On dec(k) functions

Gilles Van Assche[1]

[1]STMicroelectronics

Indocrypt, New Delhi, December 2018

Based on joint work with Guido Bertoni, Joan Daemen, Seth Hoffert, Michaël Peeters, Ronny Van Keer

About the title

dec(k) functions?

$k \leftarrow k - 1$?!?

This talk is about **dec** and **deck** functions:

- doubly extendable cryptographic functions
- doubly extendable cryptographic keyed functions

Outline

Outline

Extendable output

Keyed + extendable output = **key stream generator**

Unkeyed + extendable output = **extendable output function (XOF)**

[Ray Perlner, SHA-3 workshop 2014] [NIST FIPS 202, 2015]

Building a XOF from a hash function

Hash function $h(x)$ becomes XOF $H(x)$, with:

$$H(x) = h(x||1) \; || \; h(x||2) \; || \; \ldots \; || \; h(x||i) \; || \; \ldots$$

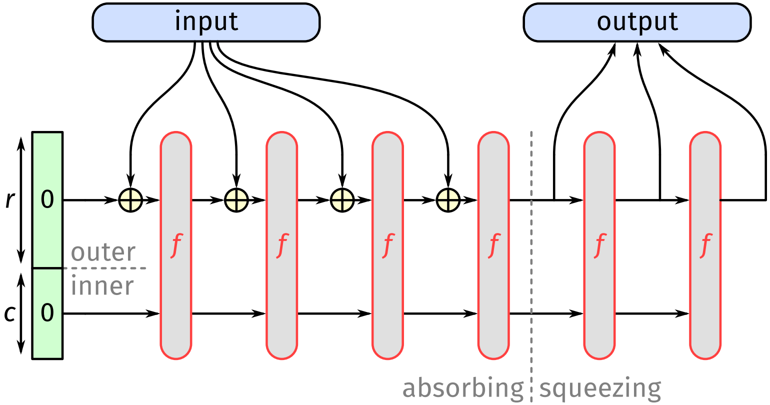
[MGF1, PKCS #1, RSA Labs 1998]

Cost per output byte depends on:

- ratio between input/output block sizes
- padding and output transformations

Typically higher than for input with traditional hash functions

Building a XOF with a permutation



Cost per output byte = cost per input byte [KT, Eurocrypt 2008]

Definition of a dec function

A dec function H

$$Z = 0^n + H\left(X^{(m-1)} \circ \cdots \circ X^{(0)}\right) \ll q$$

- Input: sequence of strings $X^{(m-1)} \circ \cdots \circ X^{(0)}$
- Output: potentially infinite output
  - **hash of the input**
  - taking n bits starting from offset q

Definition of a dec function

A dec function H

$$Z = 0^n + H\left(X^{(m-1)} \circ \cdots \circ X^{(0)}\right) \ll q$$

Efficient incrementality

- Extendable input
  1. Compute $H(X)$
  2. Compute $H(Y \circ X)$: cost independent of X
- Extendable output
  1. Request n_1 bits from offset 0
  2. Request n_2 bits from offset n_1: cost independent of n_1

Definition of a dec function

A dec function H

$$Z = 0^n + H\left(X^{(m-1)} \circ \cdots \circ X^{(0)}\right) \lll q$$

Efficient incrementality

- Extendable input
- Extendable output

Example: **TupleHashXOF** [NIST SP 800-185]

