25}X_{46} + Y_3X_{46}X_{55} \\ &\quad + Y_3X_{46}Y_{59} + L_0X_{25}X_{46}Y_{59} + L_0X_{25}. \end{aligned}$$

D-QUARK is the second-lightest flavor of QUARK. It was designed to provide 160-bit preimage resistance and at least 80-bit security against all other attacks, and to admit a parallelization degree of 8. It has parameters $r = 16$, $c = 160$, $b = 176$, $n = 176$.

Function f D-QUARK uses the same function f as U-QUARK, but with taps 0, 11, 18, 19, 27, 36, 42, 47, 58, 64, 67, 71, 79 instead of 0, 9, 14, 15, 21, 28, 33, 37, 45, 50, 52, 55, 59, respectively.

Function g D-QUARK uses the same function g as U-QUARK, but with taps 0, 9, 19, 20, 25, 38, 44, 47, 54, 63, 67, 69, 78 instead of 0, 7, 15, 16, 20, 30, 35, 37, 42, 49, 51, 54, 58, respectively.

Function h Given 88-bit registers X and Y, and a 10-bit register L, h returns

$$\begin{aligned} &L_0 + X_1 + Y_2 + X_5 + Y_{12} + Y_{24} + X_{35} + X_{40} + X_{48} + Y_{55} \\ &\quad + Y_{61} + X_{72} + Y_{79} + Y_4X_{68} + X_{57}X_{68} + X_{68}Y_{79} + Y_4X_{35}X_{57} \\ &\quad + Y_4X_{57}X_{68} + Y_4X_{57}Y_{79} + L_0X_{35}X_{57}Y_{79} + L_0X_{35}. \end{aligned}$$

S-QUARK is the heaviest flavor of QUARK. It was designed to provide 224-bit preimage resistance and at least 112-bit security against all other attacks, and to admit a parallelization degree of 16. It has parameters $r = 32$, $c = 224$, $b = 256$, $n = 256$.

Function f S-QUARK uses the same function f as U-QUARK, but with taps 0, 16, 26, 28, 39, 52, 61, 69, 84, 94, 97, 103, 111 instead of 0, 9, 14, 15, 21, 28, 33, 37, 45, 50, 52, 55, 59, respectively.

Function g S-QUARK uses the same function f as U-QUARK, but with taps 0, 13, 28, 30, 37, 56, 65, 69, 79, 92, 96, 101, 109 instead of 0, 7, 15, 16, 20, 30, 35, 37, 42, 49, 51, 54, 58, respectively.

Function h Given 128-bit registers X and Y, and a 10-bit register L, h returns

$$\begin{aligned} &L_0 + X_1 + Y_3 + X_7 + Y_{18} + Y_{34} + X_{47} + X_{58} + Y_{71} + Y_{80} + X_{90} + Y_{91} \\ &\quad + X_{105} + Y_{111} + Y_8X_{100} + X_{72}X_{100} + X_{100}Y_{111} + Y_8X_{47}X_{72} + Y_8X_{72}X_{100} \\ &\quad + Y_8X_{72}Y_{111} + L_0X_{47}X_{72}Y_{111} + L_0X_{47}. \end{aligned}$$

3.4. *Keying* QUARK

As a sponge function, all results known on the sponge construction apply to QUARK. This includes proofs of security for keyed modes of operation, as described in [16,17]. A keyed sponge function processes its input by simply hashing the string composed of the key followed by the said input. The following primitives can then be realized:

- Message authentication code (MAC);
- Pseudorandom generator;
- Stream cipher;
- Random-access stream cipher;
- Key derivation function.

Furthermore, the QUARK instances can easily be modified to operate in the *duplex construction* (a variant of the sponge construction [18]), to allow the realization of functionalities as authenticated encryption or reseedable pseudorandom generators.

4. Design Rationale

This section explains why we opted for a sponge construction and how we chose the internals of the P permutation.

4.1. *Sponge Construction*

The sponge construction [13] is arguably the only real alternative to the classical Merkle–Damgård (MD) construction based on a compression function. Most other known constructions are indeed patched versions of MD, with larger internal state, prefix-free encoding, finalization functions, etc. [7,19,27].

Rather than a (non-injective) compression function, the sponge construction can rely on a *single unkeyed permutation*, and message blocks are integrated with a simple XOR in the internal state. Sponge functions do not require storage of message blocks or of "feedforward" intermediate values as in Davies–Meyer constructions. Nevertheless, the sponge construction needs a larger state to achieve traditional security levels, which partially compensates those memory savings.

The sponge construction was proven to be indifferentiable from a random oracle (up to a bound of approximately $\sqrt{\pi}2^{c/2}$) when instantiated with a random permutation or transformation [13], which is the highest security level a hash construction can achieve. But its most interesting feature is its flexibility: given a fixed permutation P, varying the parameters r, c, and n offers a wide range of trade-offs efficiency/security. This is well illustrated by the interactive page "Tune KECCAK to your requirements" at http://keccak.noekeon.org/tune.html.

Note that during the absorbing phase of QUARK, message blocks are XOR'd to the *last r bits* of the internal state, that is, to the last bits of the Y register. This provides a better diffusion than if the first r bits were used, because differences introduced in the last bits remain in the register, while those in the first quickly disappear due to the bit shifts. During the squeezing phase, digest bits are also extracted from the *last r* bits of the state. The motivation is simple: these are the last bits computed by the permutation; extracting from the first bits would make the computation of the last rounds useless.

4.2. *Separating Digest Length and Security Level*

An originality of QUARK is that its expected security level against (second) preimages differs from its digest length (see Sect. 5.1.2 for a description of the generic attack). In particular, the sponge construction, as used in QUARK, offers a similar security against generic collision attacks and generic second preimage attacks of approximately $2^{c/2}$, and a preimage resistance of approximately 2^c (that is, of $2^{n-r} = 2^c$).

A disadvantage of this approach is that one "wastes" half the digest bits, as far as second preimage resistance is concerned. However, this little penalty brings dramatic performance gains, for it reduces memory requirements by about 50 % compared to classical designs with a same security level. For instance, U-QUARK provides 64-bit security against collisions and second preimages using memory for 146 bits (i.e., the two NFSRs plus the LFSR), while DM-PRESENT provides 64-bit security against preimages but only 32-bit security against collisions with 128 bits of required memory.

Furthermore, the choice of a single security level against all attacks is less confusing for users, who may not be able to determine the security property required for each particular protocol, and then to evaluate the security of the hash function with respect to that property. QUARK provides a single security bound against all attacks, including length extension attacks, multicollision attacks, etc., with increased preimage resistance of 2^c.

4.3. *Permutation Algorithm*

We now justify the choices made to design P. First, we chose an algorithm based on shift registers rather than on S-boxes because in the latter one needs to implement circuits for several Boolean functions (to represent an S-box), rather than a single one in a (serial) implementation of a shift register. Moreover, S-box-based designs typically include a linear transform, which, though cheap to implement, is not necessary in a shift register-based design (as diffusion is performed by the bit shifts within the register). Algorithms based on shift registers also tend to be easier to scale and to implement. To avoid "reinventing the wheel", we borrowed most design ideas from the stream cipher Grain and from the block cipher KATAN, as detailed below.

