Keccak,
More Than Just SHA3SUM

Guido Bertoni[1] Joan Daemen[1]
Michaël Peeters[2] Gilles Van Assche[1]

[1]STMicroelectronics

[2]NXP Semiconductors

FOSDEM 2013, Brussels, February 2-3, 2013

Outline

Outline

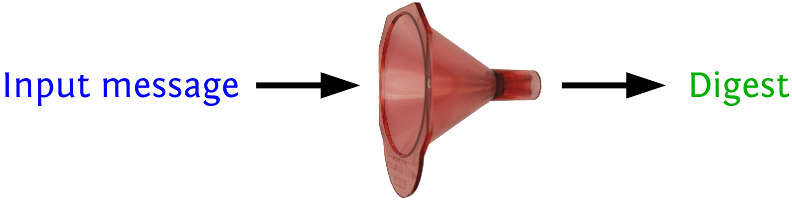
Let's talk about hash functions...

???

These are “hashes” of some sort, but they ain't **hash functions**...

Cryptographic hash functions

$$h : \{0,1\}^* \to \{0,1\}^n$$



- MD5: $n = 128$ (Ron Rivest, 1992)
- SHA-1: $n = 160$ (NSA, NIST, 1995)
- SHA-2: $n \in \{224, 256, 384, 512\}$ (NSA, NIST, 2001)

Why should you care?

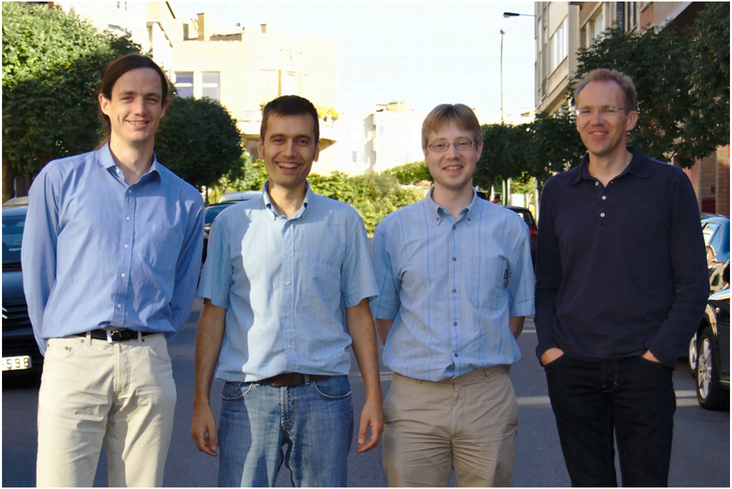
- You probably use them several times a day:
  - website authentication,
  - digital signature,
  - home banking,
  - secure internet connections,
  - software integrity,
  - version control software,
  - ...

Breaking news in crypto

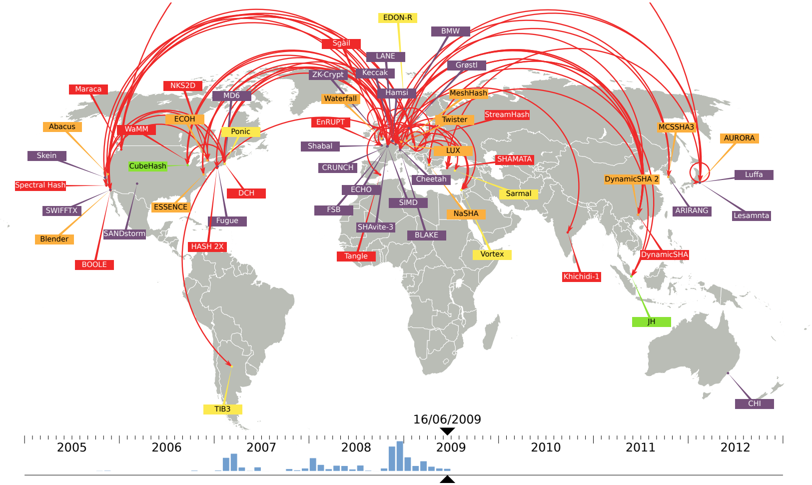
- 2004: SHA-0 broken (Joux et al.)
- 2004: MD5 broken (Wang et al.)
- 2005: practical attack on MD5 (Lenstra et al., and Klima)
- 2005: SHA-1 theoretically broken (Wang et al.)
- 2006: SHA-1 broken further (De Cannière and Rechberger)
- 2007: NIST calls for SHA-3

Who answered NIST's call?

