Heavy Quark for secure AEAD

Jean-Philippe Aumasson, Simon Knellwolf, Willi Meier

CHES 2010

QUARK
a lightweight hash

Jean-Philippe Aumasson

NAGRAVISION
KUDELSKI GROUP

with Luca Henzen, Willi Meier, María Naya-Plasencia

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

n|w University of Applied Sciences Northwestern Switzerland
School of Engineering

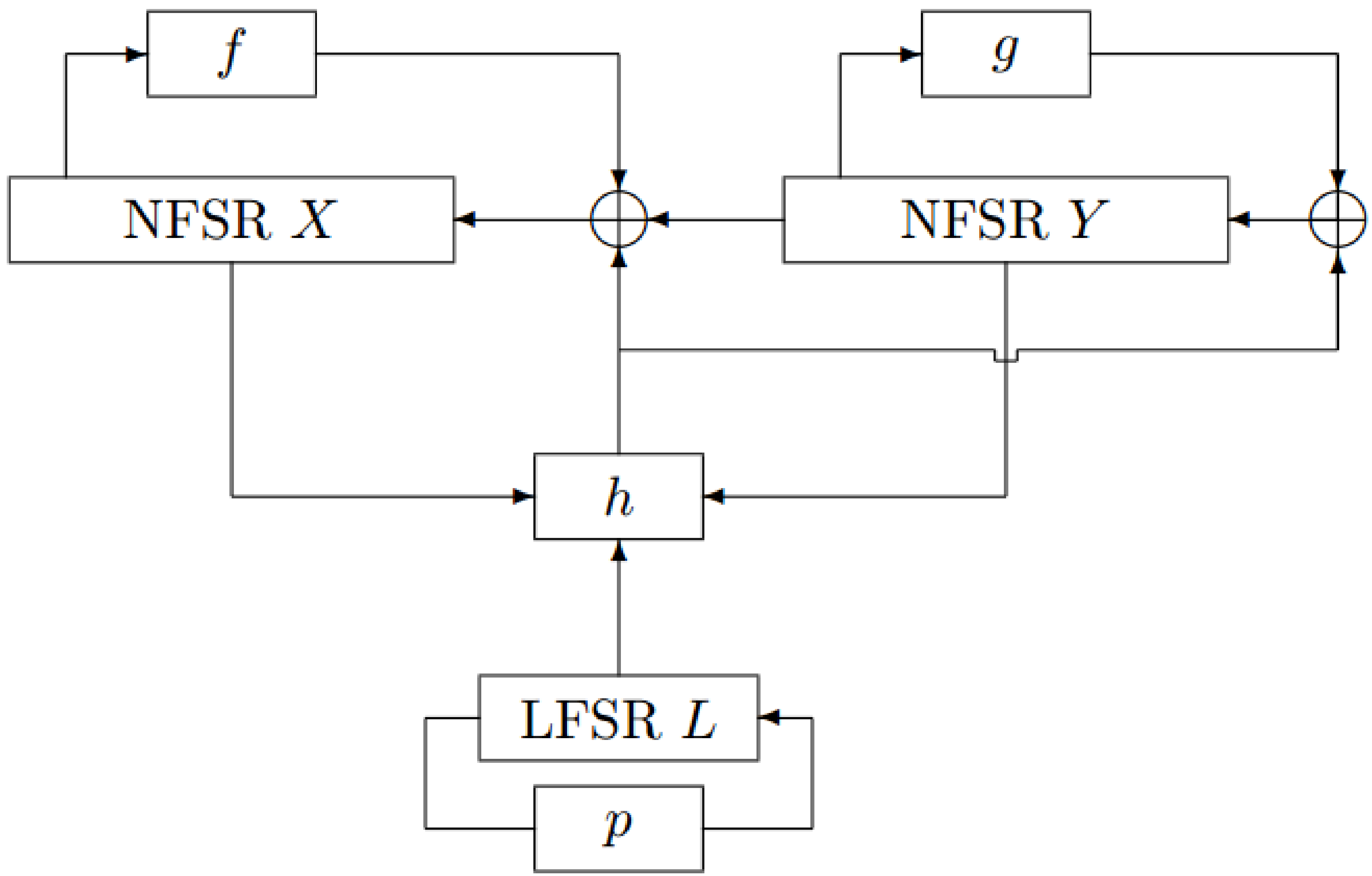
Quark

Lightweight sponge

NFSR-based (à la Grain, KATAN)

u-Quark, d-Quark, s-Quark

Architecture of Quark’s permutation



180nm ASIC simulations

Hash function	Security (bits)	Area (GE)	Speed (kbps)	Power (μW)
Compact architecture @100 kHz				
U-QUARK	64, 128	1379	1.47	2.96
D-QUARK	80, 160	1702	2.27	3.95
S-QUARK	112, 224	2296	3.13	5.53
High-speed architecture @714 MHz				Power (mW)
U-QUARK	64, 128	3032	84000	37.01
D-QUARK	80, 160	3561	130000	43.35
S-QUARK	112, 224	6220	357000	75.27