4.3.1. *Grain*

The Grain family of stream ciphers is composed of Grain-v1, Grain-128, and Grain-128a. The stream cipher Grain-v1 [44] was chosen in 2008 as one of the four "promising new stream ciphers" by the ECRYPT eSTREAM Project. It consists of two 80-bit shift registers combined with three Boolean functions, which makes it one of the lightest designs ever: Good and Benaissa [40] reported an implementation in 0.18 µm ASIC with 1294 GE, for a power consumption of 3.3 µm at 100 kHz. Grain-128 [43] is the 128-bit instance of the Grain family, with 128-bit registers, 128-bit keys, and different Boolean functions. In 2011, a new member of the Grain family was proposed: Grain-128a [1] is an improved version of Grain-128 that incorporates (optional) authentication and countermeasures against known attacks (asymmetric padding, higher nonlinearity).

The main advantages of the Grain ciphers are their simplicity and their performance flexibility (due to the possibility of parallelized implementations). However, a direct reuse of Grain fails to give a secure permutation for a hash function because of "slide

distinguishers" (see Sect. 5.4), of the existence of differential characteristics [29], and of (conjectured) statistical distinguishers for Grain-128 [3,47]. Furthermore, the full Grain-128 can be attacked using advanced cube attacks [32,33]. At the time of writing, no third-party attack on Grain-128a has been published.

4.3.2. *KATAN*

The block cipher family KATAN [30] is inspired by the stream cipher Trivium [28] and builds a keyed permutation with two NFSRs combined with two light quadratic Boolean functions. Its small block sizes (32, 48, and 64 bits) plus the possibility of "burnt-in key" (with the KTANTAN family) lead to very small hardware footprints: 802 GE for KATAN32, and 462 GE for KTANTAN32 [30]. Published third-party cryptanalysis includes shortcut attacks on KTANTAN (unapplicable to KATAN) [21,60], side-channel analysis using algebraic tools [6], and attacks on reduced versions of KATAN [47,48].

KATAN's use of two NFSRs with short feedback delay (unlike Grain's NFSR and LFSR, where feedback delay is at least eight clockings) contributes to a rapid growth of the density and degree of implicit algebraic equations, which complicates differential and algebraic attacks. Another interesting design idea is its use of a LFSR acting both as a counter of the number of rounds, and as an auxiliary input to the inner logic (to simulate two distinct types of rounds). Like Grain, however, KATAN is inappropriate for a direct reuse in a hash function because of its small block size.

4.3.3. *Taking the Best of Both*

Based on the above observations, we decided to borrow the following features from Grain (more precisely, from Grain-v1):

- A mechanism in which each register's update depends on both registers.
- Boolean functions of high degree (up to six, rather than two in KATAN) and high density.

From KATAN, we chose to reuse:

- Two NFSRs instead of an NFSR and an LFSR; Grain's use of a LFSR was motivated by the need to ensure a long period during the keystream generation (where the LFSR is autonomous), but this seems unnecessary for hashing. Moreover, the dissymmetry in such a design is a potential threat for a secure permutation.
- An auxiliary LFSR to act as a counter and to avoid self-similarity of the round function.

Furthermore, we aimed to choose the parallelization degree as a reasonable trade-off between performance flexibility and security. The number of rounds, equal to four times the size of the internal state, was chosen high enough to provide a comfortable security margin against future attacks.

4.3.4. *Choice of the Boolean Functions*

The quality of the Boolean functions in P strongly affects its security. We thus first chose the functions in QUARK according to their individual properties, according to known metrics (see, e.g., [56]). The final choice was made by observing the empirical

Table 2. Properties of the Boolean functions of each QUARK instance (for h, we consider that the parameter L_0 is zero).

Instance	Boolean function	Var.	Deg.	Nonlin. (max)	Resil.
QUARK (all)	f	13	6	3440 (4056)	3
QUARK (all)	g	13	6	3440 (4056)	3
U-QUARK	h	12	3	1280 (2016)	6
D-QUARK	h	15	3	10240 (16320)	9
S-QUARK	h	16	3	20480 (32640)	10

resistance of the *combination* of the three functions to known attacks (see Sects. 5.2–5.3).

The most important properties to consider in the design of Boolean functions for cryptographic applications are

- *Nonlinearity*: the distance to the set of affine functions.
- *Resilience*: the maximum level of correlation immunity, i.e., the maximum number of variables that one can fix and still obtain a balanced function.
- *Algebraic degree*: the maximum degree of a monomial in the algebraic normal form (ANF) of the function.
- *Density*: the proportion of monomials appearing in the ANF.

Nonlinearity and resilience are closely related to the feasibility of attacks based on linear approximations. The degree and density affect the possibility of (higher-order) differential attacks, as they respectively relate to the notions of confusion and diffusion. For efficiency purposes, however, one seldom uses functions of optimal degree and density.

In QUARK, we chose f and g functions similar to the non-linear function of Grain-v1. These functions achieve good, though suboptimal, nonlinearity and resilience (see Table 2). They have degree six and include monomials of each degree below six. An increase of the degree (from two to six) induces only marginal extra cost in terms of hardware gates, since AND logic needs fewer gates than XOR logic (respectively, approximately one and 2.5). The distinct taps for each register break the symmetry of the design. Note that KATAN also employs similar functions for each register's feedback.

As h function, distinct for each flavor of QUARK, we use a function of lower degree than f and g, but with more linear terms to increase the cross-diffusion between the two registers.

4.3.5. *Choice of the Taps*

The taps of f and g, which correspond respectively to indices within the X and Y registers, were chosen with respect to criteria both analytical (invertibility, irregularity of intervals between two consecutive taps) and empirical (measured diffusion and resistance to cube testers and differential attacks). For h, and contrary to Grain, taps are distributed uniformly in X and Y.

For both f, g, and h, no tap is chosen in the last N bits of the register, where N equals eight for U-QUARK and D-QUARK, and 16 for S-QUARK. This allows one to parallelize an implementation of QUARK in N branches by implementing up to N instances of each

Table 3. Security of the proposed instances of QUARK against the standard security notions, in terms of approximate expected number of computations of P.

Instance	Collision resistance	2nd preimage resistance	Preimage resistance
QUARK	$2^{c/2}$	$2^{c/2}$	2^{c}
U-QUARK	2^{64}	2^{64}	2^{128}
D-QUARK	2^{80}	2^{80}	2^{160}
S-QUARK	2^{112}	2^{112}	2^{224}

function in parallel, and thus to compute N updates of the mechanism within a single clock cycle. We chose a lower (maximum) parallelization degree than Grain because the shorter feedback delay contributes to a more rapid growth of the degree and density of the implicit algebraic equations.

5. Preliminary Security Analysis

This section summarizes the known formal security arguments applying to QUARK, as well as our preliminary cryptanalysis results. We applied state-of-the-art cryptanalysis techniques to all flavors of QUARK, including cube attacks and conditional differential attacks, and could obtain results on at most 25 % of P's rounds.

5.1. *The Hermetic Sponge Strategy*

Like the SHA-3 finalist KECCAK [14], QUARK follows the *hermetic sponge strategy*, which consists in adopting the sponge construction with a permutation that should not have exploitable properties. The indifferentiability proof of the sponge construction [13] implies that any non-generic attack on a QUARK hash function leads to a distinguisher for its permutation P (but a distinguisher for P does not necessarily lead to an attack on QUARK). This reduces the security of P to that of the hash function that uses it.

Since QUARK follows the hermetic sponge strategy, the indifferentiability proof in [13] is directly applicable. The proof ensures an expected complexity at least $\sqrt{\pi}2^{c/2}$ against any *differentiating attack*, regardless of the digest length. This covers for example multicollision attacks or herding attacks [46]. Below we give the known refined bounds for the sponge construction regarding the three standard security notions—as described in [42]—and apply them to the parameters of QUARK. Table 3 summarizes the latter results.