Definition of a deck function

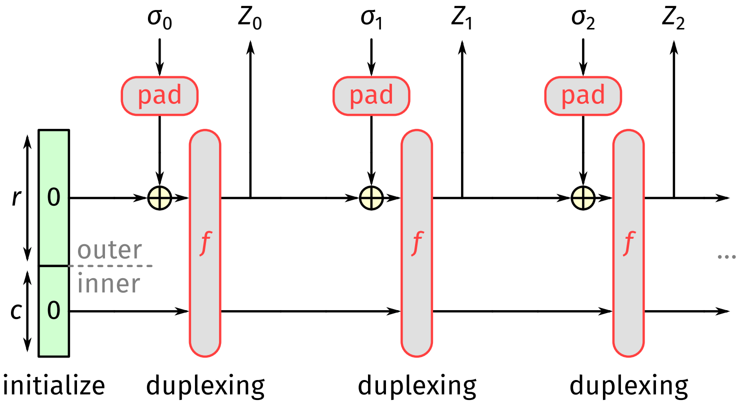
A deck function F_K

$$Z = 0^n + F_K\left(X^{(m-1)} \circ \cdots \circ X^{(0)}\right) \ll q$$

- Input: key K and sequence of strings $X^{(m-1)} \circ \cdots \circ X^{(0)}$
- Output: potentially infinite output
  - **pseudo-random function of the input**
  - taking n bits starting from offset q
- Same efficient incrementality as dec (with unchanged key)

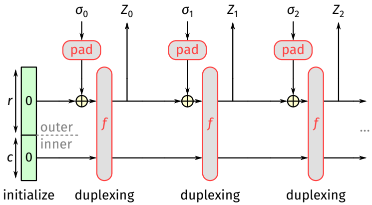
Outline

Duplex object as a dec function?



[KT, SAC 2011]

Duplex object as a dec function?



Is this a dec function?

- Input is a sequence of strings $\sigma_2 \circ \sigma_1 \circ \sigma_0$
  - Extendable input, but $|\sigma_i| \leq r - 2$
- Output is equivalent to [Sponge duplexing lemma]

$$Z_i = \text{sponge}(\sigma_0||\text{pad}||\sigma_1||\ldots||\text{pad}||\sigma_i)$$

  - Extendable output, but $|Z_i| \leq r$

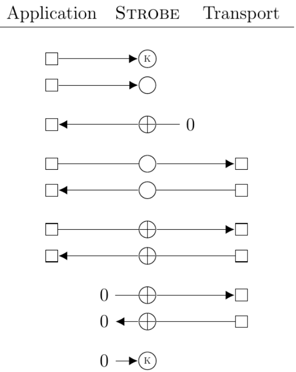
STROBE

- Layer above the duplex construction
  - compliant with **cSHAKE** [NIST SP 800-185]
- Safe and easy syntax, to achieve, e.g.,
  - secure channels
  - hashing of protocol transcripts
  - signatures over a complete session
- Very compact implementation
- Mechanism to prevent side-channel attacks

[Hamburg, RWC 2017]

Operations and data flow in STROBE

Abbr.	Operation	Flags
KEY	Secret key	AC
AD	Associated data	A
PRF	Hash / PRF	IAC
CLR	Send cleartext data	$A\ \ T$
recv-CLR	Receive cleartext data	$IA\ \ T$
ENC	Encrypt	ACT
recv-ENC	Decrypt	$IACT$
MAC	Compute MAC	CT
recv-ENC	Verify MAC	$I\ \ CT$
RATCHET	Rekey to prevent rollback	C



Legend: □ Send/recv ○ Absorb into sponge ⊕ Xor with cipher Ⓚ Roll key

figure courtesy of Mike Hamburg

Example: protocol

KEY(shared key K)	$X \leftarrow K$
AD`[nonce]`(seq. number i)	$X \leftarrow (i) \circ$ "`nonce`" $\circ X$
AD`[auth-data]`($IP_1 \| IP_2$)	$X \leftarrow (IP_1 \| IP_2) \circ$ "`auth-data`" $\circ X$
send_ENC("GET file")	ciphertext $\leftarrow$ "GET file" $+ H(X)$ $X \leftarrow$ "GET file" $\circ X$
send_MAC(128 bits)	MAC $\leftarrow 0^{128} + H(X)$
recv_ENC(ciphertext buffer)	plaintext $\leftarrow$ ciphertext $+ H(X)$ $X \leftarrow$ plaintext $\circ X$
recv_MAC(128 bits)	check that MAC $= 0^{128} + H(X)$