KECCAK Team to the rescue!



The battlefield



[courtesy of Christophe De Cannière]

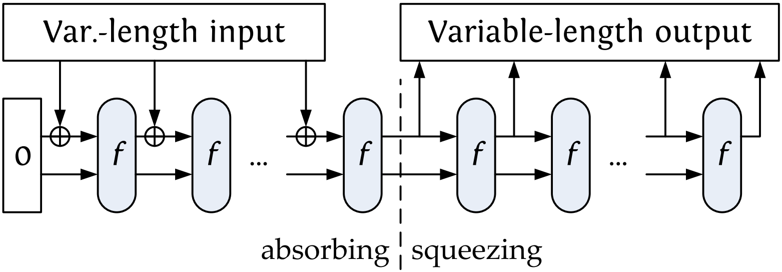
SHA-3 time schedule

- 2007: SHA-3 initial call
- 2008: submission deadline
- 2009: first SHA-3 conference
- 2010: second SHA-3 conference
- 2010: finalists are Blake, Grøstl, JH, KECCAK and Skein
- 2012: final SHA-3 conference
- Oct. 2, 2012: KECCAK wins!

Participants: $64 \to 51 \to 14 \to 5 \to 1$

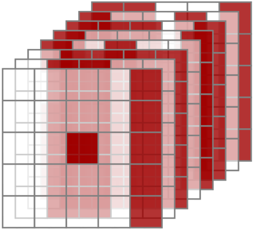
Outline

KECCAK, a sponge function



- Arbitrary *input* and *output* length
- More flexible than regular hash functions
- Parameters
  - r bits of *rate* (defines the speed)
  - c bits of *capacity* (defines the security level)
- Use the permutation KECCAK-f

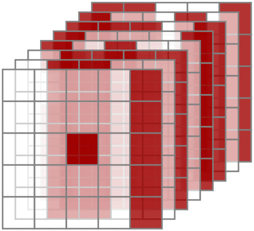
The seven permutation army



- (5×5) lanes
- up to 64-bit each

- 7 permutations:
  - 25, 50, 100, 200, 400, 800, 1600 bits
  - toy, lightweight, fastest
- repetition of a simple round function
  - operates on a 3D state
- like a block cipher but without a key

The seven permutation army



- (5×5) lanes
- up to 64-bit each

- First, choose your permutation ...
  - e.g. *width* $= 1600$
- ...then choose the *rate* and *capacity*
  - such that *rate* $+$ *capacity* $= 1600$
- Security-speed trade-offs using the same permutation:

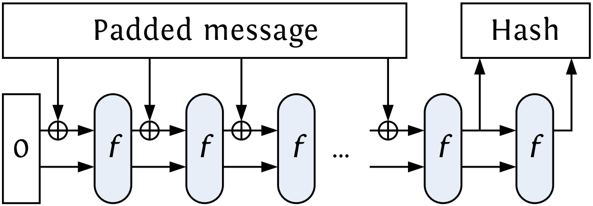
Rate	Capacity	Strength	Speed
1344	256	128	$\times 1.312$
1216	384	192	$\times 1.188$
1088	512	256	$\times 1.063$
1024	576	288	1.000

Outline

One primitive to rule them all

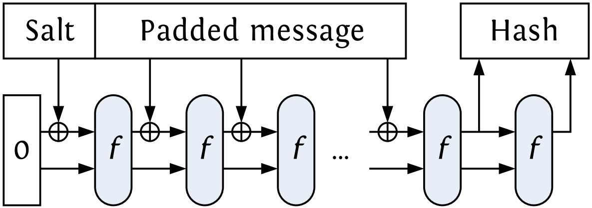
- Full range of cryptographic functions
  - hashing (regular, salted)
  - key derivation
  - message authentication
  - encryption
  - authenticated encryption
- ...in a simple way
  - simple & straightforward usage
  - easy to understand security claim
- ...and increasing diversity of standard portfolio
  - very different from SHA-1 and SHA-2
  - very different from AES and block cipher modes

