Heavy Quark for secure AEAD

Jean-Philippe Aumasson[1], Simon Knellwolf[2], and Willi Meier[2]

[1] NAGRA, Switzerland
[2] FHNW, Switzerland

Abstract. Lightweight primitives are generally limited to 80- or 128-bit security, because lightweight applications seldom need more than this. However, non-lightweight platforms like multimedia systems-on-chip would also greatly benefit from a smaller hardware footprint, as it reduces development and integration costs, and leaves more circuit area to another component, or to add another functionality. Such systems sometimes need up to 256-bit security, for example to ensure a consistent security level across primitives. This paper thus breaks with the tradition and proposes a 256-bit authenticated encryption scheme with associated data (AEAD), based on the lightweight design Quark. We create a new Quark instance to use in a custom SpongeWrap mode, offering one-pass AEAD supporting arbitrary interleaving of encrypted and associated data, as well as a range of trade-offs between security and usage limit. More than a new primitive, this work provides insights on the scalability of lightweight designs to higher security levels: our new design c-Quark has internal state of 384 bits, and allows the implementation of 256-bit AEAD with in the order of 4000 GE.

1 Introduction

A recent research trend is the design of lightweight hash functions, with proposals Quark [1], photon [2], and spongent [3], all three being sponge functions. Those designs were preceded by reduced versions of Keccak [4] as well as by hash functions based on the block cipher present [5]. The above sponge functions achieve a competitive ratio between security and hardware footprint, due to a second preimage resistance of $n/2$ rather than n bits. In addition to this resource-efficiency, those designs can easily be used to construct other primitives than hash functions like stream ciphers, MACs, key derivation functions, or authenticated encryption schemes with associated data (AEADs).

This paper presents c-Quark, a new Quark instance with state size $b = 384$ bits, and a dedicated AEAD mode based on the SpongeWrap construction [6] of Bertoni et al. Compared to the original general definition of SpongeWrap, our scheme explicitly defines nonce management as well as a padding rule and a usage limit. This instance of SpongeWrap is called c-QuarkWrap. Security of at least 253 bits is ensured against adversaries limited to 2^{64} queries with a given key. Lower bounding to 253 rather than 256 bits simplifies a lot the construction.

We wrote a VHDL description of the new instance c-Quark, and simulated its performance in 90 nm ASIC. We estimate that the compact architecture fits in approximately 4000 gate-equivalents. Both our serial and parallel architectures show combinations of security, area, and speed that are competitive with those of previous designs.

Similar AEAD schemes can be easily constructed for (reduced) Keccak, photon, or spongent. A thorough comparison of the security and efficiency of all those schemes remains to be done.

2 Specification of c-Quark

c-Quark is a new instance of the Quark family, with parameters $r = 64, c = 320, b = 384, n = 384$. Here r is the *rate* (or block size), c is the *capacity* (or security), b is the width (or state size), and n is the digest's size. The description below is succinct and we refer to [1] for more details.

Like previous Quark instances, c-Quark processes a message m in three steps:

Version date: 2012.08.15

(1) **Initialization**: m 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.
(3) **Squeezing phase**: the last r bits of the state are returned as output, interleaved with applications of P until n bits are returned.

Unlike previous QUARK instances, C-QUARK's permutation P makes $2b$ rounds rather than $4b$ (see §§4.2). Note that we denote the internal sponge state $s = (s_0, \ldots, s_{b-1})$, where s_0 is the *first* bit of the state. The IV of C-QUARK is given in Appendix A.

The P permutation of C-QUARK is composed of three feedback shift registers (see Fig. 1):

- An NFSR X of 192 bits, denoted $X^t = (X_0^t, \ldots, X_{191}^t)$, at epoch $t \geq 0$.
- An NFSR Y of 192 bits, denoted $Y^t = (Y_0^t, \ldots, Y_{191}^t)$.
- An LFSR L of 16 bits, denoted $L^t = (L_0^t, \ldots, L_{15}^t)$.

Given a b-bit input and the internal state s of the sponge function, P is initialized as follows:

- $(X_0^0, \ldots, X_{191}^0) := (s_0, \ldots, s_{191})$.
- $(Y_0^0, \ldots, Y_{191}^0) := (s_{192}, \ldots, s_{383})$.
- $(L_0^0, \ldots, L_{15}^0) := (1, \ldots, 1)$.

