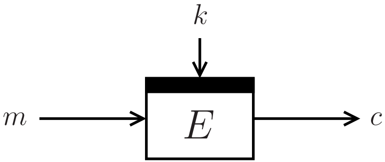
Improved Masking for Tweakable Blockciphers with Applications to Authenticated Encryption

Robert Granger, Philipp Jovanovic, Bart Mennink, Samuel Neves

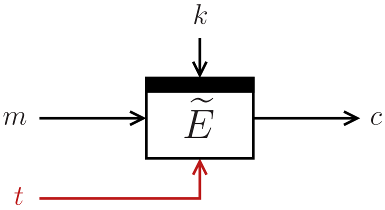
EPFL, EPFL, KU Leuven, University of Coimbra

Dagstuhl — January 12, 2016

Tweakable Blockciphers

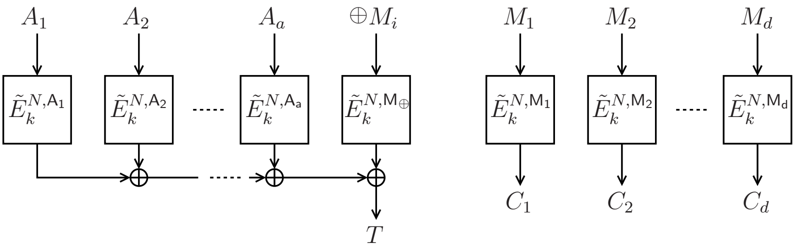


Tweakable Blockciphers



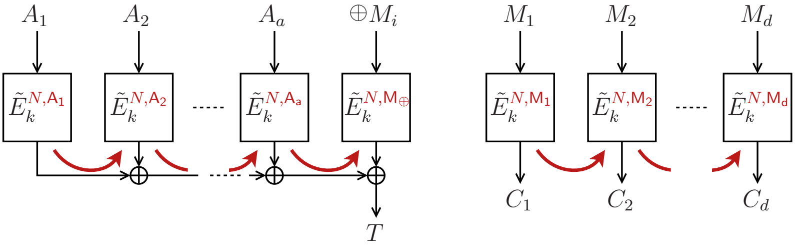
- Tweak: flexibility to the cipher
- Each tweak gives different permutation

Tweakable Blockciphers in OCBx



- Generalized OCB by Rogaway et al. [RBBK01,Rog04,KR11]
- Internally based on tweakable blockcipher $\widetilde{E}$
  - Tweak (N, tweak) is unique for every evaluation

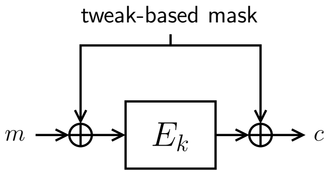
Tweakable Blockciphers in OCBx



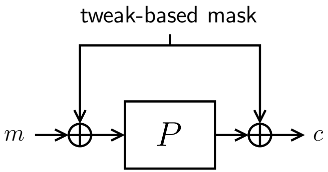
- Generalized OCB by Rogaway et al. [RBBK01,Rog04,KR11]
- Internally based on tweakable blockcipher $\widetilde{E}$
    - Tweak (N, tweak) is unique for every evaluation
- Change of tweak should be efficient

Masking-Based Tweakable Blockciphers

Blockcipher-Based

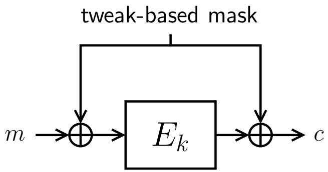


Permutation-Based



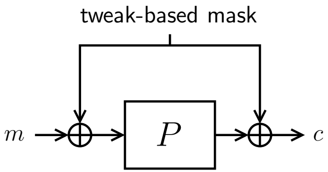
Masking-Based Tweakable Blockciphers

Blockcipher-Based



typically 128 bits

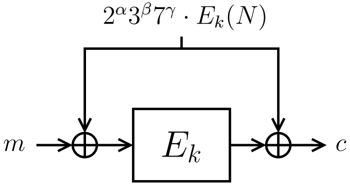
Permutation-Based



much larger: 256-1600 bits

Powering-Up Masking (XEX)

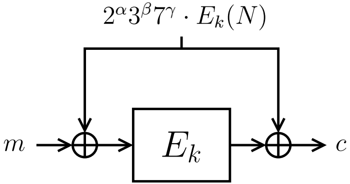
- XEX by Rogaway [Rog04]:



- $(\alpha, \beta, \gamma, N)$ is tweak (simplified)

Powering-Up Masking (XEX)

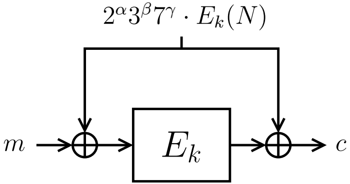
- XEX by Rogaway [Rog04]:

