Cryptographic properties of KECCAK

Gilles VAN ASSCHE\textsuperscript{1}

\textsuperscript{1}STMicroelectronics

Indocrypt, New Delhi, December 2018

Based on joint work with Guido BERTONI, Joan DAEMEN, Silvia MELLA, Michaël PEETERS, Ronny VAN KEER
Outline

1 KECCAK and SHA-3 functions
2 Inside KECCAK-f
3 Trail analysis
   - Goal
   - Propagation in KECCAK-f
   - Generating all 3-round trail cores up to some weight
   - Cases $|K|N$, $|N|K$ and $|N|N$
   - Case $|K|K$
   - Results
4 Alignment
   - What is alignment?
   - Alignment experiments in KECCAK-f
   - Relevance of alignment
5 KECCAKTOOLS
6 Conclusions
Outline

1. **KECCAK and SHA-3 functions**
2. Inside KECCAK-\(f\)
3. Trail analysis
   - Goal
   - Propagation in KECCAK-\(f\)
   - Generating all 3-round trail cores up to some weight
   - Cases \(|K|N|\), \(|N|K|\) and \(|N|N|\)
   - Case \(|K|K|\)
   - Results
4. Alignment
   - What is alignment?
   - Alignment experiments in KECCAK-\(f\)
   - Relevance of alignment
5. KECCAKTOOLS
6. Conclusions
Keccak and SHA-3 functions

Keccak

Keccak is a sponge function ...

... that uses the Keccak-f permutation

\[ b = r + c \in \{25, 50, 100, 200, 400, 800, 1600\} \]
Keccak and SHA-3 functions

NIST FIPS 202 (August 2015)

- Four drop-in replacements to SHA-2
- Two extendable output functions (XOF)

<table>
<thead>
<tr>
<th>XOF</th>
<th>SHA-2 drop-in replacements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Keccak[c = 256](M</td>
<td></td>
</tr>
<tr>
<td>Keccak[c = 512](M</td>
<td></td>
</tr>
<tr>
<td></td>
<td>first 384 bits of Keccak[c = 768](M</td>
</tr>
<tr>
<td></td>
<td>first 512 bits of Keccak[c = 1024](M</td>
</tr>
<tr>
<td>SHAKE128 and SHAKE256</td>
<td>SHA3-224 to SHA3-512</td>
</tr>
</tbody>
</table>

- Toolbox for building other functions
Customized SHAKE (cSHAKE)

- $H(x) = \text{cSHAKE}(x, \text{name}, \text{customization string})$
- E.g., $\text{cSHAKE128}(x, N, S) = \text{Keccak}[c = 256](\text{encode}(N, S) || x || \theta \theta)$
- $\text{cSHAKE128}(x, N, S) \triangleq \text{SHAKE128}$ when $N = S = ""$

**KMAC**: message authentication code (no need for HMAC-SHA-3!)

$$\text{KMAC}(K, x, S) = \text{cSHAKE}(\text{encode}(K) || x, "KMAC", S)$$

**TupleHash**: hashing a sequence of strings $x = x_n \circ x_{n-1} \circ \cdots \circ x_1$

$$\text{TupleHash}(x, S) = \text{cSHAKE}(\text{encode}(x), "\text{TupleHash}" , S)$$
Keccak and SHA-3 functions

NIST SP 800-185 (December 2016)

ParallelHash: faster hashing with parallelism

<table>
<thead>
<tr>
<th>function</th>
<th>instruction set</th>
<th>cycles/byte</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
KANGAROO TWELVE: a fast variant of KECCAK

Similar to ParallelHash, but:
- Same permutation, reduced round count (12 instead of 24)
- Kangaroo hopping

[KECCAK Team, Viguier, ACNS 2018]
Other schemes using KECCAK-\(\rho\)

- **KETJE**: lightweight authenticated encryption
  KECCAK-\(\rho\)[200 or 400] in MonkeyDuplex

- **KEYAK**: authenticated encryption
  KECCAK-\(\rho\)[800 or 1600] in full-state keyed duplex

- **KRAVATTE**: pseudo-random function and full-feature AE
  KECCAK-\(\rho\)[1600] in Farfalle
Outline

1 KECCAK and SHA-3 functions
2 Inside KECCAK-f
3 Trail analysis
   ■ Goal
   ■ Propagation in KECCAK-f
   ■ Generating all 3-round trail cores up to some weight
   ■ Cases $|K|N|$, $|N|K|$ and $|N|N|$ 
   ■ Case $|K|K|
   ■ Results
4 Alignment
   ■ What is alignment?
   ■ Alignment experiments in KECCAK-f
   ■ Relevance of alignment
5 KECCAKTOOLS
6 Conclusions
Keccak-\( f \)

- The seven permutation army:
  - 25, 50, 100, 200, 400, 800, 1600 bits
  - toy, lightweight, fastest
  - standardized in \([\text{FIPS 202}]\)

- Repetition of a simple round function
  - that operates on a 3D state
  - \((5 \times 5)\) lanes
  - up to 64-bit each
The state: an array of $5 \times 5 \times 2^\ell$ bits

- $5 \times 5$ lanes, each containing $2^\ell$ bits (1, 2, 4, 8, 16, 32 or 64)
- $(5 \times 5)$-bit slices, $2^\ell$ of them
Inside KECCAK-\textit{f}

The state: an array of $5 \times 5 \times 2^\ell$ bits

- $5 \times 5$ lanes, each containing $2^\ell$ bits (1, 2, 4, 8, 16, 32 or 64)
- $(5 \times 5)$-bit slices, $2^\ell$ of them
The state: an array of $5 \times 5 \times 2^\ell$ bits