Then the state update works as follows to compute $(X^{t+1}, Y^{t+1}, L^{t+1})$ from (X^t, Y^t, L^t):

(1) The function h is evaluated upon input bits from X^t, Y^t, and L^t, and the result is denoted h^t: $h^t := h(X^t, Y^t, L^t)$.
(2) X is clocked: $(X_0^{t+1}, \ldots, X_{191}^{t+1}) := (X_1^t, \ldots, X_{191}^t, Y_0^t + f(X^t) + h^t)$.
(3) Y is clocked: $(Y_0^{t+1}, \ldots, Y_{191}^{t+1}) := (Y_1^t, \ldots, Y_{191}^t, g(Y^t) + h^t)$.
(4) L is clocked: $(L_0^{t+1}, \ldots, L_{15}^{t+1}) := (L_1^t, \ldots, L_{15}^t, p(L^t))$.

In C-QUARK, the LFSR is of 16 bits (against 10 in previous instances), to facilitate the $\times 32$ parallel implementation. The feedback polynomial is thus adapted, so that $p(L^t)$ returns $L_0^t + L_2^t + L_3^t + L_5^t$ rather than $L_0^t + L_3^t$. The functions f, g, and h are given in Appendix B.

Once initialized, the state of QUARK is updated $2b$ 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

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

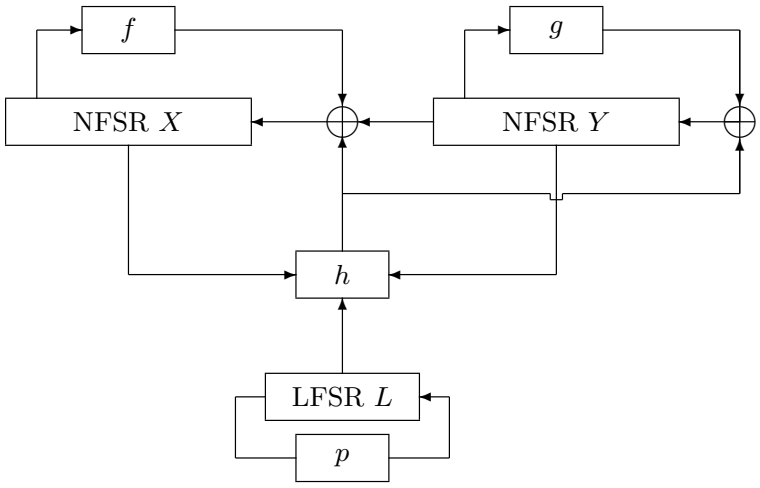

Fig. 1. Diagram of the permutation of QUARK.

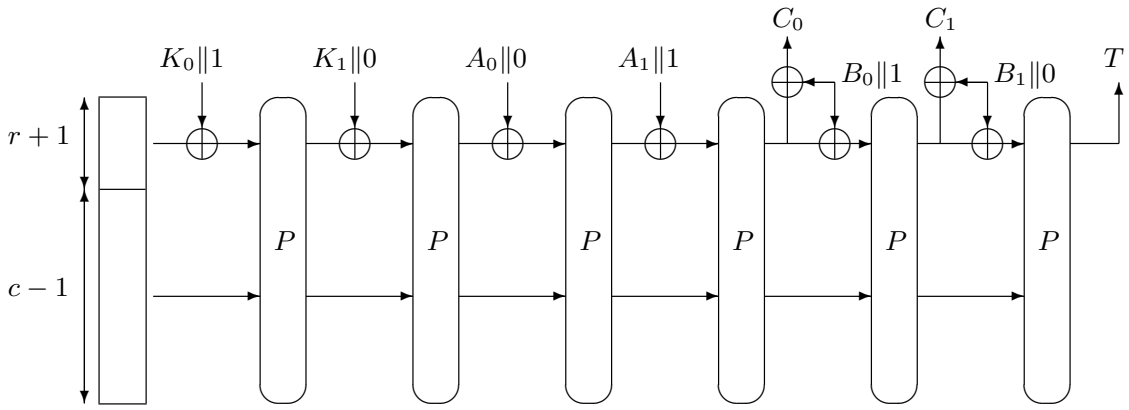

Fig. 2. Generic SpongeWrap mode a $2r$-bit (padded) key K, a $2r$-bit (padded) header A, and a $2r$-bit (padded) body B, returning an r-bit tag T.

