QUARK: A Lightweight Hash*

Jean-Philippe Aumasson[1], Luca Henzen[2],
Willi Meier[3,**], and María Naya-Plasencia[3,***]

[1] Nagravision SA, Cheseaux, Switzerland
[2] ETH Zurich, Switzerland
[3] FHNW, Windisch, Switzerland

Abstract. The need for lightweight cryptographic hash functions has been repeatedly expressed by application designers, notably for implementing RFID protocols. However not many designs are available, and the ongoing SHA-3 Competition probably won't help, as it concerns general-purpose designs and focuses on software performance. In this paper, we thus propose a novel design philosophy for lightweight hash functions, based on a single security level and on the sponge construction, to minimize memory requirements. Inspired by the lightweight ciphers Grain and KATAN, we present the hash function family QUARK, composed of the three instances U-QUARK, D-QUARK, and T-QUARK. Hardware benchmarks show that QUARK compares well to previous lightweight hashes. 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 in average 2.44 µW at 100 kHz in 0.18 µm ASIC. For 112-bit security, we propose T-QUARK, which we implemented with 2296 gate-equivalents.

1 Introduction

In 2006, Feldhofer and Rechberger [1] pointed out the lack of lightweight hash functions for use in RFID protocols, and gave recommendations to encourage the design of such primitives. But as recently observed in [2] the situation has not much evolved in four years[1], despite a growing demand; besides RFID, lightweight hashes are indeed relevant wherever the cost of hardware matters, be it in embedded systems or in smartcards.

* This work was partially supported by European Commission through the ICT programme under contract ICT-2007-216676 ECRYPT II.
** Supported by the Hasler foundation `www.haslerfoundation.ch` under project number 08065.
*** This work was carried out during the tenure of an ERCIM "Alain Bensoussan" Fellowship Programme.
[1] Note the absence of "Hash functions" category in the ECRYPT Lightweight Cryptography Lounge (`http://www.ecrypt.eu.org/lightweight/`).

S. Mangard and F.-X. Standaert (Eds.): CHES 2010, LNCS 6225, pp. 1–15, 2010.

The ongoing NIST "SHA-3" Hash Competition [3] 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)[2]. An exception is CubeHash [5], which can be implemented with 7630 GE in 0.13 µm ASIC [6] to return digests of up to 512 bits. For comparison, Feldhofer and Wolkerstorfer [7] reported a 8001 GE implementation of MD5 (128-bit digests, 0.35 µm ASIC), O'Neill [8] implemented SHA-1 (160-bit, 0.18 µm) with 6122 GE, and the compression function MAME by Yoshida et al [9] (256-bit, 0.18 µm) fits in 8100 GE. These designs, however, are still too demanding for many low-end environments.

A major step towards lightweight hashing is the work by Bogdanov et al. [10], which presented constructions based on the lightweight block cipher PRESENT [11]; they for example proposed to instantiate the Davies-Meyer ($E_m(h) \oplus h$) construction with PRESENT-80, giving a hash function with 64-bit digests implemented with as few as 1600 GE, in 0.18 µm ASIC.

Another interesting approach was taken with Shamir's SQUASH [12], 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 this paper, we present a novel approach to design lightweight hashes, illustrated with the proposal of a new family of functions, called QUARK.

2 Description of the QUARK Hash Family

2.1 Sponge Construction

QUARK uses the sponge construction, as depicted in Fig. 1. Following the notations introduced in [13], it is parametrized by a *rate* (or block length) r, a *capacity* c, and an *output length* n. The *width* of a sponge construction is the size of its internal state $b = r + c \geq n$.

Given an initial state, the sponge construction processes a message m as follows:

1. **Initialization**: The message is padded by appending a '1' bit and sufficiently many zeroes to reach length a multiple of r.
2. **Absorbing phase**: The r-bit message blocks are xored into the last r bits of the state (i.e., $Y_{b/2-r}, \ldots, Y_{b/2-1}$), interleaved with applications of 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.

[2] For example, the "5000 GE" implementation of Keccak reported in [4, §7.4.3] is only that of the coprocessor, without the memory storing the 1600-bit internal state. Hence the total gate count of the complete design is well above 10 000.

