SHA-3 and permutation-based cryptography

Joan Daemen[1]

Joint work with
Guido Bertoni[1], Michaël Peeters[2] and Gilles Van Assche[1]

[1]STMicroelectronics [2]NXP Semiconductors

Crypto summer school
Šibenik, Croatia, June 1-6, 2014

Outline

1 Prologue

2 The sponge construction

3 Keccak and SHA-3

4 Sponge modes of use

5 Block cipher vs permutation

6 Variations on sponge

Outline

1 Prologue

2 The sponge construction

3 Keccak and SHA-3

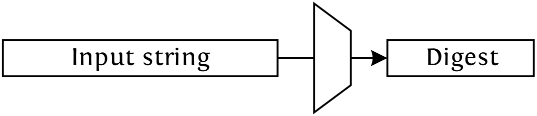
4 Sponge modes of use

5 Block cipher vs permutation

6 Variations on sponge

Cryptographic hash functions

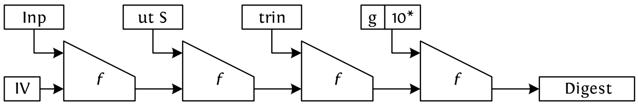
- Function h from $\mathbf{Z}_2^*$ to $\mathbf{Z}_2^n$
- Typical values for n: 128, 160, 256, 512



- Pre-image resistant: it shall take 2^n effort to
  - given y, find x such that $h(x) = y$
- 2nd pre-image resistance: it shall take 2^n effort to
  - given M and $h(M)$, find another M' with $h(M') = h(M)$
- collision resistance: it shall take $2^{n/2}$ effort to
  - find $x_1 \neq x_2$ such that $h(x_1) = h(x_2)$

Classical way to build hash functions

- Mode of use of a compression function:
  - Fixed-input-length compression function
  - Merkle-Damgård iterating mode



- Property-preserving paradigm
  - hash function *inherits* properties of compression function
  - …actually block cipher with feed-forward (Davies-Meyer)
- Compression function built on arithmetic-rotation-XOR: ARX
- Instances: MD5, SHA-1, SHA-2 (224, 256, 384, 512) …

Outline

1 Prologue

2 The sponge construction

3 KECCAK and SHA-3

4 Sponge modes of use

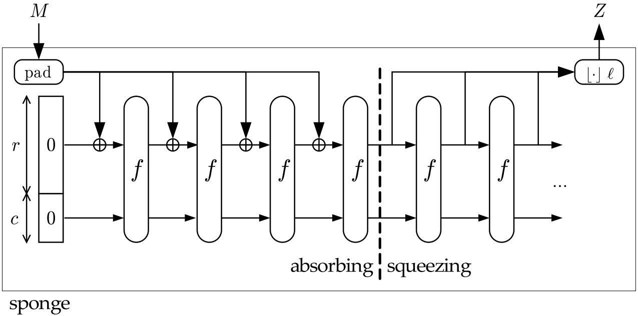
5 Block cipher vs permutation

6 Variations on sponge

Sponge origin: RadioGatún

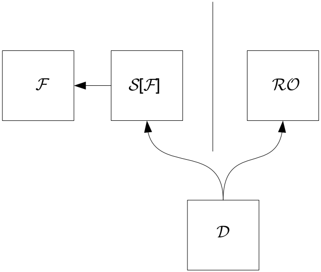
- Initiative to design hash/stream function (late 2005)
  - rumours about NIST call for hash functions
  - forming of Keccak Team
  - starting point: fixing Panama [Daemen, Clapp, FSE 1998]
- RadioGatún [Keccak team, NIST 2nd hash workshop 2006]
  - more conservative than Panama
  - arbitrary output length
  - **expressing security claim** for arbitrary output length function
- Sponge functions [Keccak team, Ecrypt hash, 2007]
  - random sponge instead of random oracle as security goal
  - sponge construction calling random permutation
  - ... *closest thing to a random oracle with a finite state* ...