Example: protocol

KEY(DH shared secret K_{AB})	$X \leftarrow K_{AB}$
AD[nonce](seq. number i)	$X \leftarrow (i) \circ$ "nonce" $\circ X$
AD[auth-data]($IP_1 \| IP_2$)	$X \leftarrow (IP_1 \| IP_2) \circ$ "auth-data" $\circ X$
send_ENC("GET file")	ciphertext $\leftarrow$ "GET file" $+ H(X)$ $X \leftarrow$ "GET file" $\circ X$
send_MAC(128 bits)	MAC $\leftarrow 0^{128} + H(X)$
recv_ENC(ciphertext buffer)	plaintext $\leftarrow$ ciphertext $+ H(X)$ $X \leftarrow$ plaintext $\circ X$
recv_MAC(128 bits)	check that MAC $= 0^{128} + H(X)$

Example: signing a protocol transcript (1/3)

Ephemeral key generation

AD[name](“Ed448_STROBE”)	$X \leftarrow$ “Ed448_STROBE” $\circ$ [name]
AD[client](“command”)	$X \leftarrow$ “command” $\circ$ [client] $\circ X$
AD[server](“response”)	$X \leftarrow$ “response” $\circ$ [server] $\circ X$
KEY[sym-key](K)	$X \leftarrow K \circ$ [sym-key] $\circ X$
$r \leftarrow$ **PRF**[r](114 bytes)	$r \leftarrow H(X)$

Example: signing a protocol transcript (2/3)

Signature generation

AD[name](“Ed448_STROBE”)	$X \leftarrow$ “Ed448_STROBE” $\circ$ [name]
AD[client](“command”)	$X \leftarrow$ “command” $\circ$ [client] $\circ X$
AD[server](“response”)	$X \leftarrow$ “response” $\circ$ [server] $\circ X$
AD[pub-key](A)	$X \leftarrow A \circ$ [pub-key] $\circ X$
CLR($R = rB$)	$X \leftarrow R \circ X$
$h \leftarrow$ **HASH**(114 bytes)	$h \leftarrow H(X)$
CLR($s = (r + ah) \bmod \ell$)	$X \leftarrow s \circ X$

Example: signing a protocol transcript (3/3)

Signature verification

AD[name](“Ed448_STROBE”)	$X \leftarrow$ “Ed448_STROBE” $\circ$ [name]
AD[client](“command”)	$X \leftarrow$ “command” $\circ$ [client] $\circ X$
AD[server](“response”)	$X \leftarrow$ “response” $\circ$ [server] $\circ X$
AD[pub-key](A)	$X \leftarrow A \circ$ [pub-key] $\circ X$
$R \leftarrow$ **CLR**(57 bytes)	$X \leftarrow R \circ X$
$h \leftarrow$ **HASH**(114 bytes)	$h \leftarrow H(X)$
$s \leftarrow$ **CLR**(57 bytes)	check that $sB = R + hA$

STROBE + Noise = Disco

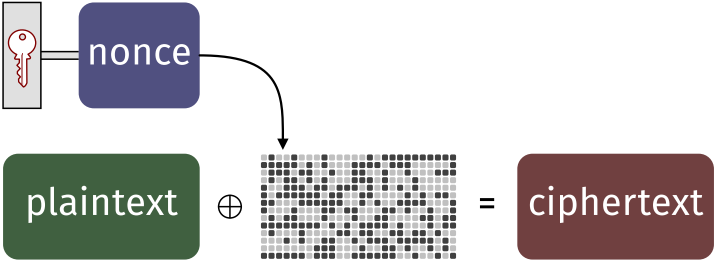
See `http://www.discocrypto.com/`

[Wong, Black Hat Europe 2017]

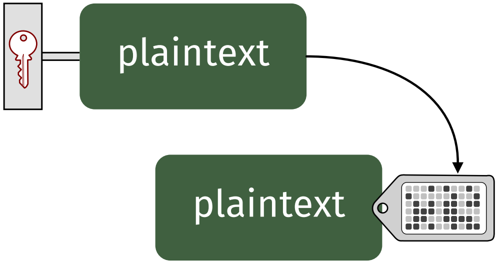
[Perrin, RWC 2018]