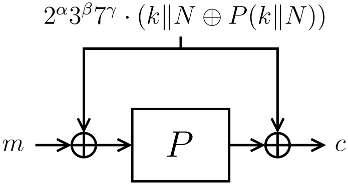


- $(\alpha, \beta, \gamma, N)$ is tweak (simplified)
- Used in OCB2 and in various CAESAR candidates

Powering-Up Masking (XEX)

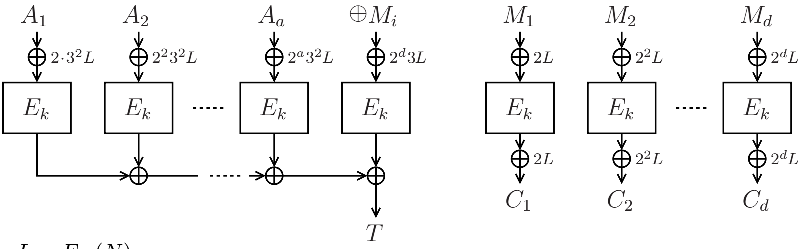
- XEX by Rogaway [Rog04]:





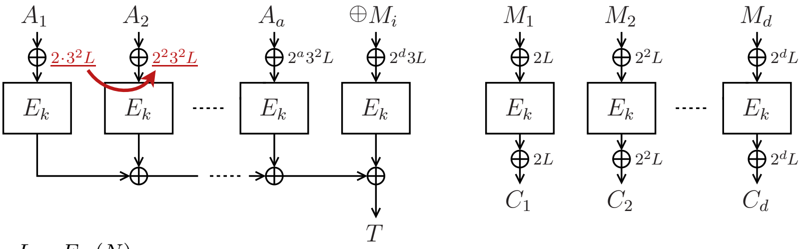
- $(\alpha, \beta, \gamma, N)$ is tweak (simplified)
- Used in OCB2 and in various CAESAR candidates
- Permutation-based variants in Minalpher and Prøst

Powering-Up Masking in OCB2



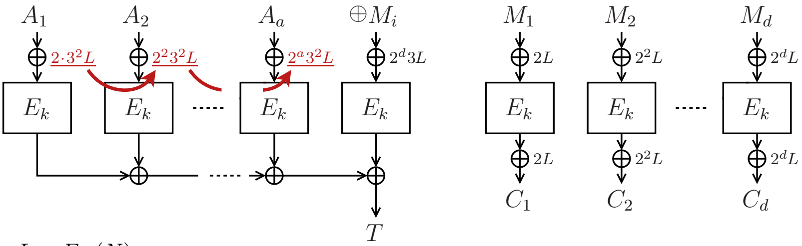
$L = E_K(N)$

Powering-Up Masking in OCB2



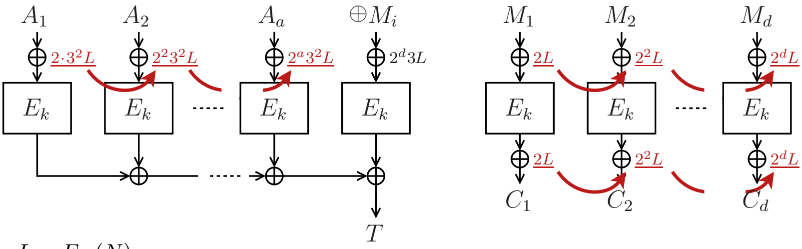
$L = E_K(N)$

Powering-Up Masking in OCB2



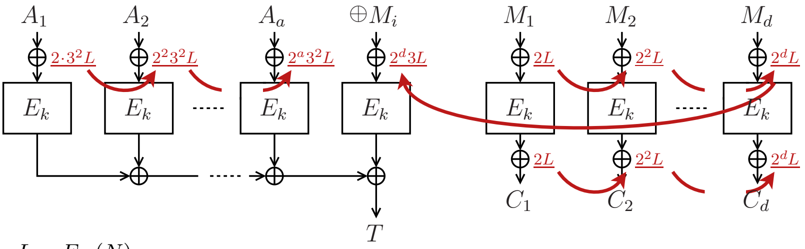
$L = E_K(N)$

Powering-Up Masking in OCB2



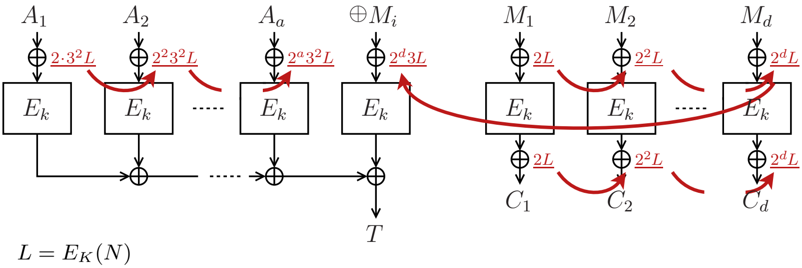
$$L = E_K(N)$$

Powering-Up Masking in OCB2



$L = E_K(N)$

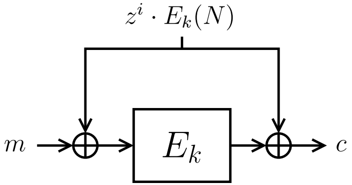
Powering-Up Masking in OCB2



- Update of mask:
  - Shift and conditional XOR
- Variable time computation
- Expensive on certain platforms