The sponge construction



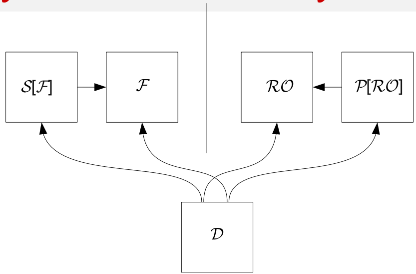
- Generalizes hash function: *extendable output function* (XOF)
- Calls a b-bit permutation f, with $b = r + c$
  - r bits of *rate*
  - c bits of *capacity* (security parameter)
- Property-preservation no longer applies

Generic security: indistinguishability



- Success probability of distinguishing between:
  - ideal function: a monolithic random oracle $\mathcal{RO}$
  - construction $\mathcal{S}[\mathcal{F}]$ calling an random permutation $\mathcal{F}$
- Adversary $\mathcal{D}$ sends queries (M, ℓ) according to algorithm
- Express $\Pr(\text{success}|\mathcal{D})$ as a function of total cost of queries N
- Problem: in real world, $\mathcal{F}$ is available to adversary

Generic security: indifferentiability [Maurer et al. (2004)]



- Applied to hash functions in [Coron et al. (2005)]
  - distinguishing mode-of-use from ideal function ($\mathcal{RO}$)
  - covers adversary with access to permutation $\mathcal{F}$ at left
  - additional interface, covered by a *simulator* at right
- Methodology:
  - build $\mathcal{P}$ that makes left/right distinguishing difficult
  - prove bound for advantage given this simulator $\mathcal{P}$
  - $\mathcal{P}$ may query $\mathcal{RO}$ for acting $\mathcal{S}$-consistently: $\mathcal{P}[\mathcal{RO}]$

Generic security of the sponge construction

Concept of *advantage*:

$$\Pr(\text{success}|\mathcal{D}) = \frac{1}{2} + \frac{1}{2}\text{Adv}(\mathcal{D})$$

Theorem (Bound on the $\mathcal{RO}$-differentiating advantage of sponge)

$$A \leq \frac{N^2}{2^{c+1}}$$

A: differentiating advantage of random sponge from random oracle
N: total data complexity
c: capacity

[Keccak team, Eurocrypt 2008]

Implications of the bound

- Let $\mathcal{D}$: n-bit output pre-image attack. Success probability:
  - for random oracle: $P_{\text{pre}}(\mathcal{D}|\mathcal{RO}) = q2^{-n}$
  - for random sponge: $P_{\text{pre}}(\mathcal{D}|\mathcal{S}[\mathcal{F}]) = ?$
- A distinguisher $\mathcal{D}$ with $A = P_{\text{pre}}(\mathcal{D}|\mathcal{S}[\mathcal{F}]) - P_{\text{pre}}(\mathcal{D}|\mathcal{RO})$
  - do pre-image attack
  - if success, conclude random sponge and $\mathcal{RO}$ otherwise
- But we have a proven bound $A \leq \frac{N^2}{2^{c+1}}$, so

$$P_{\text{pre}}(\mathcal{D}|\mathcal{S}[\mathcal{F}]) \leq P_{\text{pre}}(\mathcal{D}|\mathcal{RO}) + \frac{N^2}{2^{c+1}}$$

- Can be generalized to any attack
- Note that A is independent of output length n

Implications of the bound (cont'd)

- Informally, random sponge is like random oracle for $N < 2^{c/2}$
- Security strength for output length n:
  - collision-resistance: $\min(c/2, n/2)$
  - first pre-image resistance: $\min(c/2, n)$
  - second pre-image resistance: $\min(c/2, n)$
- Proof assumes f is a random permutation
  - provably secure against generic attacks
  - ...but not against attacks that exploit specific properties of f
- No security against multi-stage adversaries

A design approach

Hermetic sponge strategy

- Instantiate a sponge function
- Claim a security level of $2^{c/2}$

Remaining task

Design permutation f without exploitable properties

How to build a strong permutation

- Like a block cipher
  - sequence of identical rounds
  - round consists of sequence of simple step mappings
  - many approaches exist, e.g., wide-trail
- ...but without need for
  - key schedule
  - efficient inverse
  - width b that is power of two

Outline

1 Prologue

2 The sponge construction

3 Keccak and SHA-3

4 Sponge modes of use

5 Block cipher vs permutation

6 Variations on sponge

KECCAK$[r, c]$

- Sponge function using the permutation KECCAK-f
  - 7 permutations: $b \in \{25, 50, 100, 200, 400, 800, 1600\}$
    … from toy over lightweight to high-speed …
- SHA-3 instance: $r = 1088$ and $c = 512$
  - permutation width: 1600
  - security strength 256: post-quantum sufficient
- Lightweight instance: $r = 40$ and $c = 160$
  - permutation width: 200
  - security strength 80: what SHA-1 should have offered

See [The KECCAK reference] for more details

KECCAK[r, c]