Use KECCAK for regular hashing



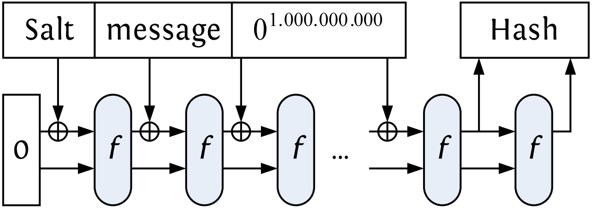
- Electronic signatures, message integrity (*GPG, X.509 …*)
- Data integrity (*shaxsum …*)
- Data identifier (*Git, Mercurial, online anti-virus, peer-2-peer …*)

Use KECCAK for salted hashing



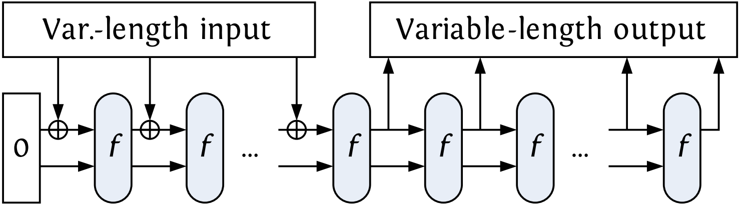
- Goal: defeat rainbow tables
- Web cookie
- Password storage and verification (*Kerberos*, /etc/shadow ...)

Use KECCAK for salted hashing



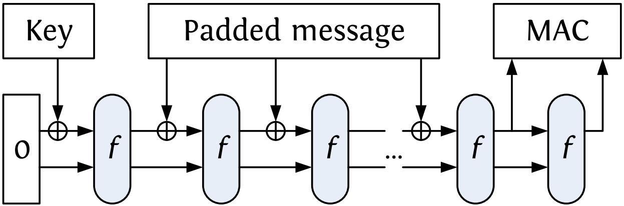
- Goal: defeat rainbow tables
- Web cookie
- Password storage and verification (*Kerberos*, /etc/shadow ...)
- ...Can be as slow as you like it!

Use KECCAK as a mask generation function



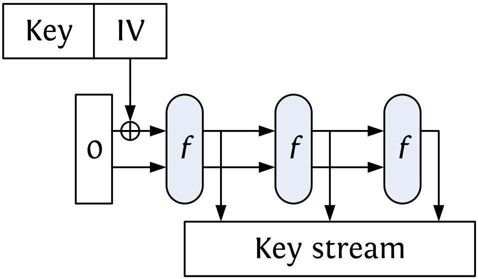
- Key derivation function in SSL, TLS
- Full-domain hashing in public key cryptography
  - electronic signatures RSA PSS [PKCS#1]
  - encryption RSA OAEP [PKCS#1]
  - key establishment RSA KEM [IEEE Std 1363a]

Use KECCAK for MACing



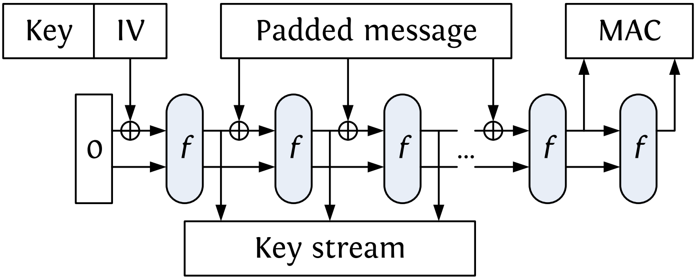
- As a message authentication code
- Simpler than HMAC [FIPS 198]
  - HMAC: special construction for MACing with SHA-1 and SHA-2
  - Required to plug a security hole in SHA-1 and SHA-2
  - No longer needed for KECCAK which is sound

Use KECCAK for (stream) encryption



- As a stream cipher

Single pass authenticated encryption



- Authentication and encryption in a single pass!
- Secure messaging (*SSL/TLS, SSH, IPSEC* ...)
- Same primitive KECCAK-f but in a (slightly) different mode
    - **Duplex** construction
    - Also for random generation with reseeding (/dev/urandom ...)