More lightweight sponges

PHOTON

(Guo, Peyrin, Poschmann; CRYPTO ‘11)

AES-like permutation

SPONGENT

(Bogdanov, Knežević, Leander, Tor, Varıcı, Verbauwhede; CHES ‘11)

PRESENT-like permutation

Quark had lower security than SHA-2

- 2^{nd} preimage ≤ 112 bits
- Preimage ≤ 224 bits

How does Quark's architecture scale to higher security?

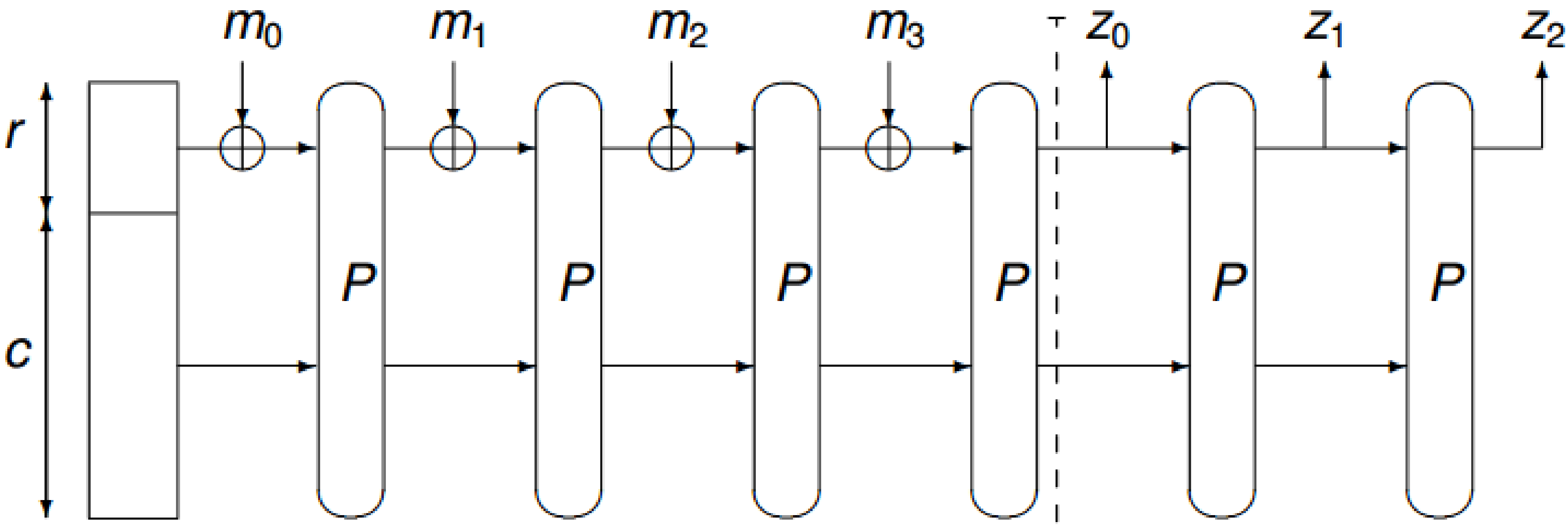
Quark had lower security than SHA-2

- 2^{nd} preimage ≤ 112 bits
- Preimage ≤ 224 bits

How small is a Quark-based circuit for high-security hash + auth'd encryption?