- **5 × 5 lanes**, each containing $2^\ell$ bits (1, 2, 4, 8, 16, 32 or 64)
- **(5 × 5)-bit slices**, $2^\ell$ of them
The state: an array of $5 \times 5 \times 2^\ell$ bits

- $5 \times 5$ lanes, each containing $2^\ell$ bits (1, 2, 4, 8, 16, 32 or 64)
- $(5 \times 5)$-bit slices, $2^\ell$ of them
Inside Keccak-\(f\)

The state: an array of \(5 \times 5 \times 2^\ell\) bits

- 5 \(\times\) 5 lanes, each containing \(2^\ell\) bits (1, 2, 4, 8, 16, 32 or 64)
- (5 \(\times\) 5)-bit slices, \(2^\ell\) of them
The nonlinear layer $\chi$

- “Flip bit if neighbors exhibit 01 pattern”
- Operates independently and in parallel on 5-bit rows
- Algebraic degree 2, inverse has degree 3
The mixing layer $\theta$

- Compute parity $c_{x,z}$ of each column
- Add to each cell parity of neighboring columns:

$$b_{x,y,z} = a_{x,y,z} \oplus c_{x-1,z} \oplus c_{x+1,z-1}$$
Difference propagation due to $\theta$

$$1 + \left(1 + y + y^2 + y^3 + y^4\right) \left(x + x^4z\right) \left(\text{mod} \left(1 + x^5, 1 + y^5, 1 + z^w\right)\right)$$
Difference propagation due to $\theta$ (kernel)

\[
1 + \left(1 + y + y^2 + y^3 + y^4\right) \left(x + x^4z\right) \\
\left(\text{mod} \left\langle 1 + x^5, 1 + y^5, 1 + z^w \right\rangle\right)
\]
Inverse of $\theta$ is dense

$$1 + \left(1 + y + y^2 + y^3 + y^4\right) Q,$$

with $Q = 1 + (1 + x + x^4 z)^{-1} \mod \left\langle 1 + x^5, 1 + z^w \right\rangle$
The inter-slice dispersion step $\rho$

- We need diffusion between the slices ...
- $\rho$: cyclic shifts of lanes
- Offsets cycle through all values below $2^\ell$
A first attempt at KECCAK-$f$

- **Round function:** $R = \iota \circ \rho \circ \theta \circ \chi$
- **Problem:** low-weight periodic trails by chaining:

- $\chi$: propagates unchanged with weight 4
- $\theta$: propagates unchanged, because all column parities are 0
- $\rho$: in general moves active bits to different slices ...
  ...but not always
The Matryoshka property

- Patterns in $Q'$ are $z$-periodic versions of patterns in $Q$
- Weight of trail $Q'$ is twice that of trail $Q$ (or $2^n$ times in general)
The intra-slice dispersion step $\pi$

We need to disturb horizontal/vertical alignment

\[ a_{x,y} \leftarrow a_{x',y'} \text{ with } \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 2 & 3 \end{pmatrix} \begin{pmatrix} x' \\ y' \end{pmatrix} \]
Inside KECCAK-f

$i$ to break symmetry

- XOR of round-dependent constant to lane in origin
- Without $i$, the round mapping would be symmetric
  - invariant to translation in the $z$-direction
  - susceptible to rotational cryptanalysis
- Without $i$, all rounds would be the same
  - susceptibility to slide attacks
  - defective cycle structure
- Without $i$, we get simple fixed points (000 and 111)
KECCAK-$f$ summary

- Round function:
  - $\theta$ for diffusion
  - $\rho$ for inter-slice dispersion
  - $\pi$ for disturbing horizontal/vertical alignment
  - $\chi$ for non-linearity
  - $\iota$ to break symmetry

$$R = \iota \circ \chi \circ \pi \circ \rho \circ \theta$$

- Number of rounds: $12 + 2\ell$
  - KECCAK-$f[25]$ has 12 rounds
  - KECCAK-$f[1600]$ has 24 rounds
Inside KECCAK-f

KECCAK-f in pseudo-code

KECCAK-f[b](A) {
  forall i in 0...nr-1
    A = Round[b](A, RC[i])
  return A
}

Round[b](A,RC) {
  \( \theta \) step
  C[x] = A[x,0] \ xor A[x,1] \ xor A[x,2] \ xor A[x,3] \ xor A[x,4], forall x in 0...4
  D[x] = C[x-1] \ xor rot(C[x+1],1), forall x in 0...4
  A[x,y] = A[x,y] \ xor D[x], forall (x,y) in (0...4,0...4)

  \( \rho \) and \( \pi \) steps
  B[y,2*x+3*y] = rot(A[x,y], r[x,y]), forall (x,y) in (0...4,0...4)

  \( \chi \) step
  A[x,y] = B[x,y] \ xor ((not B[x+1,y]) and B[x+2,y]), forall (x,y) in (0...4,0...4)

  \( \iota \) step
  A[0,0] = A[0,0] \ xor RC

  return A
}

https://keccak.team/keccak_specs_summary.html
Outline

1. KECCAK and SHA-3 functions
2. Inside KECCAK-f
3. Trail analysis
   - Goal
   - Propagation in KECCAK-f
   - Generating all 3-round trail cores up to some weight
   - Cases $|K|N$, $|N|K$ and $|N|N$
   - Case $|K|K$
   - Results
4. Alignment
   - What is alignment?
   - Alignment experiments in KECCAK-f
   - Relevance of alignment
5. KECCAKTOOLS
6. Conclusions
Outline

1. KECCAK and SHA-3 functions
2. Inside KECCAK-f
3. Trail analysis
   - Goal
   - Propagation in KECCAK-f
   - Generating all 3-round trail cores up to some weight
   - Cases $|K|N$, $|N|K$ and $|N|N$
   - Case $|K|K$
   - Results
4. Alignment
   - What is alignment?
   - Alignment experiments in KECCAK-f
   - Relevance of alignment
5. KECCAKTOOLS
6. Conclusions
Differential trails in iterated mappings