- Sponge function using the permutation KECCAK-f
  - 7 permutations: $b \in \{25, 50, 100, 200, 400, 800, 1600\}$
    ... from toy over lightweight to high-speed ...
- SHA-3 instance: $r = 1088$ and $c = 512$
  - permutation width: 1600
  - security strength 256: post-quantum sufficient
- Lightweight instance: $r = 40$ and $c = 160$
  - permutation width: 200
  - security strength 80: what SHA-1 should have offered

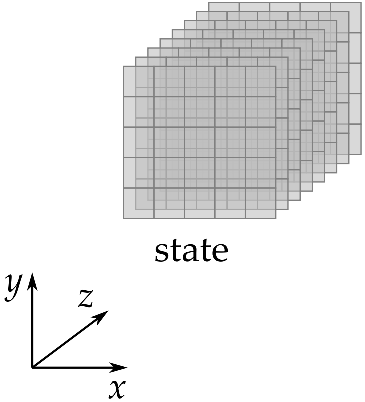
See [The KECCAK reference] for more details

KECCAK[r, c]

- Sponge function using the permutation KECCAK-f
  - 7 permutations: $b \in \{25, 50, 100, 200, 400, 800, 1600\}$
    ... from toy over lightweight to high-speed ...
- SHA-3 instance: $r = 1088$ and $c = 512$
  - permutation width: 1600
  - security strength 256: post-quantum sufficient
- Lightweight instance: $r = 40$ and $c = 160$
  - permutation width: 200
  - security strength 80: what SHA-1 should have offered

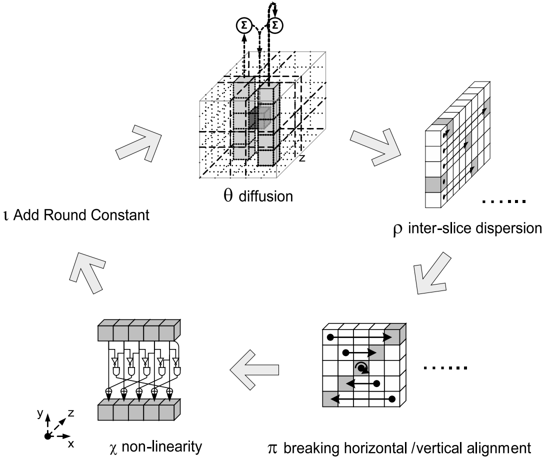
See [The KECCAK reference] for more details

The 3-dimensional KECCAK-f state



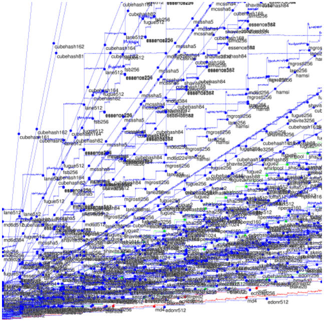
- 5×5 lanes, each containing 2^ℓ bits (1, 2, 4, 8, 16, 32 or 64)
- (5×5)-bit slices, 2^ℓ of them

The step mappings of the KECCAK-f round function



Keywords: wide-trail, lightweight, symmetry, bit-oriented, margin

Performance in software



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, hydra6 (AMD Bulldozer), http://bench.cr.yp.to/]

- KECCAKTREE: parallel tree hashing
- Speedup thanks to SIMD instructions

SHA-3 requirements and Keccak final submission

Output length	Collision resistance	Pre-image resistance	Keccak instance	Rate	Relative perf.
$n = 224$	112	224	Keccak$[c = 448]$	1152	$\times 1.125$
$n = 256$	128	256	Keccak$[c = 512]$	1088	$\times 1.063$
$n = 384$	192	384	Keccak$[c = 768]$	832	$\div 1.231$
$n = 512$	256	512	Keccak$[c = 1024]$	576	$\div 1.778$
free	up to 288	up to 288	Keccak$[c = 576]$	1024	1

Output-length oriented approach

- These instances address the SHA-3 requirements, but:
  - security strength levels outside of [NIST SP 800-57] range
  - performance penalty for high-capacity instances!