3 Authenticated encryption with c-Quark

We construct an authenticated encryption scheme with associated data (AEAD) by using c-Quark in SpongeWrap mode [6, §5], an AEAD construction based on the duplex construction. Below we briefly describe the original SpongeWrap and then how to refine it to specify a concrete AEAD based on c-Quark; for convenience of reference we name this construction c-QuarkWrap.

3.1 The original SpongeWrap mode

The SpongeWrap mode is essentially the sponge construction wherein

(1) The key K is absorbed,
(2) Data to be absorbed is given as pairs (A, B), where the *header* A is returned unencrypted, and the *body* B is encrypted by xoring each r-bit block with the r-bit block squeezed from the current internal state.
(3) A tag T is returned as the r-bit blocks squeezed after the last B block processed (empty blocks are processed if more than r bits are needed).

Moreover, each block absorbed is flagged by a *frame bit* (see [6, §§5.1]), set to '1' for

- All key blocks except the last,
- The last header block,
- All body blocks except the last,

and it is set to '0' for all other blocks. This frame bit is absorbed by extending the original rate r of one bit.

The SpongeWrap construction supports an arbitrary number of (A, B) "wraps", and thus can interleave associated and encrypted data, and can return intermediate authentication tags. We refer to Algorithm 3 in [6] for a complete, formal, description of SpongeWrap.

Fig. 2 shows an example with a $2r$-bit (padded) key K, a $3r$-bit (padded) header A, and a $2r$-bit (padded) body B, returning an r-bit tag T.

3.2 The c-QuarkWrap AEAD

We propose to use the SpongeWrap mode with c-Quark to construct an AEAD supporting interleaved encrypted and (unencrypted) associated data. As SpongeWrap is very generic, specific parameters have to be specified. The version of SpongeWrap with our particular parameters is called c-QuarkWrap.

Padding. The SpongeWrap mode supports any sponge-compliant padding rule [6, Def.1]. c-QuarkWrap uses the same padding as for c-Quark in sponge mode, namely, to append a '1' bit followed by zero to $r-1$ zeroes. This is the simplest sponge-compliant padding rule. Note that each block processed has to be padded, as per the definition of the duplex construction (see Algorithm 2 in [6])

Key. SpongeWrap supports any key length. c-QuarkWrap keys are of 256 bits, and are thus absorbed as four 64-bit blocks, each followed by the frame bit and a padding bit.

Usage exponent. [7, §§5.2] defines the *usage exponent* as the value a such that the implementation imposes an upper limit of 2^a uses of given key. c-QuarkWrap sets this limit to 2^{64}. As c-QuarkWrap has an actual capacity of $384-61-1-1=318$ (due to the duplex padding and to the frame bit) this guarantees a security of at least $2^{318-1}/2^{64}=2^{253}$ (see [7] for a proof).

Nonces. SpongeWrap does not specify how nonces should be handled (if at all). c-QuarkWrap takes a 64-bit nonce, and each value should be used at most once within a same c-QuarkWrap object. As noted in [6, §§2.2], a nonce repetition implies that an adversary can learn the xor of two plaintext bodies, but does not affect the authentication functionality. As there are more distinct nonces than uses allowed by the usage exponent, outpassing the latter does not imply a nonce repetition.

After the SpongeWrap object is initialized with the key, the 64-bit nonce is copied to A_0, that is, the first bits of the first header processed. A nonce should be specified for each wrap call. Of course if the nonce is identical for consecutive wrap calls, only one copy of it can be transmitted along.

Tag length. SpongeWrap supports the generation of tags of arbitrary length. c-QuarkWrap returns 64-bit tags, which is sufficient for most applications. Longer tags can be produced by following the SpongeWrap specification.

Initial state. SpongeWrap relies on the duplex mode, which sets the initial state to the all-zero string. Instead, c-QuarkWrap uses a specific initial state (given in Appendix A), because an all-zero state may facilitate differential attacks on reduced-round versions of P.