- **Trail**: sequence of differences
- **DP(Q)**: fraction of pairs that exhibit differences $q_i$

$$DP(Q) \approx DP_{0,1} \times DP_{1,2} \times DP_{2,3} \times DP_{3,4} \times DP_{4,5} \times DP_{5,6}$$

Diagram: 6 red squares labeled $R$ with arrows from $q_0$ to $q_6$. $Q$: $q_0, q_1, q_2, q_3, q_4, q_5, q_6$
Differential trails and weight

\[ w = - \log_2(DP) \]

\[ w(Q) = w_{0,1} + w_{1,2} + w_{2,3} + w_{3,4} + w_{4,5} + w_{5,6} \]

If independent rounds and \( w(Q) < b \): \( \#\text{pairs}(Q) \approx 2^{b-w(Q)} \)
Goal

- Security of KECCAK-$f[b]$ relies **not** on presumed hardness of
  - finding low-weight trails
  - finding pairs given a trail $Q$
- But on hardness to exploit trails with at most a few pairs

KECCAK-$f[b]$ design goal

**Absence** of trails with $w(Q) < b$

- Goal of this effort:
  - exhaustively generate trails up to some weight
  - to build assurance that there are no low-weight trails
  - inspired by similar efforts for Noekeon and MD6

<table>
<thead>
<tr>
<th></th>
<th>width</th>
<th>weight bound per round</th>
</tr>
</thead>
<tbody>
<tr>
<td>Noekeon</td>
<td>128</td>
<td>12.0 [Nessie, 2000]</td>
</tr>
<tr>
<td>MD6</td>
<td>4096</td>
<td>2.5 [Rivest et al., 2008][Heilman, 2011]</td>
</tr>
</tbody>
</table>
Outline

1. KECCAK and SHA-3 functions
2. Inside KECCAK-\(f\)
3. Trail analysis
   - Goal
   - Propagation in KECCAK-\(f\)
     - Generating all 3-round trail cores up to some weight
     - Cases \(|K|N|, |N|K| and \(|N|N|
     - Case \(|K|K|
   - Results
4. Alignment
   - What is alignment?
   - Alignment experiments in KECCAK-\(f\)
   - Relevance of alignment
5. KECCAKTOOLS
6. Conclusions
Propagating differences through $\chi$

- The propagation weight...
  - ... is determined by input difference only;
  - ... is the size of the affine base;
  - ... is the number of affine conditions.
Propagating linear masks through $\chi$

- The propagation weight...
  - ... is determined by output mask only;
  - ... is the size of the affine base.
Trails in KECCAK-\(f\)

Round: linear step \(\lambda = \pi \circ \rho \circ \theta\) and non-linear step \(\chi\)

- \(a_i\) fully determines \(b_i = \lambda(a_i)\)
- \(w(Q) = \sum_i w(b_{i-1} \xrightarrow{\chi} a_i)\)
Trails in KECCAK-f

Nonlinear step $\chi$ has algebraic degree 2

- for input $b_{i-1}$, the outputs $a_i$ form affine space $\mathcal{A}(b_{i-1})$
- dimension of $\mathcal{A}(b_{i-1})$ is $w(b_{i-1}, a_i) = w(b_{i-1})$
Trails in KECCAK-f

Trail weight fully determined by $b_i$

- We can ignore $a_4$: trail prefix
- We can ignore $a_0$
Trails in KECCAK-f

\[ w(Q) > b \] now has a simple meaning:

- \( w(Q) \): \# conditions on intermediate state bits
- \( b \): \# input bits
Given a trail, we can extend it:

- Forward: iterate $a_{r+1}$ over $A(b_r)$.
- Backward: iterate $b_{r-1}$ over all differences $\chi^{-1}$-compatible with $a_0 = \lambda^{-1}(b_0)$.

Tree search:
- Extension can be done recursively.
- Limited by total weight.

\[ w(Q) = w(b_0) + w(b_1) + w(b_2) + w(b_3) \]
Trail analysis

**Trail extension**

- Given a trail, we can extend it:
  - **forward:** iterate $a_{r+1}$ over $\mathcal{A}(b_r)$
  - **backward:** iterate $b_{r+1}$ over all differences $\chi^{-1}$-compatible with $a_0 = \lambda^{-1}(b_0)$

- **Tree search:**
  - extension can be done recursively
  - limited by total weight

---

![Diagram of trail extension and tree search]

**Diagram Legend:**
- $a_i$: Variables
- $\chi$: Contraction function
- $\lambda$: Expansion function
- $w(Q)$: Weight of the query
- $w(b_i)$: Weight of block $b_i$
- $\mathcal{A}(b_3)$: Output of $\mathcal{A}$ function for block $b_3$
Trail extension

- Given a trail, we can extend it:
  - forward: iterate $a_{r+1}$ over $A(b_r)$
  - backward: iterate $b_{-1}$ over all differences $\chi^{-1}$-compatible with $a_0 = \lambda^{-1}(b_0)$

- Tree search:
  - extension can be done recursively
  - limited by total weight
Given a trail, we can extend it:
- forward: iterate $a_{r+1}$ over $\mathcal{A}(b_r)$
- backward: iterate $b_{r-1}$ over all differences $\chi^{-1}$-compatible with $a_0 = \lambda^{-1}(b_0)$

Tree search:
- extension can be done recursively
- limited by total weight
Trail analysis

Propagation in Keccak-f

Trail core

- **Minimum reverse weight**: lower bound of weight given difference after $\chi$
  
  $$w^{\text{rev}}(a) \triangleq \min_{b : a \in A(b)} w(b)$$

- Can be used to lower bound of set of trails
- **Trail core**: set of trails with $b_1, b_2, \ldots$ in common
Trail core