What we proposed to NIST

Security strength	Capacity	Output length	Coll. res.	Pre. res.	Relative perf.	SHA-3 instance
$s \geq 112$	$c = 256$	$n = 224$	112	128	×1.312	SHA3-224
$s \geq 128$	$c = 256$	$n = 256$	128	128	×1.312	SHA3-256
$s \geq 192$	$c = 512$	$n = 384$	192	256	×1.063	SHA3-384
$s \geq 256$	$c = 512$	$n = 512$	256	256	×1.063	SHA3-512
up to 128	$c = 256$	free	up to 128		×1.312	SHAKE256
up to 256	$c = 512$	free	up to 256		×1.063	SHAKE512

Security strength oriented approach consistent with [NIST SP 800-57]

- Underlying security strength levels reduced to 128 and 256
- Strengths 384 and 512: not needed anymore

What came out after the controversy

Security strength	Capacity	Output length	Coll. res.	Pre. res.	Relative perf.	SHA-3 instance
$s \geq 224$	$c = 448$	$n = 224$	112	224	$\times 1.125$	SHA3-224
$s \geq 256$	$c = 512$	$n = 256$	128	256	$\times 1.063$	SHA3-256
$s \geq 384$	$c = 768$	$n = 384$	192	384	$\div 1.231$	SHA3-384
$s \geq 512$	$c = 1024$	$n = 512$	256	512	$\div 1.778$	SHA3-512
up to 128	$c = 256$	free	up to 128		$\times 1.312$	SHAKE128
up to 256	$c = 512$	free	up to 256		$\times 1.063$	SHAKE256

Back to square 1 for drop-ins and security-strength oriented for SHAKEs

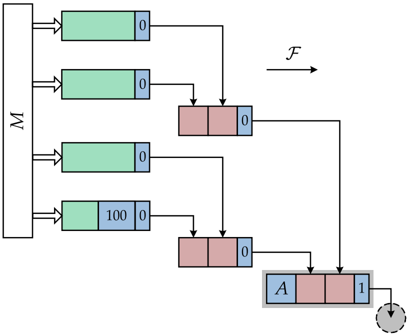
- Animated public discussion on *reducing security strength*
- Unfortunate timing: Snowden revelations on NSA, weaknesses in Dual EC DRBG

FIPS 202 draft

- Published Friday, April 4, 2014
- Four drop-in replacements identical to 3rd round submission
- Two *extendable output functions* (XOF)
- Tree-hashing ready: Sakura coding [Keccak team, ePrint 2013/231]

XOF	SHA-2 drop-in replacements
$\text{Keccak}[c=256](M\|\|11\|\|11)$	
	$\lfloor \text{Keccak}[c=448](M\|\|01) \rfloor_{224}$
$\text{Keccak}[c=512](M\|\|11\|\|11)$	$\lfloor \text{Keccak}[c=512](M\|\|01) \rfloor_{256}$
	$\lfloor \text{Keccak}[c=768](M\|\|01) \rfloor_{384}$
	$\lfloor \text{Keccak}[c=1024](M\|\|01) \rfloor_{512}$
SHAKE128 and **SHAKE256**	**SHA3-224** to **SHA3-512**

SAKURA and tree hashing



- Sound tree hashing is relatively easy to achieve [Keccak team, ePrint 2009/210 — **last updated 2014**]
- Defining tree hash modes addressing all future use cases is hard
- Defining future-proof tree hash coding is easy: SAKURA
- $M||11$ actually denotes *a single-node tree*

Outline

1 Prologue

2 The sponge construction

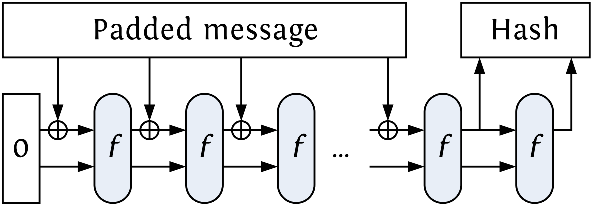
3 KECCAK and SHA-3

4 Sponge modes of use

5 Block cipher vs permutation

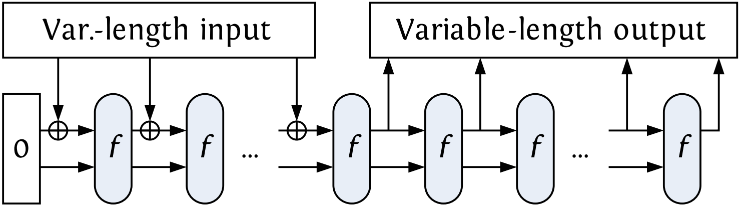
6 Variations on sponge

Regular hashing



- Salting: just pre- or append salt to message

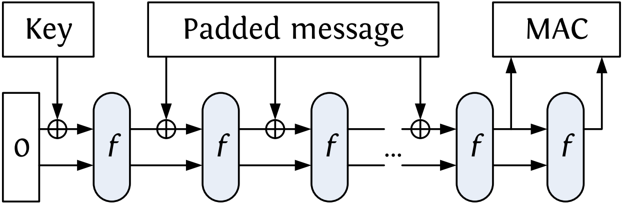
Mask generation function



output length often dictated by application ...
... rather than by security strength level