Tuning KECCAK to your own security requirements

Online tool available at http://keccak.noekeon.org/tune.html

Tune KECCAK to your requirements

The capacity parameter and chosen output length in KECCAK can be freely chosen. Their combination determines the attainable security and the capacity has an impact on performance. This page gives you the optimal capacity and output length values, given the classical hash function criteria.

Required collision resistance: 2^x	$x=$	Enter a value
Required (second) preimage resistance: 2^y	$y=$	Enter a value

Please specify your requirements...

Tuning KECCAK to your own security requirements

Online tool available at http://keccak.noekeon.org/tune.html

Tune KECCAK to your requirements

The capacity parameter and chosen output length in KECCAK can be freely chosen. Their combination determines the attainable security and the capacity has an impact on performance. This page gives you the optimal capacity and output length values, given the classical hash function criteria.

Required collision resistance: 2^x	$x=$ 128
Required (second) preimage resistance: 2^y	$y=$ 128

The optimal choice of parameters is:

KECCAK[r=1344,c=256] with a least **256 bits** of output.

Speed

For long messages, this function is **31.3% faster than** KECCAK[] (KECCAK with the default parameters). On the reference processor proposed by NIST, long messages should take about **9.6 cycles/byte**.

Tuning KECCAK to your own security requirements

Online tool available at http://keccak.noekeon.org/tune.html

Tune KECCAK to your requirements

The capacity parameter and chosen output length in KECCAK can be freely chosen. Their combination determines the attainable security and the capacity has an impact on performance. This page gives you the optimal capacity and output length values, given the classical hash function criteria.

Required collision resistance: 2^x	$x=$	128
Required (second) preimage resistance: 2^y	$y=$	256

The optimal choice of parameters is:

KECCAK[r=1088,c=512] with a least **256 bits** of output.

Speed

For long messages, this function is **6.25% faster than** KECCAK[] (KECCAK with the default parameters). On the reference processor proposed by NIST, long messages should take about **11.9 cycles/byte**.

Tuning KECCAK to your own security requirements

Online tool available at http://keccak.noekeon.org/tune.html

Security claim

In line with our hermetic sponge strategy, we make a flat sponge claim with c=256 bits of capacity: for any output length, we claim this KECCAK sponge function resists any attack up to $\mathbf{2^{128}}$ operations (each of complexity equivalent to one call to KECCAK-f), unless easier on a random oracle. For 256 bits of output specifically, this translates into the following claimed security level:

	Claimed security level
Collision resistance	2^{128}
(Second) preimage resistance	2^{128}

Addendum: how big is 2^{128}?

If an attacker has access to one billion computers, each performing one billion evaluations of KECCAK-f per second, it would take about **1.1×10^{13} years (770 times the estimated age of the universe)** to evaluate the permutation 2^{128} times.

Tuning KECCAK to your own security requirements

Online tool available at http://keccak.noekeon.org/tune.html

Security claim

In line with our hermetic sponge strategy, we make a flat sponge claim with c=512 bits of capacity: for any output length, we claim this KECCAK sponge function resists any attack up to $\mathbf{2^{256}}$ operations (each of complexity equivalent to one call to KECCAK-f), unless easier on a random oracle. For 256 bits of output specifically, this translates into the following claimed security level:

	Claimed security level
Collision resistance	2^{128}
(Second) preimage resistance	2^{256}

Addendum: how big is 2^{256}?

If an attacker has access to one billion computers, each performing one billion evaluations of KECCAK-f per second, it would take about **3.7×10^{51} years (2.6×10^{41} times the estimated age of the universe)** to evaluate the permutation 2^{256} times.

Considering an irreversible computer working at 2.735°K (the average temperature of the universe), Landauer's principle implies that it cannot consume less than 2.62×10^{-23} joule every time a bit is changed. (Computers actually consume much more than that.) Just counting from 1 to 2^{256} would take at least **3×10^{54} joules (the total energy output of the Sun during 2.5×10^{20} years)**.