3.3 Properties of c-QuarkWrap

As a particular version of SpongeWrap, c-QuarkWrap inherits its functional properties, which include

- Single-pass processing of associated and encrypted data.
- Support for arbitrarily interleaved associated and encrypted data.
- Generation of intermediate tags after each "wrap".
- Non-expanding encryption.
- Possibility to reuse the permutation to implement a hash function (or other cryptographic primitives).

Encrypting $8m$ bytes without any associated data requires $4+1+m$ calls to P, or $1+m$ if the secret internal state has been precomputed. Adding $8m'$ bytes of associated data adds m' calls to P.

Algorithm 1 c-QuarkWrap

function INIT(K)
  $s = IV$
  $s = P(s \oplus K_0 \| 11)$
  $s = P(s \oplus K_1 \| 01)$
  $s = P(s \oplus K_2 \| 01)$
  $s = P(s \oplus K_3 \| 01)$
end function

function AE(N, B) ▷ authenticated encryption of $B = B_0 \| \cdots \| B_w$ with nonce N
  $s = P(s \oplus N \| 11))$
  $C_0 = B_0 \oplus (s_{320 \ldots 383})$
  for $i = 0 \rightarrow w - 1$ **do**
    $s = P(s \oplus B_i \| 11)$
    $C_{i+1} = B_{i+1} \oplus (s_{320 \ldots 383})$
  end for
  $s = P(s \oplus B_w \| 01)$
  $T = s_{320 \ldots 383}$
  return $(N, C_0 \| \cdots \| C_w, T)$
end function

function AEAD(N, A, B) ▷ authenticated encryption of B with associated data $A_0 \| \cdots \| A_v$ and nonce N
  $s = P(s \oplus N \| 01)$
  for $i = 0 \rightarrow v - 1$ **do**
    $s = P(s \oplus A_i \| 01)$
  end for
  $s = P(s \oplus A_i \| 11)$
  $C_0 = B_0 \oplus (s_{320 \ldots 383})$
  for $i = 0 \rightarrow w - 1$ **do**
    $s = P(s \oplus B_i \| 11)$
    $C_{i+1} = B_{i+1} \oplus (s_{320 \ldots 383})$
  end for
  $s = P(s \oplus B_w \| 01)$
  $T = s_{320 \ldots 383}$
  return $(N, A, C_0 \| \cdots \| C_w, T)$
end function

4 Security

4.1 Security with ideal permutation

As previously observed, c-Quark is actually used in c-QuarkWrap with a capacity of 318 bits, due to the SpongeWrap frame bit and the duplex padding bit adding up to the 64 bits of effective data xored to the state. As per the analysis reported in [6, Th.1], this guarantees a security (with respect to confidentiality and authenticity) of approximately 2^{159} in the general case. However, by defining an upper bound of 2^{64} on the number of usages of a given key, a bound of 2^{253} on the complexity of a generic attack succeeding with high probability is obtained.

4.2 Security of the permutation

Reduced-round P permutations of previous Quark instances were attacked [1] with cube testers, truncated differential attacks, and conditional differentials attacks, with the latter method outperforming the first two. While the three original instances of Quark have $4b$ rounds, fewer than b rounds could be attacked. This security margin, along with the results of the analysis below, is the reason why we reduced the number of rounds to $2b$ for c-Quark.

We first applied the same kind of truncated differential analysis to c-Quark than to the original Quark's: 348 rounds could be distinguished from an ideal permutation with complexity 2^{20}, by exploiting a biased difference in s_0 given an input difference in s_3.

We then applied the more advanced technique of conditional differential cryptanalysis, with two specific approaches:

(1) Control the propagation of the single-input difference in s_3 as far as possible.
(2) Find an input difference (of arbitrary weight) that leads to a single-bit difference in the state bit s_3 after q rounds. After q rounds, the propagation is not controlled anymore.

The details and results of both approaches are described in the following.