Outline

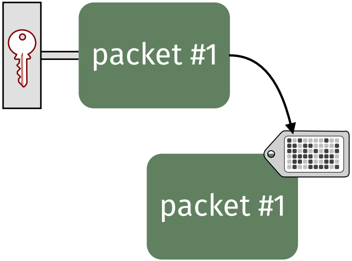
Stream cipher



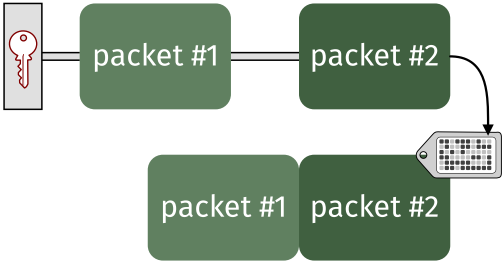
Message authentication code (MAC)



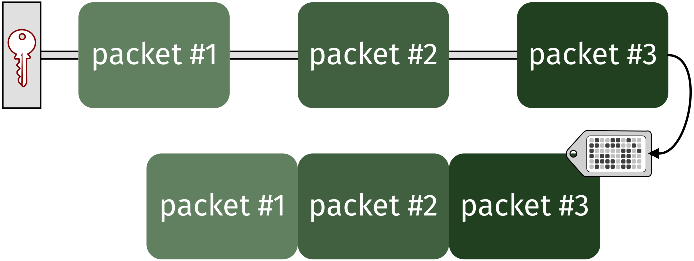
Incremental MACs



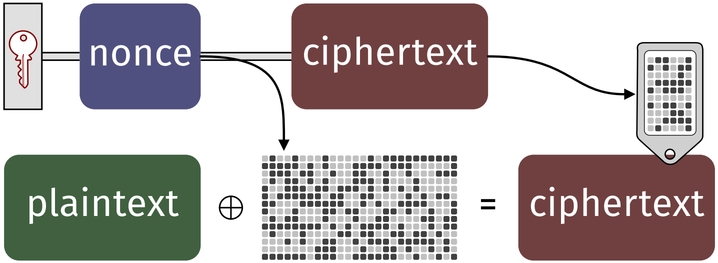
Incremental MACs



Incremental MACs



Authenticated encryption



Deck-SANE: session-supporting and nonce-based

Initialization taking nonce $N \in \mathbb{Z}_2^*$
$e \leftarrow \mathtt{0}^1$
history $\leftarrow N$
return optional setup tag $T = \mathtt{0}^t + F_K(\text{history})$

Wrap taking metadata $A \in \mathbb{Z}_2^*$ and plaintext $P \in \mathbb{Z}_2^*$
$C \leftarrow P + F_K(\text{history}) \ll t$
history $\leftarrow A||\mathtt{0}||e \circ \text{history}$
history $\leftarrow C||\mathtt{1}||e \circ \text{history}$
$T \leftarrow \mathtt{0}^t + F_K(\text{history})$
$e \leftarrow e + \mathtt{1}^1$
return ciphertext C and tag T

Deck-SANSE: session-supporting and SIV-based

Initialization
$e \leftarrow \mathtt{0}^1$
history $\leftarrow$ (the empty string sequence)

Wrap taking metadata $A \in \mathbb{Z}_2^*$ and plaintext $P \in \mathbb{Z}_2^*$
history $\leftarrow A||0||e \circ \text{history}$
$T \leftarrow \mathtt{0}^t + F_K(P||\mathtt{01}||e \circ \text{history})$
$C \leftarrow P + F_K(T||\mathtt{11}||e \circ \text{history})$
history $\leftarrow P||\mathtt{01}||e \circ \text{history}$
$e \leftarrow e + \mathtt{1}^1$
return ciphertext C and tag T