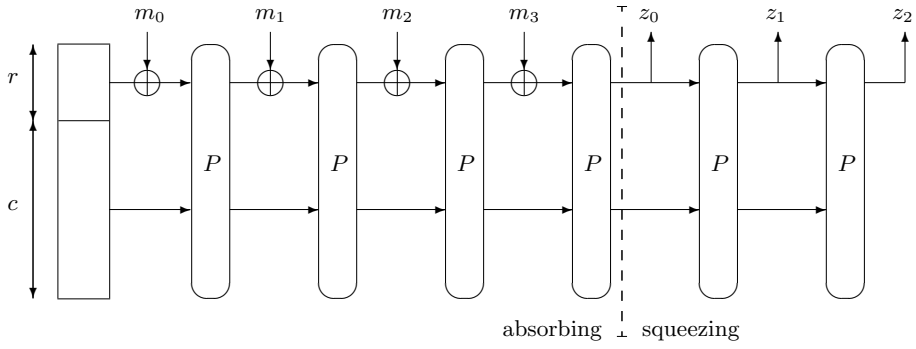

Fig. 1. The sponge construction as used by QUARK, for the example of a 4-block (padded) message

2.2 Permutation

QUARK uses a permutation P inspired by the stream cipher Grain and by the block cipher KATAN (see §3.3 for details), as depicted in Fig. 2.

The permutation P relies on three non-linear Boolean functions f, g, and h, on a linear Boolean function p, and on an internal state composed, at epoch $t \geq 0$, of

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

P processes a b-bit input in three stages, as described below:

Initialization. Upon input $s = (s_0, \ldots, s_{b-1})$, 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: $(L_0^0, \ldots, L_{\lceil \log 4b \rceil - 1}^0) := (1, \ldots, 1)$.

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: $h^t := h(X^t, Y^t, L^t)$;
2. X is clocked using Y_0^t, the function f, and h^t:
$$(X_0^{t+1}, \ldots, X_{b/2-1}^{t+1}) := (X_1^t, \ldots, X_{b/2-1}^t, Y_0^t + f(X^t) + h^t) \; ;$$
3. Y is clocked using the function g and h^t:
$$(Y_0^{t+1}, \ldots, Y_{b/2-1}^{t+1}) := (Y_1^t, \ldots, Y_{b/2-1}^t, g(Y^t) + h^t) \; ;$$
4. L is clocked using the function p:
$$(L_0^{t+1}, \ldots, L_{\lceil \log 4b \rceil}^{t+1}) := (L_1^t, \ldots, L_{\lceil \log 4b \rceil - 1}^t, p(L^t)) \; .$$

Computation of the output. Once initialized, the state of QUARK is updated $4b$ times, and the output is the final value of the NFSR's X and Y, using the same bit ordering as for the initialization.

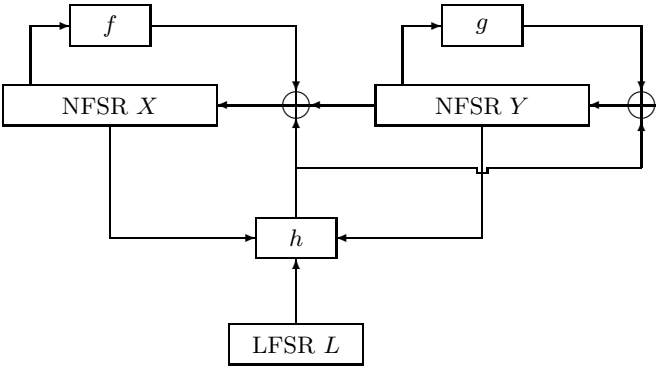

Fig. 2. Diagram of the permutation of QUARK (for clarity, the feedback of the LFSR with the function p is omitted)

2.3 Proposed Instances

There are three different flavors[3] of QUARK: U-QUARK, D-QUARK, and T-QUARK. For each, we give its rate r, capacity c, width b, digest length n, and its functions f, g, and h. The function p, used by the data-independent LFSR, is the same for all three instances: given a 10-bit register L, p returns $L_0 + L_3$.