Conditional Differential Cryptanalysis (1). The following conditions prevent the propagation of the difference whenever possible in the first 50 rounds (5 conditions in total):

In round 36:

$$s_{49} = 0,\ s_1 + s_9 + s_{14}s_{48}s_{66}s_{78}s_{95} + s_{14}s_{48} + s_{14}s_{78}s_{128}s_{160} + s_{14} + s_{18}s_{52}s_{70}s_{82}s_{99} + s_{18}s_{52} + s_{18}s_{82}s_{132}s_{164} + s_{18} + s_{26}s_{47}s_{196} + s_{26}s_{47}s_{252} + s_{29} + s_{30}s_{51}s_{200} + s_{30}s_{51}s_{256} + s_{33} + s_{35} + s_{39} + s_{41} + s_{45} + s_{47}s_{56}s_{196} + s_{47}s_{56} + s_{47}s_{196}s_{252} + s_{48}s_{66}s_{158}s_{160} + s_{51}s_{60}s_{200} + s_{51}s_{60} + s_{51}s_{200}s_{256} + s_{52}s_{70}s_{162}s_{164} + s_{56}s_{196} + s_{56}s_{252} + s_{60}s_{200} + s_{60}s_{256} + s_{66}s_{78}s_{95}s_{110}s_{128}s_{146} + s_{66}s_{78}s_{95} + s_{66} + s_{70}s_{82}s_{99}s_{114}s_{132}s_{150} + s_{70}s_{82}s_{99} + s_{70} + s_{78} + s_{82} + s_{86} + s_{90} + s_{95}s_{110}s_{146}s_{158} + s_{95}s_{110} + s_{95} + s_{99}s_{114}s_{150}s_{162} + s_{99}s_{114} + s_{99} + s_{110}s_{128}s_{146}s_{158}s_{160} + s_{110} + s_{113} + s_{114}s_{132}s_{150}s_{162}s_{164} + s_{114} + s_{117} + s_{128}s_{146}s_{158} + s_{128} + s_{132}s_{150}s_{162} + s_{132} + s_{141} + s_{142} + s_{145} + s_{147} + s_{150} + s_{151} + s_{153} + s_{157} + s_{158}s_{160} + s_{158} + s_{162}s_{164} + s_{162} + s_{193} + s_{195} + s_{197} + s_{199} + s_{226} + s_{230} + s_{252} + s_{253} + s_{255} + s_{256} + s_{257} + s_{259} + s_{280} + s_{284} + s_{292} + s_{296} + s_{331} + s_{335} + s_{341} + s_{345} = 0$$

In round 38:

$$s_{183} + 1 = 0,\ s_{132}s_{147} + s_{165} + 1 = 0,\ s_5 + s_9 + s_{18}s_{52}s_{70}s_{82}s_{99} + s_{18}s_{52} + s_{18}s_{82}s_{132}s_{164} + s_{18} + s_{30}s_{51}s_{200} + s_{30}s_{51}s_{256} + s_{33} + s_{39} + s_{45} + s_{51}s_{60}s_{200} + s_{51}s_{60} + s_{51}s_{200}s_{256} + s_{52}s_{70}s_{162}s_{164} + s_{60}s_{200} + s_{60}s_{256} + s_{70}s_{82}s_{99}s_{114}s_{132}s_{150} + s_{70}s_{82}s_{99} + s_{70} + s_{82} + s_{90} + s_{99}s_{114}s_{150}s_{162} + s_{99}s_{114} + s_{99} + s_{114}s_{132}s_{150}s_{162}s_{164} + s_{114} + s_{117} + s_{132}s_{150}s_{162} + s_{132} + s_{145} + s_{146} + s_{150} + s_{151} + s_{157} + s_{162}s_{164} + s_{162} + s_{197} + s_{199} + s_{230} + s_{256} + s_{257} + s_{259} + s_{284} + s_{296} + s_{335} + s_{345} + 1 = 0$$

On a sample that satisfies these conditions we found a stronger bias on the difference in s_0 after 348 rounds (the bias can be reliably detected on samples of size 2^{12} at significance level $\alpha = 0.001$), but no additional was detected at later rounds.