Deck-WBC: wide block cipher

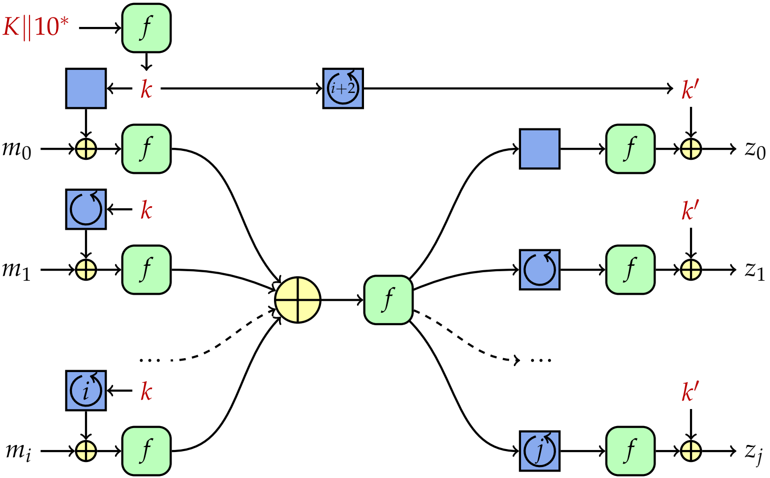
Encipher taking tweak $W \in \mathbb{Z}_2^*$ and plaintext $P \in \mathbb{Z}_2^*$
$(L, R) \leftarrow \text{split}(P)$
$R_0 \leftarrow R_0 + H_K(L||0)$ (R_0: the first $\min(b, |R|)$ bits of R)
$L \leftarrow L + F_K(R||1 \circ W)$
$R \leftarrow R + F_K(L||0 \circ W)$
$L_0 \leftarrow L_0 + H_K(R||1)$ (L_0 the first $\min(b, |L|)$ bits of L)
$C \leftarrow L||R$
return ciphertext $C \in \mathbb{Z}_2^{|P|}$

For more details, see

[Farfalle paper, FSE 2018] [Xoodoo Cookbook, IACR ePrint 2018/767]

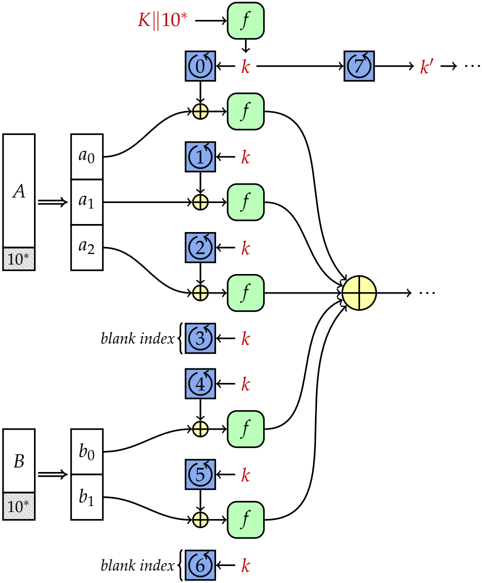
Outline

Farfalle

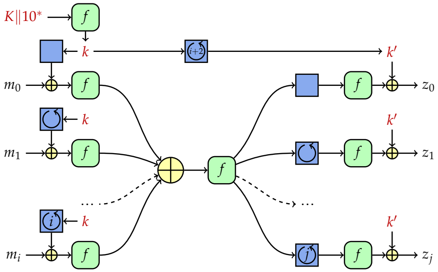


[FSE 2018]

Multi-string input and incrementality



KRAVATTE



- $f = \text{KECCAK-}p[1600, n_r = 6]$
- roll$_c$: simple linear function on 5×64 bits
- roll$_e$: simple non-linear function on 10×64 bits
- Target security: ≥ 128 bits (including post-quantum)

KRAVATTE performance

KRAVATTE		
mask derivation	461	cycles
less than 200 bytes	1236	cycles
MAC computation use case:		
long inputs	0.64	cycles/byte
Stream encryption use case:		
long outputs	0.63	cycles/byte
AES-128 counter mode	0.65	cycles/byte
AES-256 counter mode	0.90	cycles/byte

Intel® Core™ i5-6500 (Skylake), 3.20GHz (no Turbo Boost), single core

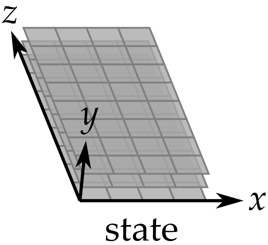
Outline

The Xoodoo permutation

- 384-bit permutation
  Keccak philosophy ported to Gimli shape
  [Bernstein, Kölbl, Lucks, Maat Costa Massolino, Mendel, Nawaz, Schneider, Schwabe, Standaert, Todo, Viguier, CHES 2017]
- Farfalle + Xoodoo = Xoofff
  - Achouffe configuration
  - Efficient on wide range of platforms