U-QUARK is the lightest flavor of QUARK. It was designed to provide *64-bit security*, and to admit a parallelization degree of 8. It has sponge numbers $r = 8, c = 128, b = 2 \times 68, n = 128$.

Given a 68-bit register X, f returns

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

Given a 68-bit register Y, g returns

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

[3] In particle physics, the u-quark is lighter than the d-quark, which itself is lighter than the t-quark; our eponym hash functions compare similarly.

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

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

D-QUARK is the second-lightest flavor of QUARK. It was designed to provide *80-bit security*, and to admit a parallelization degree of 8. It has sponge numbers $r = 16, c = 160, b = 2 \times 88, n = 160$.

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.

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.

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

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

T-QUARK is the less light flavor of QUARK. It was designed to provide *112-bit security*, and to admit a parallelization degree of 16. It has sponge numbers $r = 32, c = 224, b = 2 \times 128, n = 224$.

T-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.

T-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.

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

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

3 Design Rationale

3.1 Single Security Level

An originality of QUARK is that its expected security level against second preimages differs from its digest length (see §4.2 for a description of the generic attack). In particular it offers a similar security against generic collision attacks and generic second preimage attacks, that is, approximately $2^{c/2}$. Note that the sponge construction ensures a resistance of approximately $\max(c, n)$ bits against

preimages, as recently shown in [14, §4.2][4]. Hence, QUARK provides increased resistance to preimage attacks.

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 of 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 146 memory bits (i.e., the two NFSR's 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.

3.2 Sponge Construction

The sponge construction [13] seems the only real alternative to the classical Merkle-Damgård construction based on a compression function (although several "patched" versions were proposed, adding counters, finalization functions, etc.). It rather relies on a *single permutation*, and message blocks are integrated with a simple XOR with the internal state. Sponge functions do not require storage of message blocks nor of "feedforward" intermediate values, as in Davies-Meyer constructions, however they need a larger state to achieve traditional security levels, which compensates those memory savings.

The sponge construction was proven indifferentiable from a random oracle (up to some bound) when instantiated with a random permutation or transformation, 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[5].

3.3 Permutation Algorithm

We now briefly justify our design choices regarding the permutation P. To avoid "reinventing the wheel", we borrowed most design ideas from the stream cipher Grain and from the block cipher KATAN, as detailed below.

Grain. The stream cipher Grain-v1 was chosen in 2008 as one of the four "promising new stream ciphers" by the ECRYPT eSTREAM Project[6]. It consists of two 80-bit shift registers combined with three Boolean functions, which makes it one of the lightest designs ever. Grain's main advantages are its simplicity and its 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 §4.5), of the existence of differential characteristics [16], and of (conjecturally) statistical distinguishers for Grain-128 [17].

[4] The expected workload to find a preimage was previously estimated to $2^n + 2^{c-1}$ in [15, §5.3], although that was not proven optimal.

[5] See the interactive page "Tune KECCAK to your requirements" at `http://keccak.noekeon.org/tune.html`

[6] See `http://www.ecrypt.eu.org/stream`

KATAN. The block cipher family KATAN [18] (CHES 2009) is inspired by the stream cipher Trivium [19] and builds a keyed permutation with two NFSR's 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. KATAN's use of two NFSR's with short feedback delay 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.

Taking the best of both. Based on the above observations, QUARK borrows the following design decisions from Grain:

- 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.

And KATAN inspired us in choosing

- Two NFSR's instead of a NFSR and a 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 dissimetry 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.

Choice of the Boolean functions. The quality of the Boolean functions in P determines its security. We thus first chose the functions in QUARK according to their individual properties, according to known metrics (see, e.g., [20]). The final choice was made by observing the empirical resistance of the *combination* of the three functions to known attacks (see §4.3-4.4).

In QUARK, we chose f and g functions similar to the non-linear function of Grain-v1. These functions achieve good, though suboptimal, non-linearity and resilience (see Table 1). They have degree six and include monomials of each degree below six. Note that having a relatively high degree (e.g., six rather than two) is cheap, since logical AND's need fewer gates than XOR's (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.