u-Quark, d-Quark, s-Quark, **c-Quark**

c-Quark: 384-bit state, 64-bit blocks



320-bit capacity

320-bit preimage security

160-bit 2nd preimage / collision security

c-Quark #rounds = 2 × state size

Distinguisher on **52%** or the #rounds

4 × state size, **20%** for previous Quarks

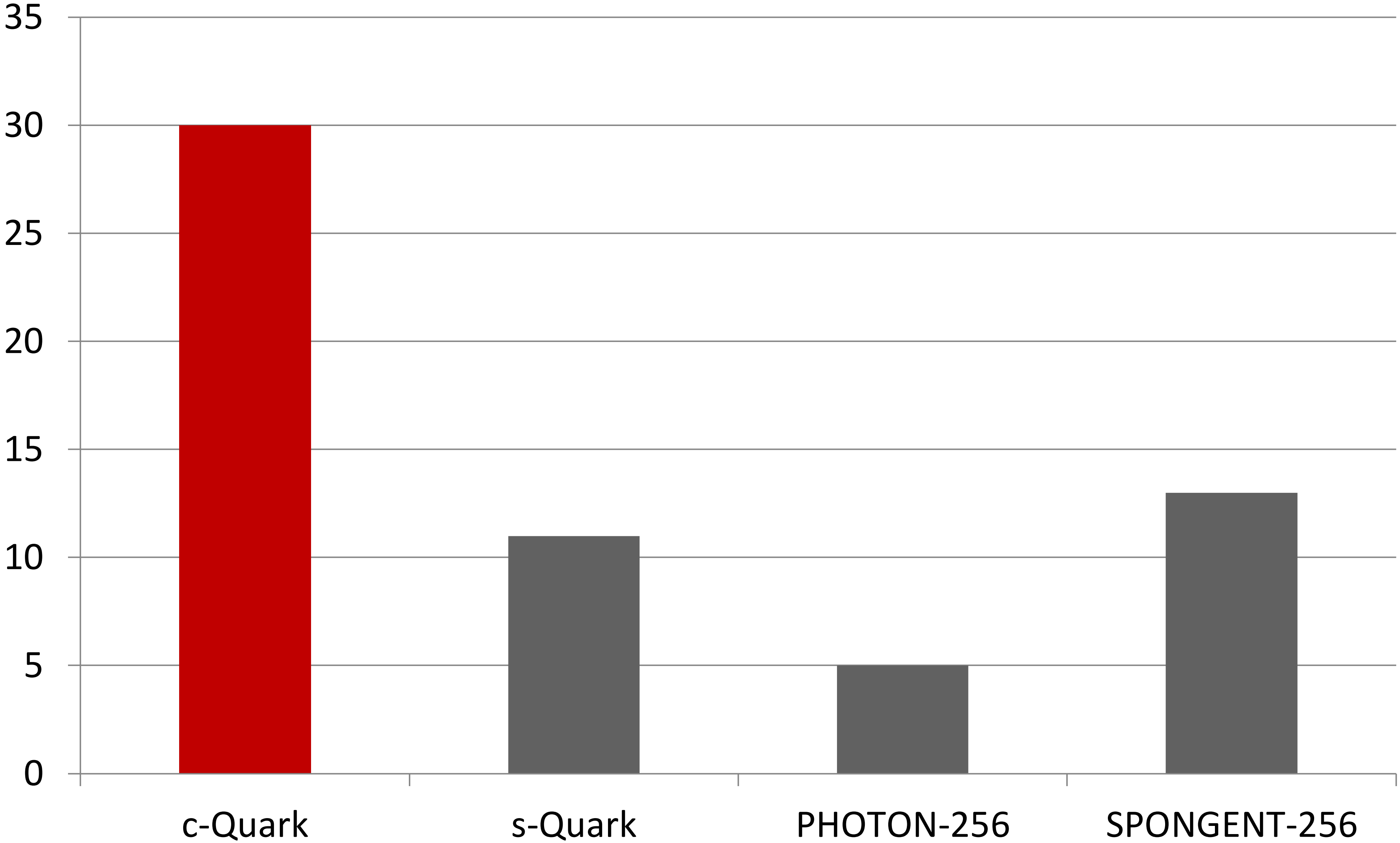
Compact hardware

≈ 4000 gates on 90nm TSMC

(estimate from pre-P&R simul, 80% density)

```
Y <= QuarkStatexDP(WWIDTH*8/2 to WWIDTH*8-1);

perm : for i in 0 to PDEG-1 generate

  Xn(i) <= Y(0) xor X(0) xor X(i+13) xor X(i+34) xor X(i+65) xor X(i+77) xor
           X(i+94) xor X(i+109) xor X(i+127) xor X(i+145) xor X(i+157) xor X(i+140) xor
           ( X(i+159) and X(i+157) ) xor ( X(i+109) and X(i+94) ) xor ( X(i+47) and X(i+13) ) xor
           ( X(i+157) and X(i+145) and X(i+127) ) xor (X(i+94) and X(i+77) and X(i+65) ) xor
           ( X(i+159) and X(i+127) and X(i+77) and X(i+13) ) xor
           ( X(i+157) and X(i+145) and X(i+109) and X(i+94) ) xor
           ( X(i+159) and X(i+157) and X(i+65) and X(i+47) ) xor
           ( X(i+159) and X(i+157) and X(i+145) and X(i+127) and X(i+109) ) xor
           ( X(i+94) and X(i+77) and X(i+65) and X(i+47) and X(i+13) ) xor
           ( X(i+145) and X(i+127) and X(i+109) and X(i+94) and X(i+77) and X(i+65) );

  Yn(i) <= Y(0) xor Y(i+21) xor Y(i+57) xor Y(i+60) xor Y(i+94) xor Y(i+112)  xor