5.1.1. *Collision Resistance*

Collisions for the sponge construction can be found by searching collisions on either the n-bit output, or c bits of the internal state (thanks to the possibility of choosing two appropriate r-bit blocks to complete the collision). The collision resistance of a sponge is thus $\min(2^{n/2}, 2^{c/2})$. The proposed instances of QUARK have $c < n$, thus have a collision resistance $2^{c/2}$.

5.1.2. *Second Preimage Resistance*

The generic second preimage attack against QUARK is similar to the generic preimage attack against the hash function CubeHash [11], which was described in [9] and discussed in [2]. It is a meet-in-the-middle attack that searches for a collision on the internal state, starting from the initial state (forwards) and from a subsequent state that leads to the target digest (backwards). For a success chance $1 - 1/e \approx 0.63$, one requires approximately $2^{c/2}$ trials in each direction, since r bits of the state can be controlled by choosing an adequate message block. This is equivalent to more than $2^{c/2+1}$ evaluations of P and thus to more than $b2^{c/2+3}$ clocks of P's mechanism, that is, 2^{74}, 2^{90}, and 2^{123} clocks for U-, D-, and S-QUARK, respectively.

Note that, contrary to CubeHash, the above attack cannot be used to search for preimages of QUARK. This is because one cannot easily determine two distinct final states that yield the same digest, due to the sponge construction—CubeHash does not follow the sponge construction, but a variant of it that allows such an attack.

If n is smaller than $c/2$, however, a second preimage attack has complexity below $2^{c/2}$. We thus have the general formula $\min(2^n, 2^{c/2})$. Our QUARK instances have $n > c/2$, thus offer $2^{c/2}$ security against second preimage attacks.

5.1.3. *Preimage Resistance*

The original proof of security of the sponge construction gave the bound $\min(2^n, 2^{c/2})$ on the (second) preimage resistance of a sponge function. However, no preimage attack proper to sponge functions with complexity $2^{c/2}$ was known. Instead, the expected workload to find a preimage was previously estimated to $2^{n-r} + 2^{c/2}$ in [17, §5.3], although that was not proven optimal. It was later proven [15] that the preimage resistance of a sponge function is essentially $\min(2^{n-r}, 2^c)$: Theorem 2 in [15, §4.2] implies that if the permutation P has no structural flaw, then finding the internal state leading to a given sequence of output blocks has complexity approximately 2^c. The bound on preimage resistance follows from the fact that finding a preimage implies that the state can be recovered. Note that in QUARK, we have $n - r = c$.

The bound above was further refined in [42], which established the bound $\min(2^{\min(b,n)}, \max(2^{\min(b,n)-r}, 2^{c/2}))$. Our QUARK instances have $b = n$, and thus have preimage resistance $\min(2^b, 2^c) = 2^c$. The generic attack consists in searching for the c bits of internal state that squeeze to the $n - r$ target digest, and then performing a meet-in-the-middle to connect the final state to the initial state, as for a second preimage attack.

5.2. *Resistance to Cube Attacks and Cube Testers*

The recently proposed cube attacks [31] and cube testers [4] are higher-order differential cryptanalysis techniques that exploit weaknesses in the algebraic structure of a cryptographic algorithm. Cube testers can be seen as generalized versions of previous monomials tests [34,55]. These techniques are mostly relevant for algorithms based on non-linear components whose ANF has low degree and low density (e.g., the feedback function of an NFSR). Cube testers were for example applied [3] to the stream cipher Grain-128 [43]. Cube attacks/testers are thus tools of choice to attack (reduced versions of) QUARK's permutation, since it resembles to the Grain ciphers, though with an enhanced security.

Table 4. Highest number of rounds t such that the state (X^t, Y^t) could be distinguished from random using a cube tester with the given complexity. Percentage of the total number of rounds is given in parentheses.

Instance	Total rounds	Rounds attacked		
		in 2^8	in 2^{16}	in 2^{24}
U-QUARK	544	109 (20.0 %)	111 (20.4 %)	114 (21.0 %)
D-QUARK	704	144 (20.5 %)	144 (20.5 %)	148 (21.0 %)
S-QUARK	1024	213 (20.8 %)	220 (21.5 %)	222 (21.7 %)

Recall that QUARK targets security against any nontrivial structural distinguisher for its permutation P. We thus applied cube testers rather than cube attacks, for the former are distinguishers rather than key-recovery attacks. We followed a methodology inspired by [3], using bitsliced C implementations of P and an evolutionary algorithm to optimize the parameters of the attack.

In our simplified attack model, the initial state is chosen uniformly at random to apply our distinguishers. Table 4 reports our results, which can be verified using the parameters given in Appendix C.

One observes in Table 4 that all QUARK flavors showed a similar resistance to our cube testers, with a fraction of approximately 21.0 % of the total number of rounds attacked with complexity 2^{24}. It is difficult to extrapolate to higher complexities; the number of rounds attacked cannot be determined analytically to our present knowledge, though heuristical arguments can be given based on previous results [3,4,31]. The number of rounds attackable seems indeed to evolve logarithmically rather than linearly, as a function of the number of variables used. A worst-case assumption (for the designers) is thus that of a linear evolution. Under this assumption, one could attack 126 rounds of U-QUARK in 2^{64} (23.2 % of the total), 162 rounds of D-QUARK in 2^{80} (23.0 %), and 271 rounds of S-QUARK in 2^{112} (26.5 %).

Using an efficient greedy search rather than evolutionary search seems likely to find better cubes, as suggested by a 2010 work by Stankovski [58]: he found a 40-bit cube leading to a distinguisher on 246 rounds, whereas [3] only reached 237 rounds with a cube of same size (an improvement of almost 5 %).

Note that all of Grain-128's 256 rounds could be attacked in [3] in 2^{24}; this result, however, should not be compared to the value 222 reported in Table 4, since the latter attack concerns *any bit of the internal state*, while the former concerns *the first keystream bit* extracted from the internal state after 220 rounds. Our observation of a bias in P after 222 rounds would thus translate into a distinguisher on $222 - 64 = 158$ of the rounds of a stream cipher derived from P. Conversely, one could thus attack $220 + 64 = 284$ rounds of a version of QUARK using Grain-128 in P, since a bias in the output comes from biases in bits of the internal state. The improved results by Stankovski [58] would lead to a distinguisher on $256 + 64 = 320$ rounds of a version of P directly built from Grain-128. Therefore, although S-QUARK uses registers of same length as Grain-128, it is significantly more resistant to cube testers, and shows a comfortable security margin.

5.3. *Resistance to Differential Attacks*

Differential attacks cover all attacks that exploit non-ideal propagation of differences in a cryptographic algorithm (or components thereof). A large majority of attacks on

Table 5. Highest number of rounds t such that the state (X^t, Y^t) could be distinguished from random using a simple differential distinguisher with the given complexity. Percentage of the total number of rounds is given in parentheses.

Instance	Total rounds	Rounds attacked		
		in 2^8	in 2^{16}	in 2^{24}
U-QUARK	544	109 (20.0 %)	116 (21.3 %)	119 (21.9 %)
D-QUARK	704	135 (19.2 %)	145 (20.6 %)	148 (21.0 %)
S-QUARK	1024	206 (20.1 %)	211 (20.6 %)	216 (21.1 %)

hash functions are at least partially differential, starting with the breakthrough results on MD5 and SHA-1. It is thus crucial to analyze the resistance of new designs to differential attacks. We applied a simple search for truncated differential, as well as state-of-the-art conditional differential attacks, as reported below.