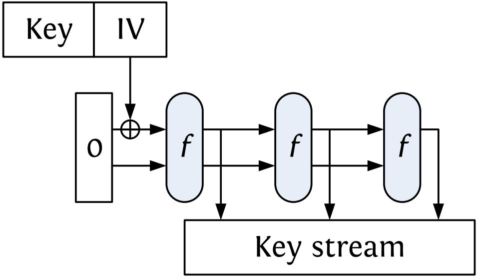
- Key derivation function in SSL, TLS
- Full-domain hashing in public key cryptography
  - electronic signatures RSASSA-PSS [PKCS#1]
  - encryption RSAES-OAEP [PKCS#1]
  - key encapsulation methods (KEM)

Message authentication codes



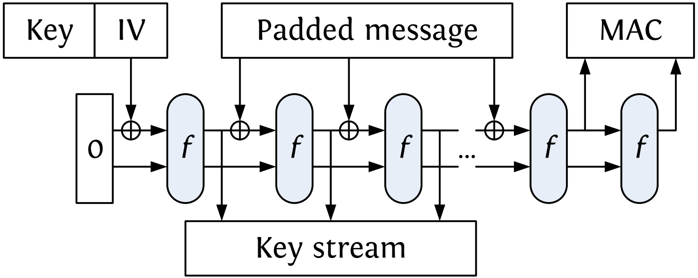
- Simpler than HMAC [FIPS 198]
  - Required for SHA-1, SHA-2 due to length extension property
  - HMAC is **no longer needed** for sponge!

Stream encryption



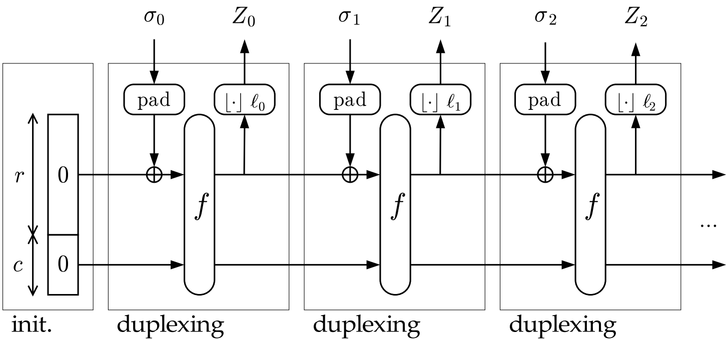
- As a stream cipher
  - Long output stream per IV: similar to OFB mode
  - Short output stream per IV: similar to counter mode

Single pass authenticated encryption



- Authentication and encryption in a single pass!
- Secure messaging (*SSL/TLS, SSH, IPSEC …*)
- This is no longer sponge

The duplex construction

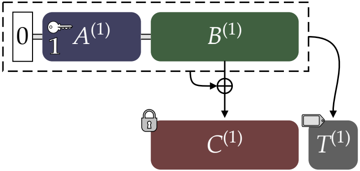


- Generic security equivalent to Sponge [Keccak team, SAC 2011]
- Applications include:
  - Authenticated encryption: spongeWrap, duplexWrap
  - Reseedable pseudorandom sequence generator

DuplexWrap layer

DuplexWrap (used in our CAESAR candidate Keyak)

- nonce-based authenticated encryption mode;
- works on sequences of header-body pairs.



$A^{(1)}$ must be unique and secret, e.g.,

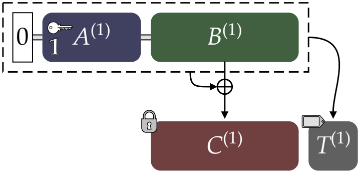
- $A^{(1)}$ contains a session key used only once;
- $A^{(1)}$ contains a key and a nonce.

In general: $A^{(1)} = \text{key}||\text{nonce}||\text{associated data}$.

DuplexWrap layer

DuplexWrap (used in our CAESAR candidate Keyak)

- nonce-based authenticated encryption mode;
- works on sequences of header-body pairs.



$A^{(1)}$ must be unique and secret, e.g.,

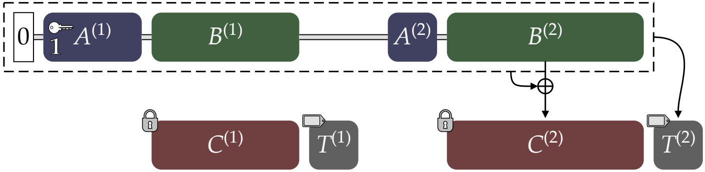
- $A^{(1)}$ contains a session key used only once;
- $A^{(1)}$ contains a key and a nonce.