           Y(i+125) xor Y(i+133) xor Y(i+152) xor Y(i+157) xor  Y(i+146) xor ( Y(i+159) and Y(i+157) ) xor
           ( Y(i+125) and Y(i+112) ) xor ( Y(i+36) and Y(i+21) ) xor
           ( Y(i+157) and Y(i+152) and Y(i+133) ) xor (Y(i+112) and Y(i+94) and Y(i+60) ) xor
           ( Y(i+159) and Y(i+133) and Y(i+94) and Y(i+21) ) xor
           ( Y(i+157) and Y(i+152) and Y(i+125) and Y(i+112) ) xor
           ( Y(i+159) and Y(i+157) and Y(i+60) and Y(i+36) ) xor
           ( Y(i+159) and Y(i+157) and Y(i+152) and Y(i+133) and Y(i+125) ) xor
           ( Y(i+112) and Y(i+94) and Y(i+60) and Y(i+36) and Y(i+21) ) xor
           ( Y(i+152) and Y(i+133) and Y(i+125) and Y(i+112) and Y(i+94) and Y(i+60) );

  h(i) <= LxDP(i) xor X(i+25) xor Y(i+59) xor ( Y(i+3) and X(i+55) ) xor (X(i+46) and X(i+55)) xor
          (X(i+55) and Y(i+59)) xor (Y(i+3) and X(i+25) and X(i+46) ) xor (Y(i+3) and X(i+46) and X(i+55) ) xor
          (Y(i+3) and X(i+46) and Y(i+59) ) xor (X(i+25) and X(i+46) and Y(i+59) and LxDP(i)) xor (X(i+25) and LxDP(i) ) xor
          X(i+4) xor X(i+28) xor X(i+40) xor X(i+85) xor X(i+112) xor X(i+141) xor X(i+146) xor X(i+152) xor
          Y(i+2) xor Y(i+33) xor Y(i+60) xor Y(i+62) xor Y(i+ 87) xor Y(i+ 99) xor Y(i+138) xor Y(i+148);

  Xnn(i) <= Xn(i) xor h(i);
  Ynn(i) <= Yn(i) xor h(i);
```