5.3.1. *Simple Truncated Differential Attacks*

We first consider a simple attack model where the initial state is assumed chosen uniformly at random and where one seeks differences in the initial state that give biased differences in the state obtained after the (reduced-round) permutation. We focus on high-probability truncated differentials wherein the output difference concerns a small subset of bits (e.g., a single bit). These are sufficient to distinguish the (reduced-round) permutation from a random one, and are easier to find for an adversary than differentials on all the b bits of the state.

First, we observe that it is easy to track differences during the first few rounds, and in particular to find probability-1 (truncated) differential characteristics for reduced-round versions. For example, in U-QUARK, a difference in the bit Y_{29}^0 in the initial state never leads to a difference in the output of f or of h at the 30th round; hence after $(67 + 30) = 97$ rounds, X_0^{97} will be unchanged. Similar examples can be given for 117 rounds of D-QUARK and 188 rounds of S-QUARK. For higher number of rounds, however, it becomes difficult to manually track differences, and so an automated search becomes necessary. As a heuristical indicator of the resistance to differential attacks, we programmed an automated search for high-probability truncated differentials, given an input difference in a single bit. Table 5 presents our results, showing that we could attack approximately as many rounds with truncated differentials as with cube testers (see Table 4).

We expect advanced search techniques to give differential distinguishers for more rounds (e.g., where the sparse difference occurs slightly later in the internal state, as in [29]). However, such methods seem unlikely to apply to the $4b$-round permutation of QUARK. For example, observe that [29] presented a characteristic of probability 2^{-96} for the full 256-round Grain-128; for comparison, S-QUARK makes 1024 rounds, uses more complex feedback functions, and targets a security level of 112 bits; characteristics of probability greater than 2^{-112} are thus highly improbable, even assuming that the adversary can control differences during (say) the first 256 rounds.

5.3.2. *Resistance to Conditional Differential Attacks*

Conditional differential cryptanalysis [47,48] is a technique introduced to analyze NFSR-based algorithms. This analysis is based on a truncated differential or higher-

Table 6. Highest number of rounds t such that the state (X^t, Y^t) could be distinguished from random using a conditional differential distinguisher with the given complexity. Percentage of the total number of rounds is given in parentheses.

Instance	Total rounds	Rounds attacked		
		in 2^2	in 2^{21}	in 2^{27}
U-QUARK	544	111 (20.4 %)	123 (22.6 %)	136 (25.0 %)
D-QUARK	704	117 (16.6 %)	151 (21.4 %)	159 (22.6 %)
S-QUARK	1024	206 (20.1 %)	233 (22.8 %)	237 (23.1 %)

order differential where the attacker controls the first rounds with some conditions of different types. It was applied to (reduced versions of) KATAN, KTANTAN, Grain-v1, and Grain-128 to mount key-recovery attacks and distinguishing attacks. Unsurprisingly, conditional differential cryptanalysis applies to reduced versions of QUARK's permutation P. Roughly speaking, these attacks start with a random initial state, then impose some conditions on the internal variables, and the samples are generated from some bits that will not modify the said conditions. We obtained the best results by using first order differentials on a single output bit of P.

Table 6 shows improved results compared to cube attacks and simple differential attacks (cf. Tables 4 and 5), but are still very far from $4b$ rounds (the slightly lower percentage of rounds attacked of D-QUARK can be explained by its low ratio parallelization degree over state size, although this point would require further investigation). We can expect that some conditions and differences would improve our results, but it seems unlikely that they will reach even $2b$ rounds of QUARK's permutation.

5.4. *Resistance to Slide Resynchronization Attacks*

Suppose that the initial state of the LFSR of QUARK is not the all-one string, but instead is determined by the input of P—that is, P is redefined to accept $(b + 10)$ rather than b input bits. It is then straightforward to distinguish the modified P from a random transform: pick a first initial state (X^0, Y^0, L^0), and consider the second initial state $(X'^0, Y'^0, L'^0) = (X^1, Y^1, L^1)$, i.e., the state obtained after clocking the first state once. Since all rounds are identical, the shift will be preserved between the two states, leading to final states (X^{4b}, Y^{4b}, L^{4b}) and $(X'^{4b}, Y'^{4b}, L'^{4b}) = (X^{4b+1}, Y^{4b+1}, L^{4b+1})$. One thus obtains two input/output pairs satisfying a nontrivial relation, which is a distinguisher for the modified P considered. The principle of the attack is that of slide attacks on block ciphers [20]; we thus call the above a *slide distinguisher*.

The above idea is at the basis of "slide resynchronization" attacks on Grain-v1 and Grain-128 [29,49], which are related-key attacks using as relation a rotation of the key, to simulate a persistent shift between two internal states.

To avoid the slide distinguisher, we use a trick previously used in KATAN: making each round dependent on a bit coming from a LFSR initialized to a fixed value, in order to simulate two distinct types of rounds. It is thus impossible to have two valid initial states shifted by one or more clocks, and such that the shift persists through the $4b$ rounds.

5.5. *Resistance to Side-Channel Attacks*

Implementations of hash functions are potential targets of side-channel attacks in keyed settings when the adversary's goal is to obtain information on the key (previous works include DPA on HMAC-SHA-2 [51], and template attacks on HMAC-SHA-1 [38]). Without keys, side-channel attacks can also be a threat; for example, fault injection can force a message to successfully pass the integrity check, DPA can be used to obtain information on an unknown message (e.g., a password) hashed multiple times with distinct salts, etc.

Like most cryptographic algorithms (including PRESENT [54]), an unprotected implementation of QUARK is likely to be vulnerable to side-channel attacks, in particular to DPA (see [37] for a DPA of Grain). Protected implementations are expected to need at least thrice more gates than the implementations reported below, due to the overhead imposed by countermeasures such as hiding and masking.

6. Hardware Implementation

This section reports our hardware implementation of the QUARK instances. Note that QUARK is not optimized for software (be it 64- or 8-bit processors), and other designs are preferable for such platforms (such as PHOTON [41]). We thus focus our evaluation on hardware efficiency.

Our results arise from pure simulations, and are thus not supported by real measurements on a fabricated chip. However, we believe that this evaluation gives a fair and reliable overview of the overall VLSI performance of QUARK.

6.1. *Architectures*

Three characteristics make QUARK particularly attractive for lightweight hashing: first, the absence in its sponge construction of "feedforward" values, which normally would require additional dedicated memory components; second, its use of shift registers, which are straightforward to implement in hardware; and third, the possibility of several space/time implementation trade-offs. Based on the two extremal trade-off choices, we designed two architecture variants of U-QUARK, D-QUARK, and S-QUARK:

- **Serial architecture**: Only one permutation module, hosting the circuit for the functions f, g, and h, is implemented. Each clock cycle, the bits of the registers X, Y, and L are shifted by one. These architectures correspond to the most compact designs. They contain the minimal circuitry needed to handle incoming messages and to generate the correct output digests.
- **Parallel architecture**: The number of the implemented permutation modules corresponds to the parallelization degree given in Sect. 3.3. The bits in the registers are accordingly shifted. These architectures increase the number of rounds computed per cycle—and therefore the throughput—at extra area costs.