In general: $A^{(1)} = \text{key}||\text{nonce}||\text{associated data}$.

DuplexWrap layer

DuplexWrap (used in our CAESAR candidate Keyak)

- nonce-based authenticated encryption mode;
- works on sequences of header-body pairs.



$A^{(1)}$ must be unique and secret, e.g.,

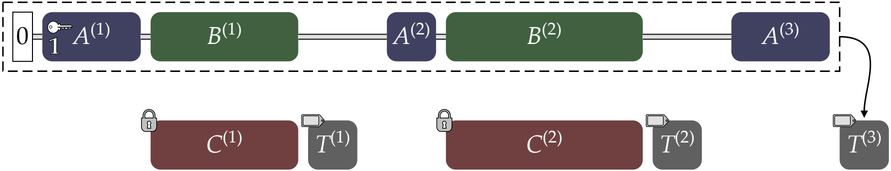
- $A^{(1)}$ contains a session key used only once;
- $A^{(1)}$ contains a key and a nonce.

In general: $A^{(1)} = \text{key}||\text{nonce}||\text{associated data}$.

DuplexWrap layer

DuplexWrap (used in our CAESAR candidate Keyak)

- nonce-based authenticated encryption mode;
- works on sequences of header-body pairs.



$A^{(1)}$ must be unique and secret, e.g.,

- $A^{(1)}$ contains a session key used only once;
- $A^{(1)}$ contains a key and a nonce.

In general: $A^{(1)} = \text{key}||\text{nonce}||\text{associated data}$.

Outline

1 Prologue

2 The sponge construction

3 KECCAK and SHA-3

4 Sponge modes of use

5 Block cipher vs permutation

6 Variations on sponge

Block cipher modes of use

- Hashing (in MDX and SHA-X) and its modes HMAC, MGF1, ...
- Block encryption: ECB, CBC, ...
- Stream encryption:
  - synchronous: counter mode, OFB, ...
  - self-synchronizing: CFB
- MAC computation: CBC-MAC, C-MAC, ...
- Authenticated encryption: OCB, GCM, CCM ...

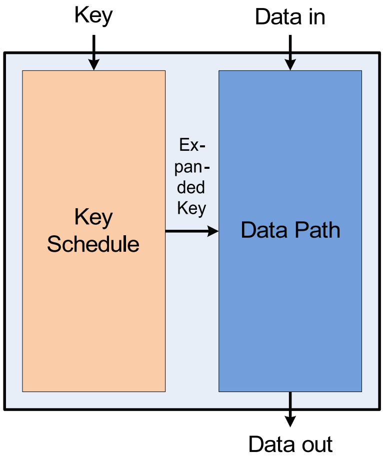
Etc.

Block cipher modes of use requiring the inverse

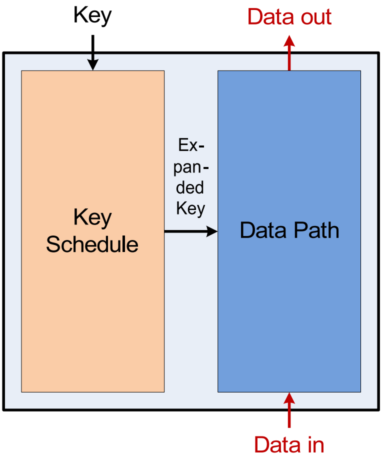
- Hashing (in MDX and SHA-X) and its modes HMAC, MGF1, ...
- Block encryption: ECB, CBC, ...
- Stream encryption:
  - synchronous: counter mode, OFB, ...
  - self-synchronizing: CFB
- MAC computation: CBC-MAC, C-MAC, ...
- Authenticated encryption: OCB, GCM, CCM ...

In many cases you don't need the inverse

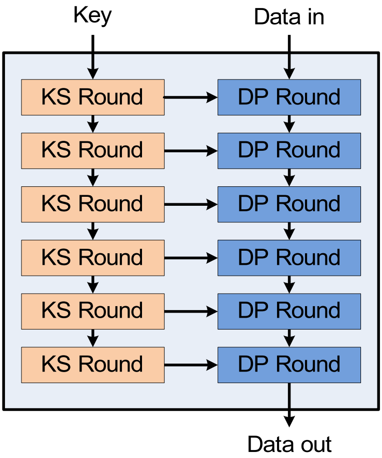
Structure of a block cipher



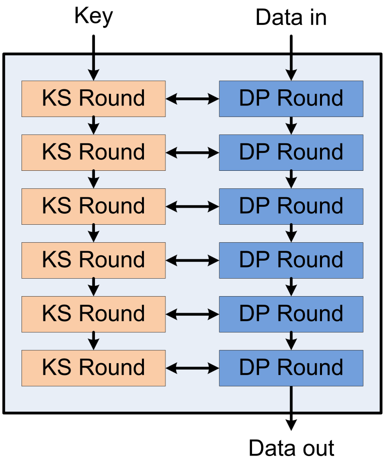
Structure of a block cipher (inverse operation)