- **Minimum reverse weight**: lower bound of weight given difference after $\chi$

\[
    w^{\text{rev}}(a) \triangleq \min_{b : a \in \mathcal{A}(b)} w(b)
\]

- Can be used to lower bound of set of trails
- **Trail core**: set of trails with $b_1, b_2, \ldots$ in common
Trail core

- **Minimum reverse weight**: lower bound of weight given difference after $\chi$

$$w^{\text{rev}}(a) \triangleq \min_{b : a \in \mathcal{A}(b)} w(b)$$

- Can be used to lower bound of set of trails

- **Trail core**: set of trails with $b_1, b_2, \ldots$ in common
Outline

1. KECCAK and SHA-3 functions
2. Inside KECCAK-
3. Trail analysis
   - Goal
   - Propagation in KECCAK-
   - Generating all 3-round trail cores up to some weight
     - Cases |K|N|, |N|K| and |N|N|
     - Case |K|K|
     - Results
4. Alignment
   - What is alignment?
   - Alignment experiments in KECCAK-
   - Relevance of alignment
5. KECCAKTOOLS
6. Conclusions
The parity kernel

- $\theta$ acts as the identity if parity is zero
- A state with parity zero is in the kernel (or in $|K|$)
- A state with parity non-zero is outside the kernel (or in $|N|$)
Covering the space of 3-round trail cores

\[ w(Q) = w_{\text{rev}}(a_1) + w(b_1) + w(b_2) \]

- Space split based on parity of \( a_i \)
- Four classes: \(|K|K|, |K|N|, |N|K| \) and \(|N|N|\)
Covering the space of 3-round trail cores

\[ w(Q) = w_{rev}(a_1) + w(b_1) \]

- Generating \((a_1, b_1)\)
- Extending forward by one round
Covering the space of 3-round trail cores

\[ w(Q) = w_{\text{rev}}(a_1) + w(b_1) \]

- Generating \((a_1, b_1)\)
- Extending forward by one round
Covering the space of 3-round trail cores

\[ w(Q) = w_{rev}(a_2) + w(b_2) \]

- Generating \((a_2, b_2)\)
- Extending backward by one round
COVERING THE SPACE OF 3-ROUND TRAIL CORES

\[ w(Q) = w_{\text{rev}}(a_2) + w(b_2) \]

- Generating \((a_2, b_2)\)
- Extending backward by one round
Outline

1. KECCAK and SHA-3 functions
2. Inside KECCAK-$f$
3. Trail analysis
   - Goal
   - Propagation in KECCAK-$f$
   - Generating all 3-round trail cores up to some weight
     - Cases $|K|N$, $|N|K$ and $|N|N$
     - Case $|K|K$
     - Results
4. Alignment
   - What is alignment?
   - Alignment experiments in KECCAK-$f$
   - Relevance of alignment
5. KECCAKTOOLS
6. Conclusions
Definition

Set $U$ of \textit{units} with a total order relation $\prec$

\section*{Tree}

- **Node**: subset of $U$, represented as a \textit{unit list}

  $$ a = (u_i)_{i=1,...,n} \quad u_1 \prec u_2 \prec \cdots \prec u_n $$

- **Children of a node $a$**:

  $$ a \cup \{u_{n+1}\} \quad \forall \ u_{n+1} : \ u_n \prec u_{n+1} $$

- **Root**: the empty set $a = \emptyset$
Example

- \( U = \{ (x, y, z) \} \), the coordinates in the KECCAK-f state
- \( \prec \) is \( [x, y, z] \) the lexicographical order in \( x \), then \( y \), then \( z \)
- State value \( a \) as unit list \( (x_i, y_i, z_i)_{i=1,\ldots,n} \) for all \( a_{x,y,z} = 1 \)
Bounding the cost

Goal: tree traversal up to given cost target $T$

Cost-related functions

- Cost function $w(a)$
- Subtree bounding function $L(a)$

\[ L(a) \leq w(a') \quad \text{for all descendants } a' \text{ of } a \]

$\Rightarrow$ Skip all the subtrees with $L(a) > T$
Example (continued)

- $U = \{(x, y, z)\}$, the coordinates in the KECCAK-f state
- $\prec$ is $[x, y, z]$ the lexicographical order in $x$, then $y$, then $z$
- State value $a$ as unit list $(x_i, y_i, z_i)_{i=1,...,n}$ for all $a_{x,y,z} = 1$
- Cost $w(a)$: Hamming, differential or restriction weight
- $L(a) = w(a)$ due to monotonicity in KECCAK-f
Properties of $\theta$

- **Affected** columns are complemented
- **Unaffected** columns are not changed
Properties of $\theta$ (continued)
Parity-bare state and orbitals

Lemma

*Each state can be decomposed in a unique way in a parity-bare state and a list of orbitals*
Lemma

Each state can be decomposed in a unique way in a parity-bare state and a list of orbitals
Generating parity-bare states

- **Root:** the empty state
- **Units:** column assignments \((x, z, \text{odd/affected, column value})\)
- **Bound:** weight minus potential loss due to new CAs

![Diagram showing the generation of parity-bare states](image-url)
Completing with orbitals

- Root: a parity-bare state
- Units: orbitals in unaffected columns \((x, y_1, y_2, z)\)
- Bound: weight of the trail itself
Outline

1 KECCAK and SHA-3 functions
2 Inside KECCAK-f
3 Trail analysis
   - Goal
   - Propagation in KECCAK-f
   - Generating all 3-round trail cores up to some weight
   - Cases $|K|N$, $|N|K$ and $|N|N$
   - Case $|K|K$
   - Results
4 Alignment
   - What is alignment?
   - Alignment experiments in KECCAK-f
   - Relevance of alignment
5 KECCAKTOOLS
6 Conclusions
Dealing with the kernel