In addition to the three feedback shift registers, each design has a dedicated controller module that handles the sponge process. This module is made up of a finite-state machine and of two counters for the round and the output digest computation. After processing all message blocks during the absorbing phase, the controller switches automat-

ically to the squeezing phase (computation of the digest), if no further r-bit message blocks are given. This implies that the message has been externally padded.

6.2. *Methodology*

We described the serial and parallel architectures of each QUARK instance in functional VHDL, and synthesized the code with Synopsys Design Vision-2009.06 targeting the UMC 0.18 µm 1P6M CMOS technology with the FSA0A_C cell library from Faraday Technology Corporation. We used the generic process (at typical conditions), instead of the low-leakage for two reasons: first, the leakage dissipation is not a big issue in 0.18 µm CMOS, and second, for such small circuits the leakage power is about two orders of magnitude smaller than the total power. To provide a thorough and more reliable analysis, we extended the implementation up to the back-end design. Place and route have been carried out with the help of Cadance Design Systems Velocity-9.1. In a square floorplan, we set a 98 % row density, i.e., the utilization of the core area. Two external power rings of 1.2 µm were sufficient for power and ground distribution. In this technology, six metal layers are available for routing. However, during the routing phase, the fifth and the sixth layers were barely used. The design flow has been placement, clock tree synthesis, and routing with intermediate timing optimizations.

Each architecture was implemented at the target frequency of 100 kHz. As noted in [23,30], this is a typical operating frequency of cryptographic modules in RFID systems. Power simulation was measured for the complete design under real stimuli simulations (two consecutive 512-bit messages) at 100 kHz. The switching activity of the circuit's internal nodes was computed generating Value Change Dump (VCD) files. These were then used to perform statistical power analysis in the velocity tool. Besides the mean value, we also report the peak power consumption, which is a limiting parameter in RFID systems (a maximum of 27 µW is suggested in [36]). Table 7 reports the performance metrics obtained from our simulations at 100 kHz. To give an overview of the best speed achievable, we also implemented the parallel architectures increasing the timing constraints (see Table 8).

6.3. *Results and Discussion*

As reported in Table 7, each of the three serial designs needs fewer than 2300 GE, thus making 112-bit security affordable for restricted-area environments. Particularly appealing for ultra-compact applications is the U-QUARK function, which offers 64-bit security but requires only 1379 GE and dissipates less than 2.5 µW. To the best of our knowledge, U-QUARK is lighter than all previous designs with comparable security claims. We expect an instance of QUARK with 256-bit security (e.g., with $r = 64, c = 512$) to fit in 4500 GE.

Note that in the power results of the QUARK circuits, the single contributions of the mean power consumption are 68 % of internal, 30 % of switching, and 2 % of leakage power. Also important is that the peak value exceeds maximally 27 % of the mean value.

The maximum speed achieved by the parallel cores is 357 Mbps with S-QUARK×16 clocked with a period of 1.4 ns (see Table 8). At this frequency, the values of the power dissipation increase up to 30–60 mW. The leakage component does not contribute significantly. Indeed, 38 % of the total power is devoted to switching, with the rest for

Table 7. Compared hardware performance of PRESENT-based (post-synthesis), KECCAK, and QUARK (post-layout) lightweight hash functions. Note that the QUARK results are post-layout figures. Security is expressed in bits (e.g., "64" in the "Pre." column means that preimages can be found within approximately 2^{64} calls to the function). Throughput and power consumption are given for a frequency of 100 kHz, assuming a long message.

Hash function	Parameters[a]			Security		Area[b]	Lat.	Thr.	Power [μW]	
	n	c	r	Pre	Col	[GE]	[cycles]	[kbps]	Mean	Peak
u-QUARK	136	128	8	128	64	1379	544	1.47	2.44	2.96
u-QUARK×8	136	128	8	128	64	2392	68	11.76	4.07	4.84
d-QUARK	176	160	16	160	80	1702	704	2.27	3.10	3.95
d-QUARK×8	176	160	16	160	80	2819	88	18.18	4.76	5.80
s-QUARK	256	224	32	224	112	2296	1024	3.13	4.35	5.53
s-QUARK×16	256	224	32	224	112	4640	64	50.00	8.39	9.79
Implementations of PRESENT-based hashes from [23] (0.18 μm)										
DM-PRESENT-80	64	64	80	64	32	1600	547	14.63	1.83	–
DM-PRESENT-80	64	64	80	64	32	2213	33	242.42	6.28	–
DM-PRESENT-128	64	64	128	64	32	1886	559	22.90	2.94	–
DM-PRESENT-128	64	64	128	64	32	2530	33	387.88	7.49	–
H-PRESENT-128	128	128	64	128	64	2330	559	11.45	6.44	–
H-PRESENT-128	128	128	64	128	64	4256	32	200.00	8.09	–
Implementations[c] of KECCAK[200] from [14, §9.4] (0.13 μm)										
KECCAK[72,128]	200	128	72	128	64	1300	3870	1.86	–	–
KECCAK[40,160]	200	160	40	160	80	1300	3870	1.03	–	–
Implementations of KECCAK[200] from [45] (0.13 μm)										
KECCAK[72,128]	200	128	72	128	64	2520	900	8.00	5.60	–
KECCAK[72,128]	200	128	72	128	64	4900	900	400.00	27.60	–
KECCAK[40,160]	200	160	40	160	80	2520	900	4.44	5.60	–
KECCAK[40,160]	200	160	40	160	80	4900	900	222.22	27.60	–
Implementations of PHOTON from [42] (0.18 μm)										
PHOTON-128/16/16	128	128	16	112	64	1122	996	1.61	2.29	–
PHOTON-128/16/16	128	128	16	112	64	1708	156	10.26	3.45	–
PHOTON-160/36/36	160	160	36	124	80	1396	1332	2.70	2.74	–
PHOTON-160/36/36	160	160	36	124	80	2117	180	20.00	4.35	–
PHOTON-224/32/32	224	224	32	112	64	1736	1716	1.86	4.01	–
PHOTON-224/32/32	224	224	32	112	64	2786	204	15.69	6.50	–
Implementations of SPONGENT from [24] (0.13 μm)										
SPONGENT-128	128	128	8	120	64	1060	2380	0.34	2.20	–
SPONGENT-128	128	128	8	120	64	1687	70	11.43	3.58	–
SPONGENT-160	176	160	16	144	80	1329	3960	0.40	2.85	–
SPONGENT-160	176	160	16	144	80	2190	90	17.78	4.47	–
SPONGENT-224	240	224	16	208	112	1728	7200	0.22	3.73	–
SPONGENT-224	240	224	16	208	112	2903	120	13.33	5.97	–

[a] For the non-sponge PRESENT-based functions, n, c, r are respectively the lengths of a digest, internal state, and message block.

[b] For QUARK implementations, one GE is the area of a 2-input drive-one NAND gate, i.e., in the target 0.18 μm technology, 9.3744 μm^2.

[c] These implementations use external memory to store the internal state.

Table 8. Maximum-speed performances of our parallel implementations of QUARK, compared with the implementation of KECCAK[200] in [14].

Hash function	Block [bits]	Area [GE]	Lat. [cycles]	Freq. [MHz]	Thr. [Mbps]	Power [μW]	
						Mean	Peak
U-QUARK×8	8	3032	68	714	84.0	30.46	37.01
D-QUARK×8	16	3561	88	714	129.8	37.14	43.35
S-QUARK×16	32	6220	64	714	357.0	65.34	75.27
KECCAK[$r = 40, c = 160$][a]	40	1600	3870	714	7.4	–	–

[a]This implementation uses external memory to store the internal state, and is on 0.13 μm technology.

internal power. We do not exclude the possibility to reach higher speed ratios with different architectures. In practice, the latency could be further reduced by implementing more permutation modules. Due to the tap configuration in the function f, g, and h, this would also increase the circuit complexity (i.e., more area and lower frequency), which was outside the scope of this analysis.