Table 1. Properties of the Boolean functions of each QUARK instance (for h, we consider that the parameter L_0 is zero)

Hash function	Boolean function	Var.	Deg.	Non-lin. (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
T-QUARK	h	16	3	20480 (32640)	10

4 Preliminary Security Analysis

4.1 The Hermetic Sponge Strategy

Like for the hash function KECCAK [4], we follow 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 implies a distinguisher for its permutation P (but a distinguisher for P does not necessarily leads to an attack on QUARK). This reduces the security of P to that of the hash function that uses it.

More precisely, the indifferentiability proof of the sponge construction ensures an expected complexity at least $\sqrt{\pi}2^{c/2}$ against any *differentiating attack*, independently of the digest length (of course, practical attacks of complexity below that bound exist when short digests are used, but these apply as well to a random oracle). This proof covers collision and (second) preimage attacks, as well as more specific attacks as multicollision or herding attacks [21]. As QUARK follows the hermetic sponge strategy, the proof of [13] is directly applicable.

4.2 Generic Second Preimage Attack

The generic second preimage attack against QUARK is similar to the generic preimage attack against the hash function CubeHash [5], which was described in [22] and discussed in [23]. It has complexity equivalent to more than $2^{c/2+1}$ evaluations of P and so 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 T-QUARK respectively.

4.3 Resistance to Cube Attacks and Cube Testers

The recently proposed "cube attacks" [24] and "cube testers" [25] are higher-order differential cryptanalysis techniques that exploit weaknesses in the algebraic structure of a cryptographic algorithm. 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 [17] to the stream cipher Grain-128 [26]. Cube attacks/testers are thus tools of choice to attack (reduced version of) QUARK's permutation, since it resembles Grain-128.

Recall that QUARK targets security against any nontrivial structural distinguisher for its permutation P. We thus applied cube testers—i.e., distinguishers—rather than cube attacks—i.e., key recovery attacks—to the permutation of each QUARK flavor. We followed a methodology inspired by [17], 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 try our distinguishers. Table 2 reports our results.

Table 2. 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	111	114 (21.0 %)
d-QUARK	704	144	144	148 (21.0 %)
t-QUARK	1024	213	220	222 (21.7 %)

One observes from Table 2 that all QUARK flavors showed a similar resistance to cube testers, with a fraction $\approx 21\,\%$ of the total number of rounds attacked with an effort 2^{24}. How many rounds would we break with a higher complexity? It cannot be determined analytically (to our present knowledge), however heuristical arguments can be given, based on previous results [24, 25, 17]: 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 t-QUARK in 2^{112} (26.5 %). Therefore, QUARK is unlikely to be broken by cube attacks or cube testers.

Note that 220 of Grain-128's 256 rounds could be attacked in [17] in 2^{24}; this result, however, should not be compared to the value 222 reported in Table 2, 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. One could thus attack at least $220 + 127 = 347$ rounds of Grain-128 by observing any bit of the internal state. Therefore, although t-QUARK uses registers of same length as Grain-128, it is significantly more resistant to cube testers.

4.4 Resistance to Differential Attacks

Differential attacks covers all attacks that exploit nonideal propagation of differences in a cryptographic algorithm (or components thereof). A large majority

of attacks on 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, which means in our case to analyze the permutation of Quark against differential distinguishers.

We consider a standard attack model, where the initial state is assumed chosen uniformly at random and where we seek differences in the initial state that give biased differences in the state obtained after the (reduced-round) permutation. We focus on "truncated" differentials in which the output difference concerns a small subset of bits (e.g., a single bit), because 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 t-Quark. For higher number of rounds, however, it becomes difficult to manually track differences, and so an automated search is 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 3 presents our results, showing that we could attack approximately as many rounds with truncated differentials as with cube testers (see Table 2).

Table 3. 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	116	119 (21.9 %)
d-Quark	704	135	145	148 (21.0 %)
t-Quark	1024	206	211	216 (21.1 %)

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 [16]). However, such methods seem unlikely to apply to the $4b$ rounds of Quark's permutation. For example, observe that [16] presented a characteristic of probability 2^{-96} for the full 256-round Grain-128; for comparison, t-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.