- Problem: too many states in $|K|$
- Problematic case:

  $\lambda [a_1] + \lambda [b_1] + \lambda [b_2] = \frac{3T}{r}$

- Trail cores $(b, d)$ with $w^{rev}(a) + w(b) + w(d) \leq 3T/r$
  - $a = \lambda^{-1}(b)$ is in the kernel
  - intersection of $\mathcal{A}(b)$ and kernel is not empty
  - $b$ is tame
Dealing with the kernel

- Problem: too many states in $|K|$
- Problematic case:

  - Trail cores $(b, d)$ with $w^{\text{rev}}(a) + w(b) + w(d) \leq 3T/r$
  - $a = \lambda^{-1}(b)$ is in the kernel
  - Intersection of $\mathcal{A}(b)$ and kernel is not empty
  - $b$ is tame
Dealing with the kernel

- Problem: too many states in $|K|$
- Problematic case:

\[
\begin{array}{c}
\mathcal{A}(b) + w(b) + w(d) = \frac{3T}{r}
\end{array}
\]

- Trail cores $(b, d)$ with $w^{rev}(a) + w(b) + w(d) \leq 3T/r$
  - $a = \lambda^{-1}(b)$ is in the kernel
  - Intersection of $\mathcal{A}(b)$ and kernel is not empty
  - $b$ is tame
Third-order approach: tame states

- Condition that $a$ is in kernel
  - One-to-one mapping of active bit positions between $a$ and $b$
  - Translate conditions to $b$

- Tameness of slices of $b$
  - Empty slice is tame
  - Single-bit slice cannot be tame
  - Two-bit slice is tame iff bits are in same column (orbital)
  - More than 2 bits: knot

- **Chains**: sequences of active bits $p_i$ that:
  - Start and end in a knot
  - $p_{2i}$ and $p_{2i+1}$ are in same column in $a$
  - $p_{2i+1}$ and $p_{2i}$ are in same column in $b$
\(\rho, \pi\) and chains

Bit transpositions \(\rho\) and \(\pi\)

- \(\rho\): inter-slice

- \(\pi\): intra-slice

Example of a chain:
Third-order approach

- Representation of tame states:
  - set of chains between knots
  - plus some circular chains: vortices

- Efficiently iterating over tame states:
  - start from empty state
  - recursively add chains and vortices until predicted weight exceeds $3T/r$
  - if all knots are tame, valid output

- Full coverage guaranteed by
  - monotonous weight prediction function
  - well-defined order of chains
Outline

1 KECCAK and SHA-3 functions
2 Inside KECCAK-f
3 Trail analysis
   - Goal
   - Propagation in KECCAK-f
   - Generating all 3-round trail cores up to some weight
   - Cases $|K|N|$, $|N|K|$ and $|N|N$
   - Case $|K|K$
   - Results
4 Alignment
   - What is alignment?
   - Alignment experiments in KECCAK-f
   - Relevance of alignment
5 KECCAKTOOLS
6 Conclusions
## Summary of current results

- **All 3-round trail cores with weight \( \leq 45 \)**

- **No 6-round trail with weight \( \leq 91 \)**
## Summary of current results (cont’d)

<table>
<thead>
<tr>
<th>rounds</th>
<th>$b = 200$</th>
<th>$b = 400$</th>
<th>$b = 800$</th>
<th>$b = 1600$</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>3</td>
<td>20</td>
<td>24</td>
<td>32</td>
<td>32</td>
</tr>
<tr>
<td>4</td>
<td>46</td>
<td>[48,63]</td>
<td>[48,104]</td>
<td>[48,134]</td>
</tr>
<tr>
<td>5</td>
<td>[50,89]</td>
<td>[50,147]</td>
<td>[50,247]</td>
<td>[50,372]</td>
</tr>
<tr>
<td>6</td>
<td>[92,142]</td>
<td>[92,278]</td>
<td>[92,556]</td>
<td>[92,1112]</td>
</tr>
</tbody>
</table>

**Table:** Current bounds for the minimum weight of differential trails
KECCAK and SHA-3 functions

Inside KECCAK-\(f\)

Trail analysis

- Goal
- Propagation in KECCAK-\(f\)
- Generating all 3-round trail cores up to some weight
- Cases \(|K|N\), \(|N|K\) and \(|N|N\)
- Case \(|K|K|
- Results

Alignment

- What is alignment?
- Alignment experiments in KECCAK-\(f\)
- Relevance of alignment

KECCAKTOOLS

Conclusions
Outline

1 KECCAK and SHA-3 functions
2 Inside KECCAK-\(f\)
3 Trail analysis
   - Goal
   - Propagation in KECCAK-\(f\)
   - Generating all 3-round trail cores up to some weight
   - Cases \(|K|N|, |N|K| and |N|N|
   - Case \(|K|K|
   - Results
4 Alignment
   - What is alignment?
   - Alignment experiments in KECCAK-\(f\)
   - Relevance of alignment
5 KECCAKTOOLS
6 Conclusions
Difference propagation in Rijndael: strong alignment

- Propagation of differences:
  - MixColumns, ShiftRows and AddRoundKey: 1-to-1
  - SubBytes: 1-to-N
    - state with $x$ active bytes at input: $N = 126^x \approx 2^{7x}$

- Propagation of truncated differences (active/passive bytes)
  - SubBytes, ShiftRows and AddRoundKey: 1-to-1
  - MixColumns: 1-to-N
    - column with 1 active bytes at input: $N = 1$
    - column with 2 active bytes in input: $N = 5$
    - column with 3 active bytes in input: $N = 11$
    - column with 4 active bytes in input: $N = 15$
Alignment