Efficiency (bps/GE) of parallel architectures @100KHz



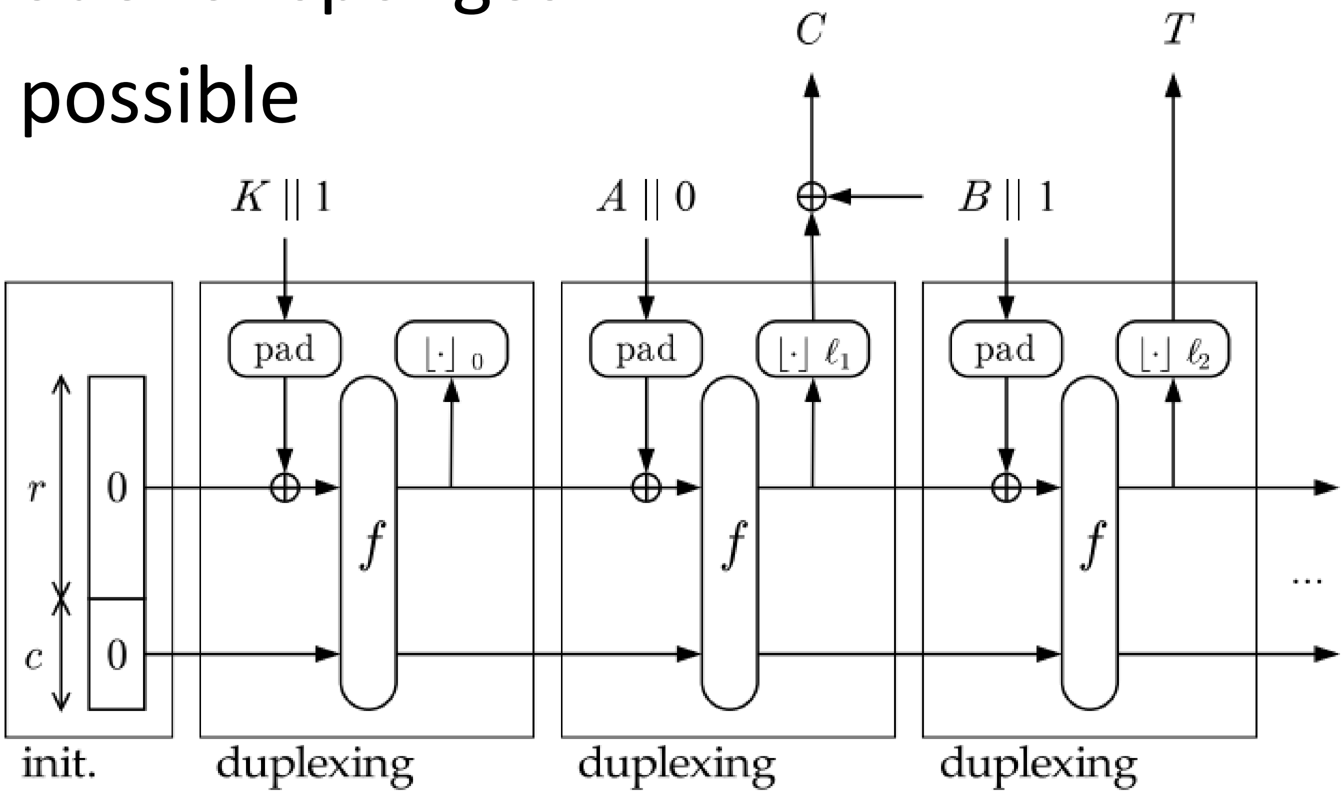
How to encrypt with c-Quark?

SpongeWrap

Bertoni, Daemen, Peeters, Van Asche; SHA-3 2010

General AEAD mode for sponges

Many trade-offs possible



How to best instantiate SpongeWrap?

QuarkWrap: tweaked SpongeWrap for c-Quark-based AEAD

- Explicit nonce support
- 64-bit tags (for minimal overhead)
- 64-bit nonces (should be unique)
- Nonce repetition does not affect authentication
- If each key is used at most 2^{64} times, **security of at least 253 bits**

Initialization with a 256-bit key

$$
\begin{aligned}
&\textbf{function } \textsc{Init}(K)\\
&\quad s = IV\\
&\quad s = P(s \oplus K_0 \| 11)\\
&\quad s = P(s \oplus K_1 \| 01)\\
&\quad s = P(s \oplus K_2 \| 01)\\
&\quad s = P(s \oplus K_3 \| 01)\\
&\textbf{end function}
\end{aligned}
$$

AE of B_0, B_1, .. B_w with nonce N

function AE(N, B)

$s = P(s \oplus N \| 11))$

$C_0 = B_0 \oplus (s_{320\ldots383})$

for $i = 0 \to w - 1$ **do**

$s = P(s \oplus B_i \| 11)$

$C_{i+1} = B_{i+1} \oplus (s_{320\ldots383})$

end for

$s = P(s \oplus B_w \| 01)$

$T = s_{320\ldots383}$

return $(N, C_0 \| \cdots \| C_w, T)$

end function

Conclusion:

is this the right way to go?

Probably **not** if you just want AEAD

Keyed sponges initialization is slow

High ratio cryptographic work / bit

Yes if several primitives are needed, and if speed requir’ts are reasonable

Can reuse the same permutation for hashing, AE, RNG, KDF, etc.

Similar construction for all functionalities (sponge/duplex)

B-o-t-E example:

c-Quark-based hash, MAC, AEAD:

≈ 5000 GE, 160- to 256-bit security

AES-128-CTR + HMAC-SHA-256:

≈ >3000 + >9000 = >**12000 GE, 128-bit security**

Heavy Quark for secure AEAD

Jean-Philippe Aumasson, Simon Knellwolf, Willi Meier