6.3.1. *Comparison to Previous Designs*

As reported in Table 7, the functions DM-PRESENT-80/128 and H-PRESENT-128 also offer time/space implementation trade-offs. For a same second preimage resistance of at least 64 bits, U-QUARK fits in a smaller area (1379 vs. 1600 GE), and even the 80-bit-secure D-QUARK does not need more GE than DM-PRESENT-128. In terms of throughput, however, QUARK underperforms PRESENT-based designs. Not only, the figures provided in Tables 7 and 8 are computed for a generic long-size message, omitting the latency of the squeezing phase. Indeed, since QUARK has a smaller rate than the digest size, the complete hash value is only generated after additional executions of the permutation P. In the case of small-size messages, this behavior penalizes QUARK, and more generally the sponge construction, with respect to the PRESENT-based hash functions. The significantly smaller speed values may be due to QUARK's higher security margin (note that 26 of the 31 rounds of PRESENT, as a block cipher, were attacked [25], suggesting a thin security margin against distinguishers in the "open key" model of hash functions). Moreover, QUARK provides at least 64-bit preimage resistance, while both DM-PRESENT versions are limited to a 32-bit collision resistance, making collision search practical.

Compared to the small versions of KECCAK, which follow the same design philosophy as QUARK, the latter have lower area requirements for a given security level (even when comparing 0.18 μm with 0.13 μm technologies). Implementations in [45] also suggest a higher power consumption than QUARK.

6.3.2. *Comparison to Subsequent Designs*

The lightweight hash functions PHOTON and SPONGENT, which appeared after the publication of QUARK, also use a sponge construction (or a slightly modified version thereof). However, they build P on highly optimized block cipher-like constructions,

and seem to allow slightly more compact implementations than QUARK (note, however, that SPONGENT implementations were realized on 0.13 µm technology, against 0.18 µm for QUARK and PHOTON). Interestingly, each of the three functions has unique characteristics, and none seems to dominate on all aspects; for example, SPONGENT and PHOTON have slightly lower footprints; however, the former has a significantly lower throughput than QUARK and PHOTON, while the latter appears to have a lower security margin.

Acknowledgements

We would like to thank the KECCAK team for many helpful comments on a preliminary version of this article, Hidenori Kuwakado for noticing an error in the first C implementation of QUARK, and the reviewers of the Journal of Cryptology for their insightful feedback.

Willi Meier is supported by the Hasler foundation www.haslerfoundation.ch under project number 08065. María Naya-Plasencia is supported by the National Competence Center in Research on Mobile Information and Communication Systems (NCCR-MICS), a center of the Swiss National Science Foundation under grant number 5005-67322, and partially by the French Agence Nationale de la Recherche through the SAPHIR2 project under Contract ANR-08-VERS-014.

Appendix A. Initial States

The initial states of each instance of QUARK are chosen as the first bits of their SHA-256 digest (e.g., U-QUARK's initial state corresponds to the first 136 bits of SHA-256("u-quark")). The hexadecimal values below present the IV from s_0 to s_{b-1} (cf. Sect. 3.2). That is, the initial state of U-QUARK has $X_0 = 1, X_1 = 1, X_2 = 0, X_3 = 1$, which corresponds to the first hexadecimal digit D (1101 in binary).

U-QUARK: `D8DACA44414A099719C80AA3AF065644DB`
D-QUARK: `CC6C4AB7D11FA9BDF6EEDE03D87B68F91BAA706C20E9`
S-QUARK: `397251CEE1DE8AA73EA26250C6D7BE128CD3E79DD718C24B8A19D09C2492DA5D`

Appendix B. Test Values

Using the same endianness convention as in Appendix A, we give below the intermediate values of the internal state when hashing the empty message (that is, after padding, the blocks `80`, `8000`, `80000000` for U-QUARK to S-QUARK).

U-QUARK

Initial state after XOR with the message block `80`:
`D8DACA44414A099719C80AA3AF0656445B`

State after applying the only permutation of the absorbing phase:
9A03A9DEFBB9ED3867DAB18EC039276212

States after each permutation of the squeezing phase:
4C983B073679AD44498C7DED5B5A3EC16B
CD18A9431D86D59100F114398B45869375
DE2DA1946E4D047A641F31EF8A884E13BC
61A3BF954EC85422ADAF58349D485D2CAB
A526ABB27ABD03661D3E04876FCB7B6423
C47103489721DEF7E7F67F6952F4180A14
FA5671E806083DB70885867946CE0BC947
25C149CA3418D1F86FDC4A195827174250
47A44A6590C7A05B8A3B641B262ECB2ED0
FE3D800B292D9DC5E766BAFD9F1CD36A8B
DC21EF190455FD30B84F8012ACC03E72A3
865D7978420A74F7F1901C7724F97FE013
50C180B068D3CD04CE25F1DDAB868E9DBB
62347472491643FABB8051344C4CA38CD8
89C3B410F2EBE58E8CCC9AB056A5E50A00

Digest returned:
126B75BCAB23144750D08BA313BBD800A4

D-QUARK

Initial state after XOR with the message block 8000:
CC6C4AB7D11FA9BDF6EEDE03D87B68F91BAA706CA0E9

State after applying the only permutation of the absorbing phase:
E1AFDDED75F72D33AE3F60D3A1A9E9FA759AC6F082C7

States after each permutation of the squeezing phase:
D013143E679FAEC7A2B6EB458498FED5DC498145F380
7D9E93000F8A30236E8FD3E85BE3C096705E2FD6E231
FC595197C3415152DB7FF0E246CD4AB92E98D3C2578E
FA0E4CF5390554A0841F15310C908C4066F8CF162FF4
9498279CAA9AC4C293245DB08CA40BAF61FB32EFC2A4
ADAD6159EAB1656B022A4B06E4454A4B025426B302E1
D30C6F301AD93B8A07E212732B7B6C7B0DA1FDC38BF3
12F7B39AFC823FB430892B89D0F6CBAADF36A46C7AEA
0D6B4D0554F5F343BCB26AAC85CC8019BC486AF88477

Digest returned:
82C7F380E231578E2FF4C2A402E18BF37AEA8477298D

S-QUARK

Initial state after XOR with the message block 80000000:
397251CEE1DE8AA73EA26250C6D7BE128CD3E79DD718C24B8A19D09CA492DA5D

State after applying the only permutation of the absorbing phase:
3D63F54100A7BC5135692F3BDE1563F7998A6965FE6D26AB40262D2003256214

States after each permutation of the squeezing phase:
12603448212FCAF31D611E986F6C9C10C42E1DD79D91B74407ECE15AB92E811C
FFDBEED704CC5D6BE6CCF7E32A9F563278DAA52D38C870588E84DBEA321AE86B
5804FEAD1E4357EC99D9B6D98624F4F649A9FAF384C434D7C79988A0AB4B0E7A
3B2EFFC05882C5BCA5A191FD20945445AC1C1A660B1B8FAD0F746670E9C22C42
7B2184B713EE554B914D66447D76F725340199622EE4F768069F2C07882FCCDE
D69E1FA2067F8A54606D81F9DE212D51C48B3C4C12CFF9EE013740118C22BFF6