Conditional Differential Cryptanalysis (2). Let $q = 30$, that is, we aim for a single bit difference in s_3 after 30 rounds. By backward computation with linearized update functions one can find an initial difference that has differences at the following bits (25 in total): s_1, s_3, s_4, s_5, s_{17}, s_{30}, s_{192}, s_{193}, s_{195}, s_{196}, s_{198}, s_{199}, s_{200}, s_{201}, s_{202}, s_{203}, s_{204}, s_{205}, s_{206}, s_{208}, s_{210}, s_{212}, s_{214}, s_{216}, and s_{218}. The following conditions make sure that this difference boils down to a single bit difference after 30 rounds (22 conditions in total):

In rounds 0 to 9:
$s_{25}s_{46}+s_{46}s_{55}+s_{46}s_{251}+s_{55} = 0$ $s_{229}s_{253}s_{287}s_{305}+s_{229}+s_{287}s_{326}s_{352} = 0$ $s_{26}s_{47}+s_{47}s_{56}+s_{47}s_{252}+s_{56} = 0$ $s_{231}s_{255}s_{289}s_{307}+s_{231}+s_{289}s_{328}s_{354} = 0$ $s_{28}s_{49}+s_{49}s_{58}+s_{49}s_{254}+s_{58} = 0$ $s_{51}s_{69}s_{81}s_{98}+s_{51}+s_{81}s_{131}s_{163} = 0$ $s_{29}s_{50}+s_{50}s_{59}+s_{50}s_{255}+s_{59} = 0$ $s_{233}s_{257}s_{291}s_{309}+s_{233}+s_{291}s_{330}s_{356} = 0$ $s_{30}s_{51}+s_{51}s_{60}+s_{51}s_{200}+s_{51}+s_{60} = 0$ $s_{31}s_{52}+s_{52}s_{61}+s_{52}s_{257}+s_{61} = 0$ $s_{32}s_{53}+s_{53}s_{62}+s_{53}s_{258}+s_{62} = 0$ $s_{33}s_{54}+s_{54}s_{63}+s_{54}s_{259}+s_{63} = 0$ $s_{34}s_{55} + s_{55}s_{64} + s_{55}s_{260} + s_{64} = 0$

In rounds 10 to 19:
$s_{35}s_{56} + s_{56}s_{65} + s_{56}s_{261} + s_{65} = 0$ $s_{36}s_{57} + s_{57}s_{66} + s_{57}s_{262} + s_{66} = 0$ $s_{38}s_{59} + s_{59}s_{68} + s_{59}s_{264} + s_{68} = 0$ $s_{40}s_{61} + s_{61}s_{70} + s_{61}s_{266} + s_{70} = 0$ $s_{64}s_{82}s_{94}s_{111} + s_{64} + s_{94}s_{144}s_{176} = 0$ $s_{42}s_{63} + s_{63}s_{72} + s_{63}s_{268} + s_{72} = 0$ $s_{44}s_{65} + s_{65}s_{74} + s_{65}s_{270} + s_{74} = 0$

In rounds 20 to 29:
$s_{46}s_{67} + s_{67}s_{76} + s_{67}s_{272} + s_{76} = 0$
$s_{48}s_{69} + s_{69}s_{78} + s_{69}s_{274} + s_{78} = 0$

A sample that satisfies all these conditions can be generated by setting the following bits of the initial state to zero (and assigning random values to the other bits): s_{46}, s_{47}, s_{49}, s_{50}, s_{51}, s_{52}, s_{53}, s_{54}, s_{55}, s_{56}, s_{57}, s_{58}, s_{59}, s_{60}, s_{61}, s_{62}, s_{63}, s_{64}, s_{65}, s_{66}, s_{67}, s_{68}, s_{69}, s_{70}, s_{72}, s_{74}, s_{76}, s_{78}, s_{81}, s_{94}, s_{229}, s_{231}, s_{233}, s_{287}, s_{289}, and s_{291}. Using a sample of size 2^{20}, a bias can always be detected in the difference of s_0 after $348 + 30 = 378$ rounds (at significance level $\alpha = 0.001$).

It is possible to choose a slightly larger q (and hence to further increase the number of rounds). Table 1 shows the weight of the input difference and the number of required conditions for some larger values of q. Not only grows the number of conditions quickly, but they also get very complicated, which means that large parts of the state must be fixed.

Table 1. Hamming weight of the input difference and number of conditions for different q (the bias can be detected after $348 + q$ rounds).

Controlled rounds (q)	30	32	34	36	38	40	42	44	46	48
Difference weight	25	28	32	35	37	39	36	38	40	42
Number of conditions	22	24	28	33	41	49	53	61	67	78

Conclusion. It seems very unlikely that the c-Quark permutation with $2b = 768$ rounds admits differential properties that can be exploited for an attack on the hash function or its use in the proposed AEAD mode. Using conditional differential cryptanalysis we could "attack" $348 + 48 = 396$ rounds, which leaves a reasonable security margin.