- Property of round function
  - relative to partition of state in blocks

- **Strong alignment**
  - low uncertainty in propagation along block boundaries
  - e.g., RIJNDAEL strongly aligned on byte boundaries

- **Weak alignment**
  - high uncertainty in propagation along block boundaries
  - e.g., KECCAK weakly aligned on row boundaries...
Outline

1 KECCAK and SHA-3 functions
2 Inside KECCAK-f
3 Trail analysis
   - Goal
   - Propagation in KECCAK-f
   - Generating all 3-round trail cores up to some weight
   - Cases |K|N|, |N|K| and |N|N|
   - Case |K|K|
   - Results
4 Alignment
   - What is alignment?
   - Alignment experiments in KECCAK-f
   - Relevance of alignment
5 KECCAKTOOLS
6 Conclusions
Differential patterns
Attempt at quantifying alignment

For a given input activity pattern (specified in blocks)

- **$N$:** number of possible different output activity patterns
  - e.g., MixColumns 1 active byte: $N = 1$ (4 active bytes)
  - e.g., MixColumns 4 active bytes: $N = 15$ (1-4 active bytes)

- $h = - \sum_z \Pr(z|A) \log_2 \Pr(z|A)$: “entropy”
  - e.g., MixColumns 4 active bytes: $h \approx 0$ (most often 4)

- $\overline{w}$: average number of active blocks
  - e.g., MixColumns 4 active bytes: $\overline{w} \approx 4$ (most often 4)
Row activity: typical results

Output row-activity for single-row differences in row $y = 0$ at round input:

<table>
<thead>
<tr>
<th>$2^\ell$</th>
<th>$N$</th>
<th>$h$</th>
<th>$\overline{w}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0.00</td>
<td>5.00</td>
</tr>
<tr>
<td>2</td>
<td>11</td>
<td>1.97</td>
<td>9.35</td>
</tr>
<tr>
<td>4</td>
<td>26</td>
<td>4.60</td>
<td>15.54</td>
</tr>
<tr>
<td>8</td>
<td>31</td>
<td>4.95</td>
<td>19.22</td>
</tr>
<tr>
<td>16</td>
<td>31</td>
<td>4.95</td>
<td>23.09</td>
</tr>
<tr>
<td>32</td>
<td>31</td>
<td>4.95</td>
<td>25.29</td>
</tr>
<tr>
<td>64</td>
<td>31</td>
<td>4.95</td>
<td>25.54</td>
</tr>
</tbody>
</table>
Differential patterns (kernel)
Slice activity: the results

Output slice-activity for single-slice differences at round input:

<table>
<thead>
<tr>
<th>$2^\ell$</th>
<th>full single-slice set</th>
<th></th>
<th>in-kernel subset</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$N$</td>
<td>$h$</td>
<td>$\bar{w}$</td>
<td>$N$</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0.00</td>
<td>1.00</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>0.0002</td>
<td>1.99</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>15</td>
<td>0.04</td>
<td>3.99</td>
<td>15</td>
</tr>
<tr>
<td>8</td>
<td>247</td>
<td>0.98</td>
<td>7.85</td>
<td>247</td>
</tr>
<tr>
<td>16</td>
<td>50622</td>
<td>7.86</td>
<td>13.93</td>
<td>49999</td>
</tr>
<tr>
<td>32</td>
<td>5611775</td>
<td>19.66</td>
<td>20.25</td>
<td>1048575</td>
</tr>
<tr>
<td>64</td>
<td>12599295</td>
<td>22.87</td>
<td>22.50</td>
<td>1048575</td>
</tr>
</tbody>
</table>
Differential patterns (backwards)
Linear patterns
Linear patterns (backwards)
Linear patterns (backwards, kernel)
Outline

1. KECCAK and SHA-3 functions
2. Inside KECCAK-\(f\)
3. Trail analysis
   - Goal
   - Propagation in KECCAK-\(f\)
   - Generating all 3-round trail cores up to some weight
   - Cases \(|K|N|, |N|K|, and |N|N|
   - Case \(|K|K|
   - Results
4. Alignment
   - What is alignment?
   - Alignment experiments in KECCAK-\(f\)
   - Relevance of alignment
5. KECCAKTOOLS
6. Conclusions
Strong versus weak alignment

- Benefits of strong alignment
  - propagation analysis easy to describe and understand
  - strong trail bounds with simple proofs, e.g. 4R AES: 25 S-boxes
  - allows efficient table-lookup implementations

- Benefits of weak alignment
  - low clustering of trails
  - hard to build truncated differential trails
  - rebound attacks become very expensive

- impacts how attacks work: integral, impossible, zero-correlation, ...
Clustering of differential trails

\[ DP_{2R}(a, b) = \sum_{Q \in (a,b)} DP(Q) \approx \sum_{q'} DP_R(a, q') DP_R(q', b) \]

- **Necessary conditions** for a trail \( Q \) to contribute to \((a, b)\):
  - \( a \) and \( q \) have same S-box activity pattern
  - \( b' \) and \( L(q) \) have same S-box activity pattern

- **Relevance of alignment** of \( L \) along S-box boundaries:
  - **strong alignment**: \( L(q) \) has low variety in activity pattern
  - **weak alignment**: \( L(q) \) has wide variety in activity pattern

- Similar arguments apply for correlations and linear trails
Clustering of differential trails

\[ DP_{2R}(a, b) = \sum_{Q \in (a, b)} DP(Q) \approx \sum_{q'} DP_R(a, q') DP_R(q', b) \]

- Necessary conditions for a trail \( Q \) to contribute to \((a, b)\):
  - \( a \) and \( q \) have same S-box activity pattern
  - \( b' \) and \( L(q) \) have same S-box activity pattern