Digest returned:
03256214B92E811C321AE86BAB4B0E7AE9C22C42882FCCDE8C22BFF6A0A1D6F1

Appendix C. Index Sets for Cube Testers

Below we give the 24-bit index sets used to obtain the results in Table 4:

U-QUARK:
{0, 5, 6, 8, 15, 17, 18, 23, 29, 33, 34, 42, 44, 45, 57, 58, 72, 78, 99, 101, 114, 118, 120, 126}

D-QUARK:
{0, 22, 24, 25, 31, 32, 34, 39, 43, 48, 55, 58, 62, 69, 82, 90, 94, 101, 128, 133, 140, 141, 143, 159}

S-QUARK:
{12, 19, 23, 30, 36, 37, 38, 43, 46, 49, 51, 56, 57, 59, 66, 91, 95, 97, 162, 170, 182, 185, 219, 249}

References

[1] M. Ågren, M. Hell, T. Johansson, W. Meier, A new version of Grain-128 with authentication, in *ECRYPT Symmetric Key Encryption Workshop 2011* (2011). Available at http://skew2011.mat.dtu.dk/
[2] J.-P. Aumasson, E. Brier, W. Meier, M. Naya-Plasencia, T. Peyrin, Inside the hypercube, in *ACISP*, ed. by C. Boyd, J. Manuel González Nieto. LNCS, vol. 5594 (Springer, Berlin, 2009), pp. 202–213
[3] J.-P. Aumasson, I. Dinur, L. Henzen, W. Meier, A. Shamir, Efficient FPGA implementations of high-dimensional cube testers on the stream cipher Grain-128, in *SHARCS* (2009)
[4] J.-P. Aumasson, I. Dinur, W. Meier, A. Shamir, Cube testers and key recovery attacks on reduced-round MD6 and Trivium, in *FSE*, ed. by O. Dunkelman. LNCS, vol. 5665 (Springer, Berlin, 2009), pp. 1–22
[5] J.-P. Aumasson, L. Henzen, W. Meier, M. Naya-Plasencia, Quark: a lightweight hash, in *Mangard and Standaert [50]* (2010), pp. 1–15
[6] G.V. Bard, N. Courtois, J. Nakahara, P. Sepehrdad, B. Zhang, Algebraic, AIDA/cube and side channel analysis of KATAN family of block ciphers, in *Gong and Gupta [39]* (2010), pp. 176–196
[7] M. Bellare, T. Ristenpart, Multi-property-preserving hash domain extension and the EMD transform, in *ASIACRYPT*, ed. by X. Lai, K. Chen. LNCS, vol. 4284 (Springer, Berlin, 2006), pp. 299–314
[8] M. Bernet, L. Henzen, H. Kaeslin, N. Felber, W. Fichtner, Hardware implementations of the SHA-3 candidates Shabal and CubeHash, in *CT-MWSCAS* (IEEE, New York, 2009)

[9] D.J. Bernstein, CubeHash appendix: complexity of generic attacks. Submission to NIST, 2008. http://cubehash.cr.yp.to/submission/generic.pdf
[10] D.J. Bernstein, CubeHash parameter tweak: 16 times faster, 2009. http://cubehash.cr.yp.to/submission/tweak.pdf
[11] D.J. Bernstein, CubeHash specification (2.B.1). Submission to NIST (Round 2), 2009. http://cubehash.cr.yp.to/submission2/spec.pdf
[12] G. Bertoni, J. Daemen, M. Peeters, G. Van Assche, RADIOGATÚN, a belt-and-mill hash function, in *Second NIST Cryptographic Hash Function Workshop* (2006). http://radiogatun.noekeon.org/
[13] G. Bertoni, J. Daemen, M. Peeters, G. Van Assche, On the indifferentiability of the sponge construction, in *EUROCRYPT*, ed. by N.P. Smart. LNCS, vol. 4965 (Springer, Berlin, 2008), pp. 181–197
[14] G. Bertoni, J. Daemen, M. Peeters, G. Van Assche, Keccak sponge function family main document (version 2.1). Submission to NIST (Round 2), 2010. http://keccak.noekeon.org/Keccak-main-2.1.pdf
[15] G. Bertoni, J. Daemen, M. Peeters, G. Van Assche, Sponge-based pseudo-random number generators, in *Mangard and Standaert [50]* (2010), pp. 33–47
[16] G. Bertoni, J. Daemen, M. Peeters, G. Van Assche, On the security of the keyed sponge construction, in *ECRYPT Symmetric Key Encryption Workshop 2011* (2011). Available at http://skew2011.mat.dtu.dk/
[17] G. Bertoni, J. Daemen, M. Peeters, G. Van Assche, Sponge functions. http://sponge.noekeon.org/SpongeFunctions.pdf
[18] G. Bertoni, J. Daemen, M. Peeters, G. Van Assche, Duplexing the sponge: single-pass authenticated encryption and other applications. Cryptology ePrint Archive, Report 2011/499, 2011
[19] E. Biham, O. Dunkelman, A framework for iterative hash functions—HAIFA. Cryptology ePrint Archive, Report 2007/278, 2007
[20] A. Biryukov, D. Wagner, Slide attacks, in *FSE*, ed. by L. Knudsen. LNCS, vol. 1636 (Springer, Berlin, 1999), pp. 245–259
[21] A. Bogdanov, C. Rechberger, A 3-subset meet-in-the-middle attack: cryptanalysis of the lightweight block cipher KTANTAN. Cryptology ePrint Archive, Report 2010/532, 2010
[22] A. Bogdanov, L.R. Knudsen, G. Leander, C. Paar, A. Poschmann, M.J.B. Robshaw, Y. Seurin, C. Vikkelsoe, PRESENT: an ultra-lightweight block cipher, in *CHES*, ed. by P. Paillier, I. Verbauwhede. LNCS, vol. 4727 (Springer, Berlin, 2007), pp. 450–466
[23] A. Bogdanov, G. Leander, C. Paar, A. Poschmann, M.J.B. Robshaw, Y. Seurin, Hash functions and RFID tags: mind the gap, in *CHES*, ed. by E. Oswald, P. Rohatgi. LNCS, vol. 5154 (Springer, Berlin, 2008), pp. 283–299
[24] A. Bogdanov, M. Knezevic, G. Leander, D. Toz, K. Varici, I. Verbauwhede, SPONGENT: a lightweight hash function, in *CHES*, ed. by B. Preneel, T. Takagi. LNCS, vol. 6917 (Springer, Berlin, 2011), pp. 312–325
[25] J.Y. Cho, Linear cryptanalysis of reduced-round PRESENT, in *CT-RSA*, ed. by J. Pieprzyk. LNCS, vol. 5985 (Springer, Berlin, 2010), pp. 302–317
[26] C. Clavier, K. Gaj (eds.), *Cryptographic Hardware and Embedded Systems—CHES 2009, 11th International Workshop, Lausanne, Switzerland, September 6–9, 2009, Proceedings*. LNCS, vol. 5747 (Springer, Berlin, 2009)
[27] J.-S. Coron, Y. Dodis, C. Malinaud, P. Puniya, Merkle–Damgård revisited: how to construct a hash function, in *CRYPTO*, ed. by V. Shoup. LNCS, vol. 3621 (Springer, Berlin, 2005), pp. 430–448
[28] C. De Cannière, B. Preneel, Trivium, in *New Stream Cipher Designs*. LNCS, vol. 4986 (Springer, Berlin, 2008), pp. 84–97
[29] C. De Cannière, Ö. Kücük, B. Preneel, Analysis of Grain's initialization algorithm, in *SASC 2008* (2008)
[30] C. De Cannière, O. Dunkelman, M. Knezevic, KATAN and KTANTAN—a family of small and efficient hardware-oriented block ciphers, in *Clavier and Gaj [26]* (2009), pp. 272–288
[31] I. Dinur, A. Shamir, Cube attacks on tweakable black box polynomials, in *EUROCRYPT*, ed. by A. Joux. LNCS, vol. 5479 (Springer, Berlin, 2009), pp. 278–299
[32] I. Dinur, A. Shamir, Breaking Grain-128 with dynamic cube attacks. Cryptology ePrint Archive, Report 2010/570, 2010
[33] I. Dinur, T. Güneysu, C. Paar, A. Shamir, R. Zimmermann, An experimentally verified attack on full Grain-128 using dedicated reconfigurable hardware, in *ASIACRYPT*, ed. by D.H. Lee, X. Wang. LNCS, vol. 7073 (Springer, Berlin, 2011), pp. 327–343

