KANGAROO TWELVE

draft-viguier-kangarootwelve-01

Benoît Viguier¹
CFRG Meeting, March 19, 2018

¹Radboud University, Nijmegen, The Netherlands
What is KangarooTwelve?

- eXtendable Output Function
- Sponge construction
- Uses Keccak-p[1600, \( n_r = 24 \)]
- BUT no parallelism

SHAKE128
What is KangarooTwelve?

- eXtendable Output Function
- Tree on top of sponge construction
- Keccak-$p$ reduced from 24 to 12 rounds
- Parallelism grows automatically with input size
- No penalty for short messages
How secure is KangarooTwelve?

- Same security claim as SHAKE128: 128 bits of security
- Sponge generic security
- Parallel mode with proven generic security
  - [IJIS 2014] – Sufficient conditions for sound tree and sequential hashing modes
  - [ACNS 2014] – Sakura: A Flexible Coding for Tree Hashing
- Sponge function on top of Keccak-p[1600, \( n_r = 12 \)]
  - Round function unchanged
    - \( \Rightarrow \) cryptanalysis since 2008 still valid
  - Safety margin: from rock-solid to comfortable
Status of Keccak cryptanalysis

- Collision attacks up to 5 rounds
  - Also up to 6 rounds, but for non-standard parameters ($c = 160$)
  [Song, Liao, Guo, CRYPTO 2017]

- Stream prediction
  - in 8 rounds ($2^{128}$ time, prob. 1)
  - in 9 rounds ($2^{256}$ time, prob. 1)
  [Dinur, Morawiecki, Pieprzyk, Srebrny, Straus, EUROCRYPT 2015]

- Lots of third party cryptanalysis available at: https://keccak.team/third_party.html
How fast is KangarooTwelve?

- At least twice as fast as SHAKE128 on short inputs
- Much faster when parallelism is exploited on long inputs

<table>
<thead>
<tr>
<th></th>
<th>Short input</th>
<th>Long input</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel Core i5-4570 (Haswell)</td>
<td>3.68 c/b</td>
<td>1.44 c/b</td>
</tr>
<tr>
<td>Intel Core i5-6500 (Skylake)</td>
<td>2.89 c/b</td>
<td>1.22 c/b</td>
</tr>
<tr>
<td>Intel Core i7-7800X (Skylake-X)</td>
<td>2.35 c/b</td>
<td>0.55 c/b</td>
</tr>
</tbody>
</table>

Single core only.
Why is it interesting for the IETF?

- **Keccak/KangarooTwelve** is an open design
  - Public design rationale
  - Result of an open international competition
  - Long-standing active scrutiny from the crypto community
- **Best security/speed trade-off**
  - Speed-up w/o wasting cryptanalysis resources (no tweaks)
  - Proven generic security
- **Scalable parallelism**
  - As much parallelism as the implementation can exploit
  - Without parameter
Analyzing the sponge construction

Diagram showing the sponge construction with layers and functions labeled as 'f'. Directions of flow and labels such as 'outer', 'inner', and 'absorbing' are indicated.
Analyzing the sponge construction
Theorem 2. A padded sponge construction calling a random permutation, $S'[\mathcal{F}]$, is $(t_D, t_S, N, \epsilon)$-indistinguishable from a random oracle, for any $t_D, t_S = O(N^2)$, $N < 2^c$ and and for any $\epsilon$ with $\epsilon > f_P(N)$.

If $N$ is significantly smaller than $2^c$, $f_P(N)$ can be approximated closely by:

$$f_P(N) \approx 1 - e^{-\frac{(1-2^{-r})N^2 + (1+2^{-r})N}{2^{c+1}}} < \frac{(1 - 2^{-r})N^2 + (1 + 2^{-r})N}{2^{c+1}}. \quad (6)$$

[EuroCrypt 2008]

http://sponge.noekeon.org/SpongeIndifferentiability.pdf
Theorem 2. A padded sponge construction calling a random permutation, $S'[F]$, is $(t_D, t_S, N, \epsilon)$-indistinguishable from a random oracle, for any $t_D$, $t_S = O(N^2)$, $N < 2^c$ and for any $\epsilon$ with $\epsilon > f_P(N)$.