4.5 Resistance to Slide Distinguishers

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 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 [27]; we thus call the above a *slide distinguisher*.

The above idea is at the basis of "slide resynchronization" attacks on the stream cipher Grain [16, 28], 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 of one or more clocks, and such that the shift persists through the $4b$ rounds.

5 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 types of designs are preferable for such platforms. We thus focus 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.

5.1 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 extremely easy to implement in hardware; and third, the possibility of several space/time implementation trade-offs. Based on the two extremal trade-off choice, we designed two architecture variants of U-QUARK, D-QUARK, and T-QUARK:

- **Serial:** 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 corresponds to the most compact designs. They contain the minimal circuitry needed to handle incoming messages and to generate the correct output digests.

– **Parallel:** The number of the implemented permutation modules corresponds to the parallelization degree given in §2.3. 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 automatically 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.

5.2 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 [10, 18], 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 ([7] suggests a maximum of 27 µW). Table 4 reports the performance metrics obtained from our simulations at 100 kHz.

5.3 Discussion and Comparison with PRESENT-Based Designs

As reported in Table 4, the three serial designs need 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.

As reported in Table 4, DM-PRESENT-80/128 and H-PRESENT-128 also offer implementation trade-offs. For a same (second) preimage resistance of at least 64 bits, u-QUARK fits in a smaller area, 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. This may be due to its high security margin (note that 26 of the 31 rounds of PRESENT, as a block cipher, were attacked [29], suggesting a thin security margin against distinguishers in the "open key" model of hash functions).

Table 4. Compared hardware performance of PRESENT-based and QUARK lightweight hash functions. Security is expressed in bits (e.g., "128" in the "Pre." column means that preimages can be found within approximately 2^{128} calls to the function). Throughput and power consumption are given for a frequency of 100 kHz.

Hash function	Security		Block	Area[a]	Lat.	Thr.	Power [µW]	
	Pre.	Coll.	[bits]	[GE]	[cycles]	[kbps]	Mean	Peak
DM-PRESENT-80	64	32	80	1600	547	14.63	1.83	-
DM-PRESENT-80	64	32	80	2213	33	242.42	6.28	-
DM-PRESENT-128	64	32	128	1886	559	22.90	2.94	-
DM-PRESENT-128	64	32	128	2530	33	387.88	7.49	-
H-PRESENT-128	128	64	64	2330	559	11.45	6.44	-
H-PRESENT-128	128	64	64	4256	32	200.00	8.09	-
u-QUARK	128	64	8	1379	544	1.47	2.44	2.96
u-QUARK×8	128	64	8	2392	68	11.76	4.07	4.84
d-QUARK	160	80	16	1702	704	2.27	3.10	3.95
d-QUARK×8	160	80	16	2819	88	18.18	4.76	5.80
t-QUARK	224	112	32	2296	1024	3.13	4.35	5.53
t-QUARK×16	224	112	32	4640	64	50.00	8.39	9.79

[a] 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.

Acknowledgments

We would like to thank Gilles Van Assche for helpful comments.