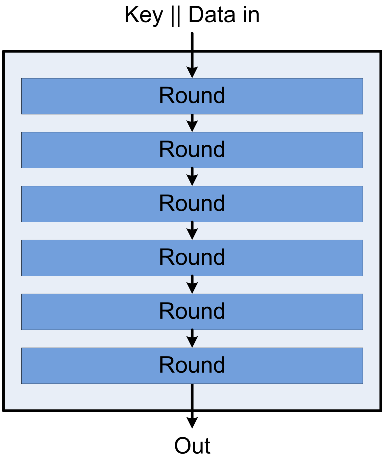
From block cipher to permutation



From block cipher to permutation



From block cipher to permutation



Block cipher vs permutation in keyed modes

- Permutation can replace block cipher mode if inverse not needed
- Dedicated permutation modes on top of sponge and duplex
- Block cipher with n-bit block and k bit key
  - processes n bits per call
  - security strength against key retrieval $\leq 2^k$
  - computation cost: data path + key schedule
  - key schedule can be factored out
- Permutation with width b
  - processes r bits per call
  - security strength against key retrieval $\geq 2^{c/2}$
  - computation cost: full permutation
- For equal dimensions $b = n + k$: block cipher clearly more efficient

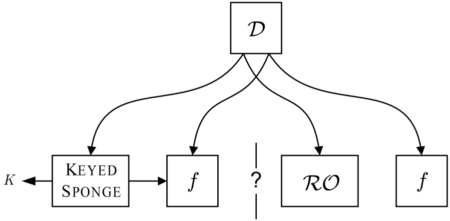
Block cipher vs permutation: a closer look

- Equal dimensions $b = n + k$
- Complexity
  - N (time): number of key guesses
  - M (data): number of input/output blocks
- Permutation: $\mathcal{RO}$-differentiating bound $N + M \geq 2^{c/2}$

Key retrieval security:

	block cipher		permutation	
Case	N	M	required c	efficiency loss
single target	2^{k-1}	≥ 1	$2k$	k/n
2^a targets	2^{k-a}	$\geq 2^a$	$2(k-a)$	$(k-2a)/n$
limit $a = k/2$	$2^{k/2}$	$\geq 2^{k/2}$	k	0

Security of keyed sponge functions



- New work building on [KECCAK team, On the security of the keyed sponge]
- Security strength against distinguishing: $\min(2^{c-(a+3)}, 2^k)$
- With 2^a the *multiplicity* of the data and $1 \leq 2^a \leq M$
  - $2^a \approx M$: limit case of very permissive mode and active adversary
  - $2^a = 1$: e.g., stream encryption with $M \leq 2^{r/2}$
- Allows reducing capacity, thereby reducing efficiency loss