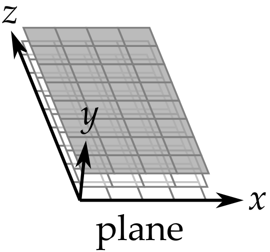
[Xoodoo Cookbook, IACR ePrint 2018/767]

XOODOO state



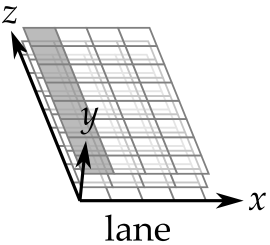
- State: 3 horizontal planes each consisting of 4 lanes

Xoodoo state



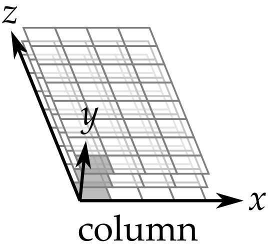
- State: 3 horizontal planes each consisting of 4 lanes

Xoodoo state



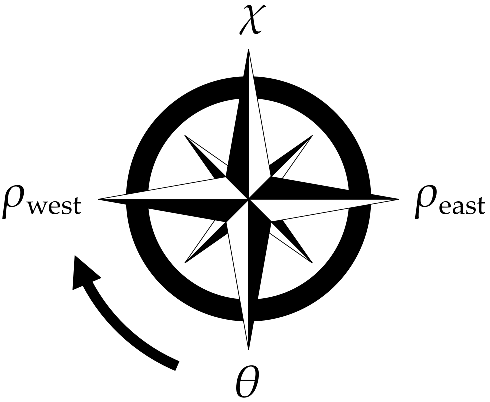
- State: 3 horizontal planes each consisting of 4 lanes

Xoodoo state



- State: 3 horizontal planes each consisting of 4 lanes

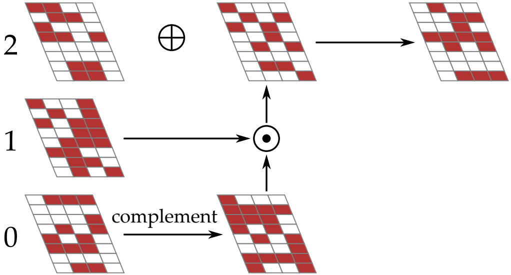
Xoodoo round function



Iterated: n_r rounds that differ only by round constant

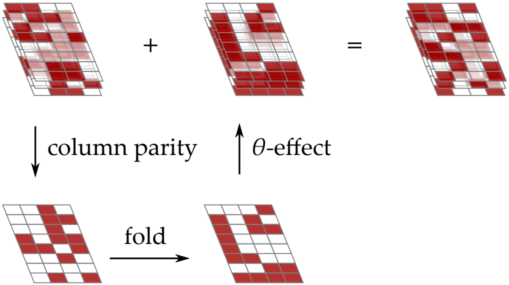
Nonlinear mapping χ

Effect on one plane:



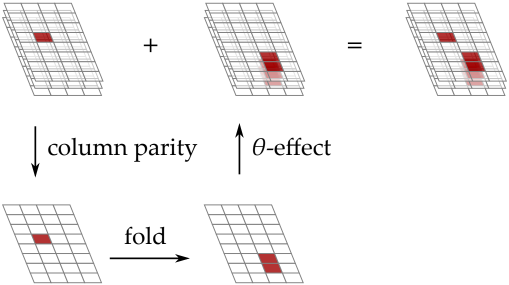
- χ as in Keccak-p, operating on 3-bit columns
- Involution and same propagation differentially and linearly

Mixing layer θ



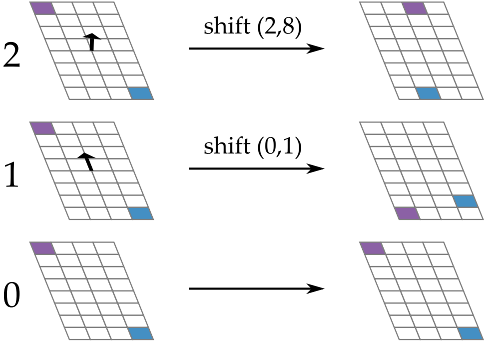
- Column parity mixer: compute parity, fold and add to state
- Good average diffusion, identity for states in *kernel*