[34] H. Englund, T. Johansson, M.S. Turan, A framework for chosen IV statistical analysis of stream ciphers, in *INDOCRYPT*, ed. by K. Srinathan, C. Pandu Rangan, M. Yung. LNCS, vol. 4859 (Springer, Berlin, 2007), pp. 268–281
[35] M. Feldhofer, C. Rechberger, A case against currently used hash functions in RFID protocols, in *OTM Workshops (1)*, ed. by R. Meersman, Z. Tari, P. Herrero. LNCS, vol. 4277 (Springer, Berlin, 2006), pp. 372–381
[36] M. Feldhofer, J. Wolkerstorfer, Strong crypto for RFID tags—a comparison of low-power hardware implementations, in *ISCAS 2007* (IEEE, New York, 2007), pp. 1839–1842
[37] W. Fischer, B.M. Gammel, O. Kniffler, J. Velten, Differential power analysis of stream ciphers, in *SASC 2007* (2007)
[38] P.-A. Fouque, G. Leurent, D. Réal, F. Valette, Practical electromagnetic template attack on HMAC, in *Clavier and Gaj [26]* (2009), pp. 66–80
[39] G. Gong, K.C. Gupta (eds.), *Progress in Cryptology—INDOCRYPT 2010—11th International Conference on Cryptology in India*, Hyderabad, India, December 12–15, 2010. LNCS, vol. 6498 (Springer, Berlin, 2010)
[40] T. Good, M. Benaissa, Hardware performance of eSTREAM phase-III stream cipher candidates, in *SASC* (2008)
[41] J. Guo, T. Peyrin, A. Poschmann, The PHOTON family of lightweight hash functions, in *CRYPTO*, ed. by P. Rogaway. LNCS, vol. 6841 (Springer, Berlin, 2011), pp. 222–239
[42] J. Guo, T. Peyrin, A. Poschmann, The PHOTON family of lightweight hash functions (2011). Available on https://sites.google.com/site/photonhashfunction/. Full version of [41]
[43] M. Hell, T. Johansson, A. Maximov, W. Meier, A stream cipher proposal: Grain-128, in *IEEE International Symposium on Information Theory (ISIT 2006)* (2006)
[44] M. Hell, T. Johansson, W. Meier, Grain: a stream cipher for constrained environments. *Int. J. Wirel. Mob. Comput.* **2**(1), 86–93 (2007)
[45] E.B. Kavun, T. Yalcin, A lightweight implementation of Keccak hash function for radio-frequency identification applications, in *RFIDSec*, ed. by S.B.O. Yalcin. LNCS, vol. 6370 (Springer, Berlin, 2010), pp. 258–269
[46] J. Kelsey, T. Kohno, Herding hash functions and the Nostradamus attack, in *EUROCRYPT*, ed. by S. Vaudenay. LNCS, vol. 4004 (Springer, Berlin, 2006), pp. 183–200
[47] S. Knellwolf, W. Meier, M. Naya-Plasencia, Conditional differential cryptanalysis of NLFSR-based cryptosystems, in *ASIACRYPT*, ed. by M. Abe. LNCS, vol. 6477 (Springer, Berlin, 2010), pp. 130–145
[48] S. Knellwolf, W. Meier, M. Naya-Plasencia, Conditional differential cryptanalysis of Trivium and KATAN, in *Selected Areas in Cryptography*, ed. by A. Miri, S. Vaudenay. LNCS, vol. 7118 (Springer, Berlin, 2012), pp. 200–212
[49] Y. Lee, K. Jeong, J. Sung, S. Hong, Related-key chosen IV attacks on Grain-v1 and Grain-128, in *ACISP*, ed. by Y. Mu, W. Susilo, J. Seberry. LNCS, vol. 5107 (Springer, Berlin, 2008), pp. 321–335
[50] S. Mangard, F.-X. Standaert (eds.), *Cryptographic Hardware and Embedded Systems, CHES 2010, 12th International Workshop*, Santa Barbara, CA, USA, August 17–20, 2010. LNCS, vol. 6225 (Springer, Berlin, 2010)
[51] R.P. McEvoy, M. Tunstall, C.C. Murphy, W.P. Marnane, Differential power analysis of HMAC based on SHA-2, and countermeasures, in *WISA*, ed. by S. Kim, M. Yung, H.-W. Lee. LNCS, vol. 4867 (Springer, Berlin, 2007), pp. 317–332
[52] NIST, Cryptographic hash algorithm competition. http://www.nist.gov/hash-competition
[53] M. O'Neill, Low-cost SHA-1 hash function architecture for RFID tags, in *Workshop on RFID Security RFIDsec* (2008)
[54] M. Renauld, F.-X. Standaert, Combining algebraic and side-channel cryptanalysis against block ciphers, in *30th Symposium on Information Theory in the Benelux* (2009), pp. 97–104. http://www.dice.ucl.ac.be/~fstandae/68.pdf
[55] M.-J.O. Saarinen, Chosen-IV statistical attacks on eStream ciphers, in *SECRYPT*, ed. by M. Malek, E. Fernández-Medina, J. Hernando (INSTICC Press, Setubal, 2006), pp. 260–266
[56] P. Sarkar, S. Maitra, Construction of nonlinear boolean functions with important cryptographic properties, in *EUROCRYPT*, ed. by B. Preneel. LNCS, vol. 1807 (Springer, Berlin, 2000), pp. 485–506
[57] A. Shamir, SQUASH—a new MAC with provable security properties for highly constrained devices such as RFID tags, in *FSE*, ed. by K. Nyberg. LNCS, vol. 5086 (Springer, Berlin, 2008), pp. 144–157

[58] P. Stankovski, Greedy distinguishers and nonrandomness detectors, in *Gong and Gupta [39]* (2010), pp. 210–226
[59] G. Van Assche, Errata for Keccak presentation. Email sent to the NIST SHA-3 mailing list on Feb. 7, 2011, on behalf of the Keccak team
[60] L. Wei, C. Rechberger, J. Guo, H. Wu, H. Wang, S. Ling, Improved meet-in-the-middle cryptanalysis of KTANTAN (poster), in *ACISP*, ed. by U. Parampalli, P. Hawkes. LNCS, vol. 6812 (Springer, Berlin, 2011), pp. 433–438
[61] H. Yoshida, D. Watanabe, K. Okeya, J. Kitahara, H. Wu, O. Kucuk, B. Preneel, MAME: a compression function with reduced hardware requirements, in *ECRYPT Hash Workshop 2007* (2007)