Obviously better distinguishers may exist on P, and "shortcut" attacks on c-Quark as a hash function may exist too. As a first step, cryptanalysts may consider reduced-round versions of c-Quark as well modified versions with weaker feedback functions. Although we do not claim resistance against related-key attacks, they could be considered in the security evaluation of c-Quark.

5 Hardware efficiency

5.1 Methodology

We wrote VHDL descriptions of a serial architecture of c-Quark (1-bit datapath; most compact, slowest) and of a parallel architecture (32-bit datapath; less compact, fastest). We synthesized them for a 90 nm TSMC technology using Synopsys DC Ultra (2011 version) with the tcbn90lphp standard cell library.

Compared to the hardware evaluation in [1], we report only a basic analysis with *pre-place-and-route* area evaluation for the c-Quark sponge function. Much more reliable results would be obtained from a working post-layout design. Extra logic and memory is necessary to implement the complete c-QuarkWrap mode.

5.2 Results

Below we report the main performance metrics for each of the architectures implemented, showing efficiency estimates competitive with other lightweight designs[3]:

Serial architecture. In this architecture the P permutation has a latency of $2b = 768$ cycles, and processes 64 bits. At 100 kHz, this is a throughput of $100/768 \times 64 \approx 8.33$ kbps. The synthesis of our RTL design gave a circuit of approximately 3125 gate-equivalents (GE); assuming a density of 80% after place-and-route, this gives an area of approximately 4000 GE, that is, an efficiency of approximately $8333/4000 \approx 2$ bps/GE.

Parallel architecture. In this architecture the P permutation has a latency of $2b/32 = 24$ cycles, and processes 64 bits. At 100 kHz, this is a throughput of $100/24 \times 64 \approx 266.67$ kbps. The synthesis of our RTL design gave a circuit of approximately 7100 gate-equivalents (GE); assuming a density of 80% after place-and-route, this gives an area of approximately 8875 GE, that is, an efficiency of approximately $266\,666/8875 \approx 30$ bps/GE.

References

1. Aumasson, J.P., Henzen, L., Meier, W., Naya-Plasencia, M.: Quark: a lightweight hash (full version). http://131002.net/quark/quark_full.pdf appeared in CHES 2010.
2. Guo, J., Peyrin, T., Poschmann, A.: The PHOTON family of lightweight hash functions. In: CRYPTO. (2011)
3. Bogdanov, A., Knezevic, M., Leander, G., Toz, D., Varici, K., Verbauwhede, I.: SPONGENT: A lightweight hash function. In: CHES. (2011)
4. Bertoni, G., Daemen, J., Peeters, M., Assche, G.V.: The Keccak SHA-3 submission. Submission to NIST (Round 3), http://keccak.noekeon.org/Keccak-submission-3.pdf (2011)
5. Bogdanov, A., Leander, G., Paar, C., Poschmann, A., Robshaw, M.J.B., Seurin, Y.: Hash functions and RFID tags: Mind the gap. In: CHES. (2008)
6. Bertoni, G., Daemen, J., Peeters, M., Assche, G.V.: Duplexing the sponge: single-pass authenticated encryption and other applications. http://sponge.noekeon.org/SpongeDuplex.pdf appeared in SAC 2011.
7. Bertoni, G., Daemen, J., Peeters, M., Assche, G.V.: On the security of the keyed sponge construction. http://sponge.noekeon.org/SpongeKeyed.pdf appeared in SKEW 2011.

A Initial state

The initial state of c-Quark is the SHA-384 digest of the string "c-quark", and like for previous Quark instances the first state bit s_0 (copied to X_0 in P) is defined as the most significant bit of the first byte of this value (`2b`):

```
3b4503ec7662c3cb30e00837ec8d38bbe5ff5acd6901a2495750f9198e2e3b5852dcaa1662b7dad65fcb5a8a1f0d5fcc
```

[3] For example, at a same 100 kHz frequency, the serial and parallel architectures of spongent-256/512/256 have respective efficiencies $350/5110 = 0.068$ bps/GE and $66490/9944 = 6.68$ bps/GE.