If $N$ is significantly smaller than $2^c$, $f_P(N)$ can be approximated closely by:

$$f_P(N) \approx 1 - e^{-\frac{(1-2^{-r})N^2+(1+2^{-r})N}{2^c+1}} < \frac{(1 - 2^{-r})N^2 + (1 + 2^{-r})N}{2^{c+1}}.$$  \hspace{1cm} (6)

[EuroCrypt 2008]

http://sponge.noekeon.org/SpongeIndifferentiability.pdf

Theorem, explained

$$\Pr[\text{attack}] \leq \frac{N^2}{2^{c+1}} \quad \text{(or so)}$$

$\Rightarrow$ if $N \ll 2^{c/2}$, then the probability is negligible
Two pillars of security in cryptography

- Generic security
  - Strong mathematical proofs
Two pillars of security in cryptography

- Generic security
  - Strong mathematical proofs
    - ⇒ scope of cryptanalysis reduced to primitive

- Security of the primitive
  - No proof!
    - ⇒ open design rationale
      - ⇒ cryptanalysis!
  - Confidence
    - ⇐ sustained cryptanalysis activity and no break
      - ⇐ proven properties
Two pillars of security in cryptography

- **Generic security**
  - Strong mathematical proofs
    - scope of cryptanalysis reduced to primitive

- **Security of the primitive**
  - No proof!
Two pillars of security in cryptography

- **Generic security**
  - Strong mathematical proofs
    - ⇒ scope of cryptanalysis reduced to primitive

- **Security of the primitive**
  - No proof!
    - ⇒ open design rationale
Two pillars of security in cryptography

- **Generic security**
  - Strong mathematical proofs
    ⇒ scope of cryptanalysis reduced to primitive

- **Security of the primitive**
  - No proof!
    ⇒ open design rationale
    ⇒ *cryptanalysis!*
Two pillars of security in cryptography

- Generic security
  - Strong mathematical proofs
    ⇒ scope of cryptanalysis reduced to primitive

- Security of the primitive
  - No proof!
    ⇒ open design rationale
    ⇒ third-party cryptanalysis!
Two pillars of security in cryptography

- Generic security
  - Strong mathematical proofs
    ⇒ scope of cryptanalysis reduced to primitive

- Security of the primitive
  - No proof!
    ⇒ open design rationale
    ⇒ lots of third-party cryptanalysis!
Two pillars of security in cryptography

▶ Generic security
  • Strong mathematical proofs
    ⇒ scope of cryptanalysis reduced to primitive

▶ Security of the primitive
  • No proof!
    ⇒ open design rationale
    ⇒ lots of third-party cryptanalysis!
  • Confidence
    ⇐ sustained cryptanalysis activity and no break
    ⇐ proven properties
## Impact of parallelism

<table>
<thead>
<tr>
<th>Keccak-$f[1600] \times 1$</th>
<th>1070 cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>Keccak-$f[1600] \times 2$</td>
<td>1360 cycles</td>
</tr>
<tr>
<td>Keccak-$f[1600] \times 4$</td>
<td>1410 cycles</td>
</tr>
</tbody>
</table>

CPU: Intel Core i5-6500 (Skylake) with AVX2 256-bit SIMD
Tree hashing

Example: **ParallelHash** [SP 800-185]

<table>
<thead>
<tr>
<th>function</th>
<th>instruction set</th>
<th>cycles/byte (^1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>KECCAK[c = 256] × 1</td>
<td>x86_64</td>
<td>6.29</td>
</tr>
<tr>
<td>KECCAK[c = 256] × 2</td>
<td>AVX2</td>
<td>4.32</td>
</tr>
<tr>
<td>KECCAK[c = 256] × 4</td>
<td>AVX2</td>
<td>2.31</td>
</tr>
</tbody>
</table>

CPU: Intel Core i5-6500 (Skylake) with AVX2 256-bit SIMD

\(^1\)for long messages.
Final node growing with kangaroo hopping and SAKURA coding

[ACNS 2014]