Mixing layer θ



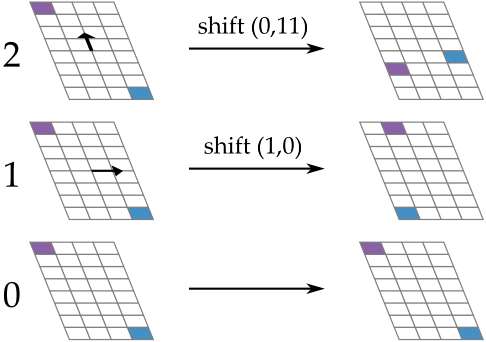
- Column parity mixer: compute parity, fold and add to state
- Good average diffusion, identity for states in *kernel*

Plane shift ρ_{east}



- After χ and before θ
- Shifts planes $y = 1$ and $y = 2$ over different directions

Plane shift ρ_{west}



- After θ and before χ
- Shifts planes $y = 1$ and $y = 2$ over different directions

Xoodoo pseudocode

n_r rounds from $i = 1 - n_r$ to 0, with a 5-step round function:

θ :

$$P \leftarrow A_0 + A_1 + A_2$$
$$E \leftarrow P \lll (1,5) + P \lll (1,14)$$
$$A_y \leftarrow A_y + E \text{ for } y \in \{0,1,2\}$$

ρ_{west} :

$$A_1 \leftarrow A_1 \lll (1,0)$$
$$A_2 \leftarrow A_2 \lll (0,11)$$

ι :

$$A_{0,0} \leftarrow A_{0,0} + \text{rc}_i$$

χ :

$$B_0 \leftarrow \overline{A_1} \cdot A_2$$
$$B_1 \leftarrow \overline{A_2} \cdot A_0$$
$$B_2 \leftarrow \overline{A_0} \cdot A_1$$
$$A_y \leftarrow A_y + B_y \text{ for } y \in \{0,1,2\}$$

ρ_{east} :

$$A_1 \leftarrow A_1 \lll (0,1)$$
$$A_2 \leftarrow A_2 \lll (2,8)$$

Xoodoo software performance

	width	cycles/byte per round	
	bytes	ARM Cortex M3	Intel Skylake
Keccak-$p[1600, n_r]$	200	2.44	0.080
ChaCha	64	0.69	0.059
Gimli	48	0.91	0.074*
Xoodoo	48	1.10	0.083

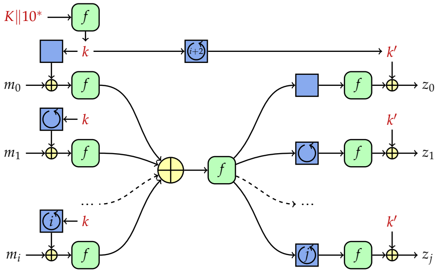
∗ on Intel Haswell

- Xoodoo has slower rounds than Gimli but ...
- ... requires less rounds for equal security objectives!

Trail bounds in Xoodoo

# rounds:	1	2	3	4	5	6
differential:	2	8	36	≥ 70	≥ 82	≥ 104
linear:	2	8	36	≥ 70	≥ 82	≥ 104

XOOFFF



- $f = $ XOODOO[6]
- roll_c: simple linear function on the whole state
- roll_e: simple non-linear function on the whole state
- Target security: $\geq$ 128 bits (96 bits post-quantum)

XOOFFF applications and implementations

Deck-{SANE, SANSE, WBC} using XOOFFF yields:

- XOOFFF-SANE
- XOOFFF-SANSE
- XOOFFF-WBC

[XOODOO Cookbook, IACR ePrint 2018/767]

KECCAK Code Package

⇓

eXtended KECCAK Code Package

[https://github.com/XKCP/XKCP]

Outline

Conclusions

- Symmetric crypto from the p.o.v. of dec(k) functions
- Concrete schemes with dec(k) functions
  - Duplex, STROBE
  - Farfalle, KRAVATTE, XOOFFF

Any questions?

Thanks for your attention!

`https://keccak.team/`

life.augmented