B Boolean functions

We define the Boolean functions f, g, and h used for computing the P permutation of c-Quark:

$f(X) = X_0+X_{13}+X_{34}+X_{65}+X_{77}+X_{94}+X_{109}+X_{127}+X_{145}+X_{157}+X_{140}+X_{159}X_{157}+X_{109}X_{94}+X_{47}X_{13}+X_{157}X_{145}X_{127}+X_{94}X_{77}X_{65}+X_{159}X_{127}X_{77}X_{13}+X_{157}X_{145}X_{109}X_{94}+X_{159}X_{157}X_{65}X_{47}+X_{159}X_{157}X_{145}X_{127}X_{109} + X_{94}X_{77}X_{65}X_{47}X_{13} + X_{145}X_{127}X_{109}X_{94}X_{77}X_{65}$

$g(Y) = Y_{i+21} + Y_{i+57} + Y_{i+60} + Y_{i+94} + Y_{i+112} + Y_{i+125} + Y_{i+133} + Y_{i+152} + Y_{i+157} + Y_{i+146} + Y_{i+159}Y_{i+157} + Y_{i+125}Y_{i+112} + Y_{i+36}Y_{i+21} + Y_{i+157}Y_{i+152}Y_{i+133} + Y_{i+112}Y_{i+94}Y_{i+60}+ Y_{i+159}Y_{i+133}Y_{i+94}Y_{i+21} + Y_{i+157}Y_{i+152}Y_{i+125}Y_{i+112} + Y_{i+159}Y_{i+157}Y_{i+60}Y_{i+36}+ Y_{i+159}Y_{i+157}Y_{i+152}Y_{i+133}Y_{i+125}+Y_{i+112}Y_{i+94}Y_{i+60}Y_{i+36}Y_{i+21}+Y_{i+152}Y_{i+133}Y_{i+125}Y_{i+112}Y_{i+94}Y_{i+60}$

$h(X, Y, L) = X_{i+25} + Y_{i+59} + Y_{i+3}X_{i+55} + X_{i+46}X_{i+55} + X_{i+55}Y_{i+59} + Y_{i+3}X_{i+25}X_{i+46}+ Y_{i+3}X_{i+46}X_{i+55} + Y_{i+3}X_{i+46}Y_{i+59} + X_{i+25}X_{i+46}Y_{i+59}L_i + X_{i+25}L_i + L_i + X_{i+4} + X_{i+28} + X_{i+40} + X_{i+85} + X_{i+112} + X_{i+141} + X_{i+146} + X_{i+152} + Y_{i+2} + Y_{i+33} + Y_{i+60} + Y_{i+62} + Y_{i+87} + Y_{i+99} + Y_{i+138} + Y_{i+148}$

C Test values

We give intermediate values of the c-Quark internal state when hashing the empty message, that is, after padding, the message block `80000000`:

Initial state after XOR with the message block `80000000`:

```
3b4503ec7662c3cb30e00837ec8d38bbe5ff5acd6901a2495750f9198e2e3b5852dcaa1662b7dad6dfcb5a8a1f0d5fcc
```

State after applying the only permutation of the absorbing phase:

```
b9a4d5653dff49af0e9c01c202e33ce30df6dc988a3f7df674ed10280b74152b0b7542795236945e1cb9770ee7c25fa9
```

States after each permutation of the squeezing phase:

```
9d4607ec0e3a744447d6f79343970a4986a6d7b5dcfa0b52f5ea3cbbc54ed1056eadbfe16ccfeafbdce2c9464578337c
97078af8b39dec11810d275fe1ee072aa766a82cffadf8e875df86c85802ebc68fa919f69aeb28e469c7e26cb4f1bdf4
eb09b18152c593c24e24b4313a92134ebe6e88099dfeeefc793f1165c9f1585910133da0b3fe393b4869f1a93639f1f3
57cec14c521600e91936829170737bfb66f9adf818abb10f6e44b1121a5916043a11a5706b4c987b60b888975ff9ffee
ea1477b135ff77cd78585224a2d224e9f6e48b812021bf68b02125f329d2310e731d0bee58c56b1b880d2c499108a27a
```

Digest returned:

```
1cb9770ee7c25fa9dce2c9464578337c69c7e26cb4f1bdf44869f1a93639f1f360b888975ff9ffee880d2c499108a27a
```