- Relevance of alignment of \( L \) along S-box boundaries:
  - strong alignment: \( L(q) \) has low variety in activity pattern
  - weak alignment: \( L(q) \) has wide variety in activity pattern

- Similar arguments apply for correlations and linear trails
Clustering of differential trails

\[ DP_{2R}(a, b) = \sum_{Q \in (a, b)} DP(Q) \approx \sum_{q'} DP_R(a, q') DP_R(q', b) \]

- Necessary conditions for a trail \( Q \) to contribute to \((a, b)\):
  - \( a \) and \( q \) have same S-box activity pattern
  - \( b' \) and \( L(q) \) have same S-box activity pattern

- Relevance of alignment of \( L \) along S-box boundaries:
  - strong alignment: \( L(q) \) has low variety in activity pattern
  - weak alignment: \( L(q) \) has wide variety in activity pattern

- Similar arguments apply for correlations and linear trails
Clustering of differential trails

\[ DP_{2R}(a, b) = \sum_{Q \in (a,b)} DP(Q) \approx \sum_{q} DP_S(a, q)DP_S(L(q), b') \]

- Necessary conditions for a trail \( Q \) to contribute to \( (a, b) \):
  - \( a \) and \( q \) have same S-box activity pattern
  - \( b' \) and \( L(q) \) have same S-box activity pattern

- Relevance of alignment of \( L \) along S-box boundaries:
  - strong alignment: \( L(q) \) has low variety in activity pattern
  - weak alignment: \( L(q) \) has wide variety in activity pattern

- Similar arguments apply for correlations and linear trails
Truncated differentials and rebound attacks

- Weak alignment means trails tend to diverge
  - low clustering of differential trails
  - hard to construct a truncated differential trail

- Open question for KECCAK
  - generalize truncation other than on block boundaries?

- Rebound attack typically requires truncated trails
  - it can also be done exploiting saturation
    - [Duc et al., Unaligned Rebound Attack: Appl. to KECCAK, FSE 2012]
  - still rather expensive
Outline

1 KECCAK and SHA-3 functions
2 Inside KECCAK-f
3 Trail analysis
   - Goal
   - Propagation in KECCAK-f
   - Generating all 3-round trail cores up to some weight
   - Cases $|K|N$, $|N|K$ and $|N|N$
   - Case $|K|K$
   - Results
4 Alignment
   - What is alignment?
   - Alignment experiments in KECCAK-f
   - Relevance of alignment
5 KECCAKTOOLS
6 Conclusions
What is KECCAKTOOLS?

A set of documented C++ classes to help analyze KECCAK-f

You said “documented”?  
- Documentation in Doxygen format  
- Various example routines in main.cpp  
- Sample of differential and linear trails

https://github.com/KeccakTeam/KeccakTools
KeccakTools

Functionality overview

- Seven permutations, from KECCAK-f[25] to KECCAK-f[1600]
  - Individual steps $\theta$, $\rho$, $\pi$, $\chi$ and $\iota$
  - And all the inverses $\iota^{-1} = \iota, \chi^{-1}, \pi^{-1}, \rho^{-1}$ and $\theta^{-1}$
- Sponge construction on any permutation
- Equations in GF(2) of rounds or steps
- Optimized C code (lane complementing, bit interleaving)
  - Macros currently in our optimized implementations
- Differential and linear cryptanalysis
The seven permutation army

**Instantiating and using KECCAK-\(f\)**

```cpp
KeccakF f(200);

KeccakF g(800, 123);
vector<LaneValue> state(25, 0);
g.forward(state);
g.inverseRound(state);
g.chi(state);
g.inverseTheta(state);
```

- Two ways to represent the state:
  - `vector<LaneValue>`, 25 lanes
  - `vector<SliceValue>`, from 1 to 64 slices
Variable naming convention

Lane naming convention

<table>
<thead>
<tr>
<th></th>
<th>x = 0</th>
<th>x = 1</th>
<th>x = 2</th>
<th>x = 3</th>
<th>x = 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>y = 0</td>
<td>ba</td>
<td>be</td>
<td>bi</td>
<td>bo</td>
<td>bu</td>
</tr>
<tr>
<td>y = 1</td>
<td>ga</td>
<td>ge</td>
<td>gi</td>
<td>go</td>
<td>gu</td>
</tr>
<tr>
<td>y = 2</td>
<td>ka</td>
<td>ke</td>
<td>ki</td>
<td>ko</td>
<td>ku</td>
</tr>
<tr>
<td>y = 3</td>
<td>ma</td>
<td>me</td>
<td>mi</td>
<td>mo</td>
<td>mu</td>
</tr>
<tr>
<td>y = 4</td>
<td>sa</td>
<td>se</td>
<td>si</td>
<td>so</td>
<td>su</td>
</tr>
</tbody>
</table>

- z coordinate as a suffix
  - E.g., bu21 is bit at $x = 4$, $y = 0$ and $z = 21$

- Alphabetical order = bit ordering at sponge level
  - Makes it easier to express concrete CICO problems (preimage, etc.)
Examples of generated equations

### Equations for $\theta$ and $\theta^{-1}$ in KECCAK-$f[100]$%

<table>
<thead>
<tr>
<th>Equation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$Oba_0 = Iba_0 + Ibe_3 + Ige_3 + Ike_3 + Ime_3 + Ise_3 + Ibu_0 + Igu_0 + Iku_0 + Imu_0 + Isu_0$</td>
<td></td>
</tr>
<tr>
<td>$Iba_0 = Oba_0 + Obi_0 + Ogi_0 + Oki_0 + Omi_0 + Osi_0 + Obo_3 + Ogo_3 + Oko_3 + Omo_3 + Oso_3 + Obi_3 + Ogi_3 + Oki_3 + Omi_3 + Osi_3 + Oba_2 + Oga_2 + Oka_2 + Oma_2 + Osa_2 + Obo_2 + Ogo_2 + Oko_2 + Omo_2 + Oso_2 + Obi_2 + Ogi_2 + Oki_2 + Omi_2 + Osi_2 + Obe_2 + Oge_2 + Oke_2 + Ome_2 + Ose_2 + Oba_1 + Oga_1 + Oka_1 + Oma_1 + Osa_1 + Obo_1 + Ogo_1 + Oko_1 + Omo_1 + Oso_1 + Obe_1 + Oge_1 + Oke_1 + Ome_1 + Ose_1</td>
<td></td>
</tr>
</tbody>
</table>
Examples of generated equations