References

1. Feldhofer, M., Rechberger, C.: A case against currently used hash functions in RFID protocols. In: Meersman, R., Tari, Z., Herrero, P. (eds.) OTM 2006 Workshops. LNCS, vol. 4277, pp. 372–381. Springer, Heidelberg (2006)
2. Preneel, B.: Status and challenges of lightweight crypto. Talk at the Early Symmetric Crypto (ESC) seminar (January 2010)
3. NIST: Cryptographic hash algorithm competition, `http://www.nist.gov/hash-competition`
4. Bertoni, G., Daemen, J., Peeters, M., Assche, G.V.: Keccak sponge function family main document. Submission to NIST, Round 2 (2009), `http://keccak.noekeon.org/Keccak-main-2.0.pdf`
5. Bernstein, D.J.: CubeHash specification (2.B.1). Submission to NIST, Round 2 (2009), `http://cubehash.cr.yp.to/submission2/spec.pdf`
6. Bernet, M., Henzen, L., Kaeslin, H., Felber, N., Fichtner, W.: Hardware implementations of the SHA-3 candidates Shabal and CubeHash. In: CT-MWSCAS. IEEE, Los Alamitos (2009)
7. Feldhofer, M., Wolkerstorfer, J.: Strong crypto for RFID tags - a comparison of low-power hardware implementations. In: ISCAS, pp. 1839–1842. IEEE, Los Alamitos (2007)
8. O'Neill, M.: Low-cost SHA-1 hash function architecture for RFID tags. In: Workshop on RFID Security RFIDsec. (2008)
9. Yoshida, H., Watanabe, D., Okeya, K., Kitahara, J., Wu, H., Kucuk, O., Preneel, B.: MAME: A compression function with reduced hardware requirements. In: ECRYPT Hash Workshop (2007)
10. 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)
11. 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)
12. Shamir, A.: SQUASH - a new MAC with provable security properties for highly constrained devices such as RFID tags. In: Nyberg, K. (ed.) FSE 2008. LNCS, vol. 5086, pp. 144–157. Springer, Heidelberg (2008)
13. 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)
14. Bertoni, G., Daemen, J., Peeters, M., Assche, G.V.: Sponge-based pseudo-random number generators. In: CHES (to appear, 2009)
15. Bertoni, G., Daemen, J., Peeters, M., Assche, G.V.: Sponge functions, `http://sponge.noekeon.org/SpongeFunctions.pdf`
16. Cannière, C.D., Kücük, O., Preneel, B.: Analysis of Grain's initialization algorithm. In: SASC 2008 (2008)
17. Aumasson, J.P., Dinur, I., Henzen, L., Meier, W., Shamir, A.: Efficient FPGA implementations of highly-dimensional cube testers on the stream cipher Grain-128. In: SHARCS (2009)
18. Cannière, C.D., Dunkelman, O., Knezevic, 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)

19. Cannière, C.D., Preneel, B.: Trivium. In: Robshaw, M.J.B., Billet, O. (eds.) New Stream Cipher Designs. LNCS, vol. 4986, pp. 84–97. Springer, Heidelberg (2008)
20. Sarkar, P., Maitra, S.: Construction of nonlinear boolean functions with important cryptographic properties. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 485–506. Springer, Heidelberg (2000)
21. Kelsey, J., Kohno, T.: Herding hash functions and the Nostradamus attack. In: Vaudenay, S. (ed.) EUROCRYPT 2006. LNCS, vol. 4004, pp. 183–200. Springer, Heidelberg (2006)
22. Bernstein, D.J.: CubeHash appendix: complexity of generic attacks. Submission to NIST (2008), `http://cubehash.cr.yp.to/submission/generic.pdf`
23. Aumasson, J.-P., Brier, E., Meier, W., Naya-Plasencia, M., Peyrin, T.: Inside the hypercube. In: Boyd, C., Nieto, J.M.G. (eds.) ACISP 2009. LNCS, vol. 5594, pp. 202–213. Springer, Heidelberg (2009)
24. Dinur, I., Shamir, A.: ube attacks on tweakable black box polynomials. In: Joux, A. (ed.) EUROCRYPT 2009. LNCS, vol. 5479, pp. 278–299. Springer, Heidelberg (2010)
25. Aumasson, J.P., Dinur, I., Meier, W., Shamir, A.: Cube testers and key recovery attacks on reduced-round MD6 and Trivium. In: Dunkelman, O. (ed.) Fast Software Encryption. LNCS, vol. 5665, pp. 1–22. Springer, Heidelberg (2009)
26. Hell, M., Johansson, T., Maximov, A., Meier, W.: A stream cipher proposal: Grain-128. In: IEEE International Symposium on Information Theory, ISIT 2006 (2006)
27. Biryukov, A., Wagner, D.: Slide attacks. In: Knudsen, L. (ed.) FSE 1999. LNCS, vol. 1636, pp. 245–259. Springer, Heidelberg (1999)
28. Lee, Y., Jeong, K., Sung, J., Hong, S.: Related-key chosen IV attacks on Grain-v1 and Grain-128. In: Mu, Y., Susilo, W., Seberry, J. (eds.) ACISP 2008. LNCS, vol. 5107, pp. 321–335. Springer, Heidelberg (2008)
29. 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)