Outline

1 Prologue

2 The sponge construction

3 Keccak and SHA-3

4 Sponge modes of use

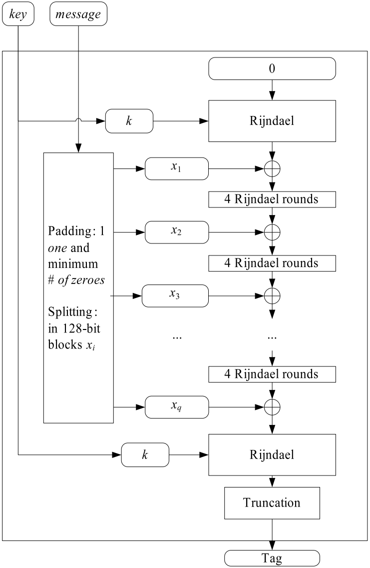
5 Block cipher vs permutation

6 Variations on sponge

Variations on sponge and duplex

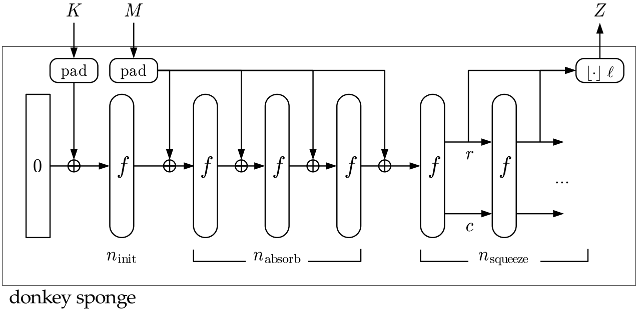
- Sponge and duplex are wide-spectrum
- Variants can be made
  - generalization: Parazoa [Andreeva, Mennink, Preneel 2011]
  - optimized for specific purposes
  - giving up *hermetic sponge* approach
- Ideas:
  - different rates during squeezing and absorbing
  - block encryption: requiring inverse permutation when decrypting
  - put the key in initial state rather than absorb it
  - ...
  - see CAESAR (and SHA-3) candidates for examples
- Two examples
  - donkeySponge for fast MACs
  - monkeyDuplex for authenticated encryption on small platforms

MAC: take a look at Pelican [Daemen, Rijmen, 2005]



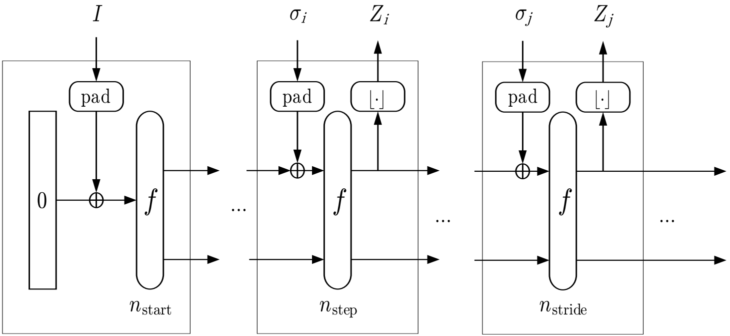
- Block cipher based MAC
  - based on Rijndael (AES)
  - permutation-based absorbing
- Speed: for long messages:
  - 4 rounds per 128 bits
  - 2.5 times faster than AES
- Security rationale
  - key recovery: block cipher
  - secret state recovery:
    - block cipher at the end
    - hardness of inner collisions
    - relies on low MDP of AES 4R
- security claims with $2^a \leq 2^{60}$
  - unbroken as yet

The donkeySponge MAC construction



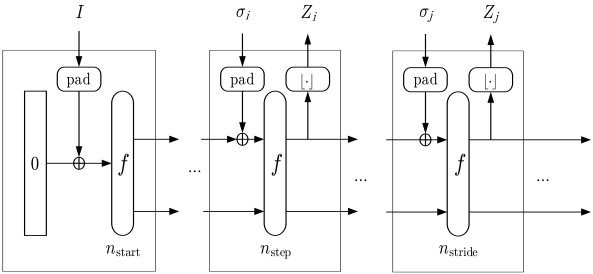
- Usage of full state width b during absorbing
- Reduced number of rounds during absorbing
- Truncated permutation instead of final block cipher
- KECCAK-$f[1600]$-based: over 5 times faster than SHAKE256

The monkeyDuplex construction



- For (authenticated) encryption
- Initialization: key, nonce in *I* followed by strong permutation
- strongly reduced number of rounds in step calls
- Used in KETJE (CAESAR) with KECCAK-$f[200]$ and KECCAK-$f[400]$

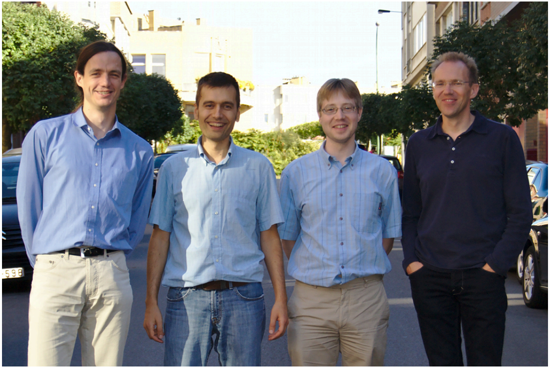
monkeyDuplex rationale



- Initialization
  - decorrelates states for different nonces
  - is assumed to rule out differential attacks
- Remaining attacks:
  - state reconstruction: number of rounds to span is $\left\lceil \frac{b-r}{r} \right\rceil n_{\text{step}}$
  - tag forgery: number of rounds to span is n_{stride}
- Price paid: in case of nonce re-use all bets are off

Conclusion

Permutation-based cryptography is here to stay!



http://sponge.noekeon.org/
http://keccak.noekeon.org/