Equations for $\chi$ and $\chi^{-1}$ in KECCAK-$f[100]$

\[
Obo2 = Ibo2 + (Ibu2 + 1)*Iba2
\]

\[
Ibo2 = Obo2 + (Oba2 + Obi2*(Obe2 + 1))*(Obu2 + 1)
\]

Equations for full round in KECCAK-$f[100]$

\[
Bgo3 = Ame2 + Abl1 + Agi1 + Aki1 + Ami1 + Asi1 + Aba2 + Aga2 + Aka2 + Ama2 + Asa2 + (Asi2 + Abo1 + Ago1 + Ako1 + Amo1 + Aso1 + Abe2 + Age2 + Ake2 + Ame2 + Ase2 + 1)*(Abo3 + Abu2 + Agu2 + Aku2 + Amu2 + Asu2 + Abi3 + Agi3 + Aki3 + Ami3 + Asi3)
\]
Differential and linear trails

- Trail: states \((a_0, a_1, \ldots)\) “before” \(\chi\)
- Propagation through “affine” direction:
  - Differential trails

\[
\begin{align*}
a_0 & \quad \lambda & \quad a_1 & \quad \lambda & \quad a_2 \\
\chi & \quad \theta, \rho, \pi & \quad \chi & \quad \theta, \rho, \pi & \quad \chi \quad \ldots
\end{align*}
\]

- Linear trails: forward propagation means backwards in time

\[
\begin{align*}
a_0 & \quad \lambda & \quad a_1 & \quad \lambda & \quad a_2 \\
\chi & \quad \pi^{-1}, \rho^{-1}, \Theta^T & \quad \chi & \quad \pi^{-1}, \rho^{-1}, \Theta^T & \quad \chi \quad \ldots
\end{align*}
\]
Differential or linear propagation context?

- **Class KeccakFDCLC**
  - Inherits from KeccakF
  - Computes all the propagation tables, both DC and LC

- **Class KeccakFPropagation**
  - Uses KeccakFDCLC
  - Specializes in either DC or LC
  - Allows uniform implementation of trail search

---

**Class instantiation for DC and LC**

```c++
KeccakFDCLC f(200);
KeccakFPropagation DC(f, KeccakFPropagation::DC);
...
KeccakFPropagation LC(f, KeccakFPropagation::LC);
...```

Displaying trails

- Examples of trails in package
- Text files, one trail per line

Checking and displaying trails

```cpp
KeccakFDCLC f(50);
KeccakFPropagation DC(f, KeccakFPropagation::DC);
fileName = DC.buildFileName("-trails");
Trail::produceHumanReadableFile(DC, fileName);
```
Forward propagation

Example: extending a linear trail

```cpp
Trail trail(inputFile);
const vector<SliceValue>& lastStateOfTrail = trail.states.back();
affineSpace = LC.buildStateBase(lastStateOfTrail);
for(SlicesAffineSpaceIterator i=affineSpace.getIterator(); [...] ) {
    Trail newTrail(trail);
    newTrail.append(*i, LC.getWeight(*i));
    newTrail.save(outputFile);
}
Trail::produceHumanReadableFile(LC, outputFileName);
```
Backward propagation

Example: extending a differential trail

```cpp
Trail trail(inputFile);
const vector<SliceValue>& firstStateOfTrail =
    trail.states.front();
DC.reverseLambda(firstStateOfTrail, [...]);
for(i=DC.getReverseStateIterator(\(\lambda^{-1}(firstStateOfTrail)\); [...]) {
    Trail newTrail(trail);
    newTrail.prepend(*i, DC.getWeight(*i));
    newTrail.save(outputFile);
}
Trail::produceHumanReadableFile(DC, outputFileName);
```
Differential trail equations

Constructing equations to follow a trail

KeccakFDCEquations f(50);
Trail trail(inputFile);
f.genDCEquations(outputFile, trail);

Example of generated equations

\[Ake1 = 0, Ako1 + 1 = 0, Ami1 = 0, [...]\]
\[Bba0 = Aba0 + [...]\]
\[Bbe0 + 1 = 0, Bbi0 + Bbo0 + 1 = 0, Bbu0 = 0, [...]\]
\[Cba0 = Bba0 + [...]\]
...

...
Outline

1. KECCAK and SHA-3 functions
2. Inside KECCAK-\( f \)
3. Trail analysis
   - Goal
   - Propagation in KECCAK-\( f \)
   - Generating all 3-round trail cores up to some weight
   - Cases \(|K|N|, |N|K| and |N|N|
   - Case \(|K|K|
   - Results
4. Alignment
   - What is alignment?
   - Alignment experiments in KECCAK-\( f \)
   - Relevance of alignment
5. KECCAKTOOLS
6. Conclusions
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
Conclusions

Status of Keccak

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

- Distinguishers
  - 7 rounds (practical time)
    
    [Huang et al., EUROCRYPT 2017]
  - 9 rounds ($2^{256}$ time, academic)
    
    [Dinur et al., EUROCRYPT 2015]

- Lots of third-party cryptanalysis available at:
  
  https://keccak.team/third_party.html
Any questions?

Thanks for your attention!

https://keccak.team/