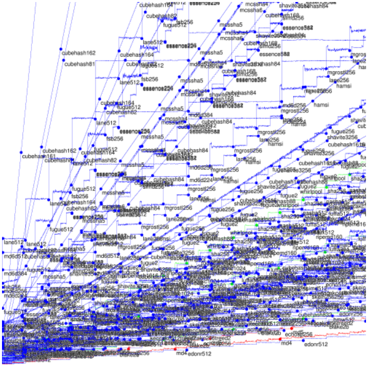
Outline

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) {
  θ 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)

  ρ and π steps
  B[y,2*x+3*y] = rot(A[x,y], r[x,y]),                        forall (x,y) in (0…4,0…4)

  χ 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)

  ι step
  A[0,0] = A[0,0] xor RC

  return A
}
```

http://keccak.noekeon.org/specs_summary.html

Performance in software



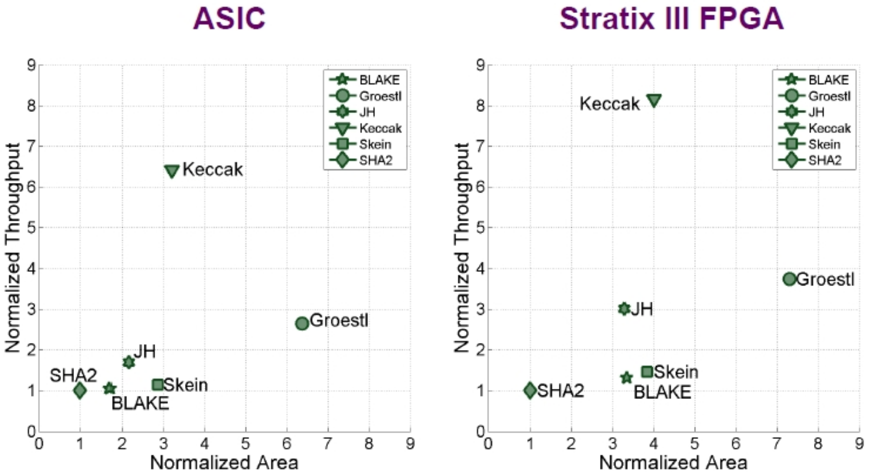
- Faster than SHA-2 on all modern PC
- KECCAKTREE faster than MD5 on some platforms

C/b	Algo		Strength
4.79	keccakc256treed2		128
4.98	md5	**broken!**	64
5.89	keccakc512treed2		256
6.09	sha1	**broken!**	80
8.25	keccakc256		128
10.02	keccakc512		256
13.73	sha512		256
21.66	sha256		128

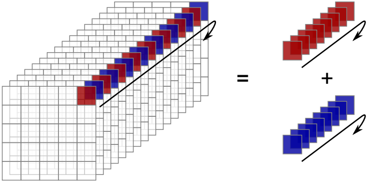
[eBASH, hydra-6, http://bench.cr.yp.to/]

Efficient and flexible in hardware

From Kris Gaj's presentation at SHA-3, Washington 2012:



Bit interleaving



- Ex.: map 64-bit lane to 32-bit words
  - ρ seems the critical step
  - Even bits in one word
    Odd bits in a second word
  - $\mathrm{ROT}_{64} \leftrightarrow 2 \times \mathrm{ROT}_{32}$
- Can be generalized
  - to 16- and 8-bit words
- Can be combined
  - with lane/slice-wise architectures
  - with most other techniques
- No mismatch CPU words *vs.* security level

[KECCAK impl. overview, Section 2.1]

Outline

SHA-3, an open contest

- Open submissions, as required by NIST:
  - Public algorithm details
  - Open-source reference and optimized implementations
  - No patents
- Open cryptanalysis
- Open benchmarks [eBASH] [XBX]

KeccakTools

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

- To encourage cryptanalysis *(we use it too!)*
- To help verify our claims [Keccak Team, FSE 2012]
- And also to generate optimized code

Prizes for best cryptanalysis results

- **Four** cryptanalysis prizes awarded!
- 25 bottles of Belgian trappist beer
  [CICO problem & cube testers, Aumasson and Khovratovich]
- Bialetti coffee machine
  [zero-sum, Aumasson and Meier]
- Lambic-based beers and book
  [zero-sum, Boura and Canteaut]
- Belgian finest chocolates
  [second preimage, Bernstein]

Crunchy Crypto Collision and Preimage Contest

- Goal:
  - Motivate 3rd-party cryptanalysis
  - Give an instant view on current state-of-the-art
- Scope: 1 to 12 rounds, including smaller instances
  - KECCAK$[r = 40, c = 160]$, $\leftarrow$ *no challenge broken yet!*
  - KECCAK$[r = 240, c = 160]$,
  - KECCAK$[r = 640, c = 160]$, and
  - KECCAK$[r = 1440, c = 160]$
- Results so far:
  - Preimages found for 1-2 rounds
  - Collisions found for 1-4 rounds

http://keccak.noekeon.org/crunchy_contest.html

Hex-Hot-Ticks

- Contest for stimulating developers in using ”exotic” platforms
- Winners:
  - KECCAK on a NVIDIA GPU using CUDA [Gerhard Hoffmann]
  - KECCAKTREE on a NVIDIA GPU also using CUDA [Guillaume Sevestre]

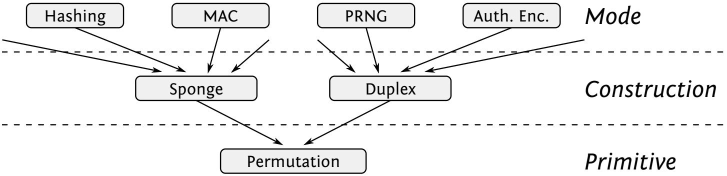
Implementations

- Reference implementations
  - Focused on readability
  - In C, C++ and Python
- Optimized implementations
  - For 8-bit, 32-bit (bit interleaving), 64-bit platforms + 128-bit SIMD
  - In C or in assembly (x86, ARM, AVR)
  - In-place for reduced memory footprint
  - KECCAKTOOLS to generate optimized code

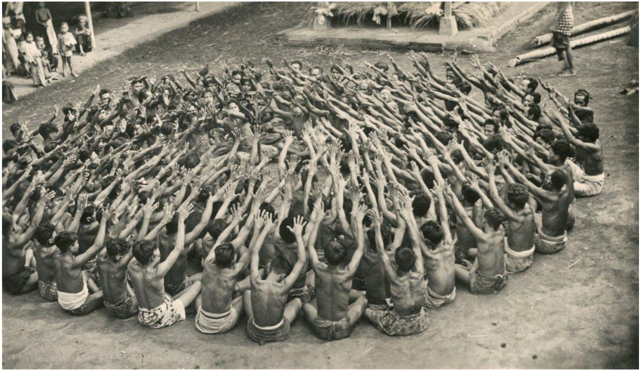
Available at http://keccak.noekeon.org/files.html

Do you want to help?

- You can
  - make static / dynamic libraries,
  - optimize current implementations,
  - write a new implementation in your favorite language.
- Implementation-oriented doc. [Keccak implementation overview]
- Please respect the Sponge / Duplex interfaces
  - API guideline to be published soon
- SHA-3 not standardized yet!



Questions?



More information on
http://sponge.noekeon.org/
http://keccak.noekeon.org/

Credits

- Creative Commons Attribution
  - http://en.wikipedia.org/wiki/File:Japanese_Secret_Puzzle_Box.jpg, from Nipaylah.
- Creative Commons Attribution-Share Alike
  - http://en.wikipedia.org/wiki/File:Cannabis_leaf_2.svg, from Christopher Thomas.
  - http://commons.wikimedia.org/wiki/File:Powder_Funnel.jpg, from Krakuspm.
  - http://www.flickr.com/photos/pfvunderground/6914321238/ from Underground PFV Uitgeverij (via http://photopin.com).
- Creative Commons Attribution-NonCommercial-NoDerivs
  - http://www.flickr.com/photos/stevie_gill/3950697539/, from stevie.gill (via http://photopin.com).
  - http://www.flickr.com/photos/marcelgermain/2078076913/, from MarcelGermain (via http://photopin.com).
- SHA-3 battlefield picture courtesy of Christophe De Cannière