Word-Based Powering-Up Masking

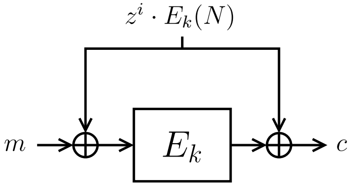
- Chakraborty and Sarkar [CS06]:



- $z \in \{0,1\}^w$ is a generator, (i, N) is tweak
- Tower of fields: $z^i \in \mathbb{F}_{2^w}[z]/g$ instead of $x^i \in \mathbb{F}_2[x]/f$

Word-Based Powering-Up Masking

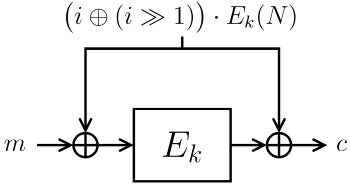
- Chakraborty and Sarkar [CS06]:



- $z \in \{0,1\}^w$ is a generator, (i, N) is tweak
- Tower of fields: $z^i \in \mathbb{F}_{2^w}[z]/g$ instead of $x^i \in \mathbb{F}_2[x]/f$
  - "Word-based powering-up"
  - Similar drawbacks as regular powering-up

Gray Code Masking

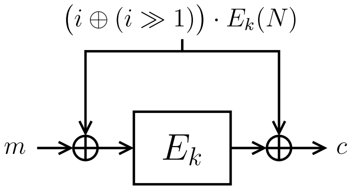
- OCB1 and OCB3 use Gray Codes:



- (i, N) is tweak
- Updating: $G(i) = G(i-1) \oplus 2^{\mathsf{ntz}(i)}$

Gray Code Masking

- OCB1 and OCB3 use Gray Codes:



- (i, N) is tweak
- Updating: $G(i) = G(i-1) \oplus 2^{\mathsf{ntz}(i)}$
  - Single XOR
  - Logarithmic amount of field doublings (precomputed)
- More efficient than powering-up [KR11]

High-Level Contributions

Masked Even-Mansour

- Improved masking of tweakable blockciphers
- Simpler to implement and more efficient
- Constant time (by default)
- Relies on breakthroughs in discrete log computation

High-Level Contributions

Masked Even-Mansour

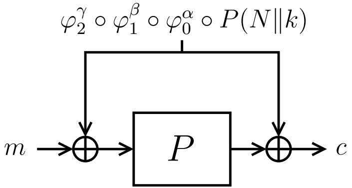
- Improved masking of tweakable blockciphers
- Simpler to implement and more efficient
- Constant time (by default)
- Relies on breakthroughs in discrete log computation

Application to Authenticated Encryption

- Nonce-respecting AE in 0.55 cpb
- Misuse-resistant AE in 1.06 cpb

Masked Even-Mansour (MEM)

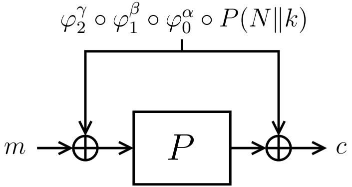
- Masked Even-Mansour (MEM):



- φ_i are fixed LFSRs, $(\alpha, \beta, \gamma, N)$ is tweak (simplified)

Masked Even-Mansour (MEM)

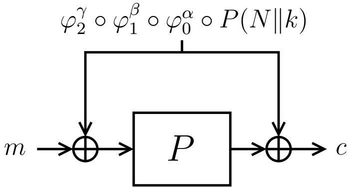
- Masked Even-Mansour (MEM):



- φ_i are fixed LFSRs, $(\alpha, \beta, \gamma, N)$ is tweak (simplified)
- Combines advantages of:
  - Powering-up masking
  - Word-based LFSRs

Masked Even-Mansour (MEM)

- Masked Even-Mansour (MEM):



- φ_i are fixed LFSRs, $(\alpha, \beta, \gamma, N)$ is tweak (simplified)
- Combines advantages of:
  - Powering-up masking
  - Word-based LFSRs
- Simpler, constant-time (by default), more efficient

Design Considerations

- Particularly suited for large states (permutations)
- Low operation counts by clever choice of LFSR

Design Considerations

- Particularly suited for large states (permutations)
- Low operation counts by clever choice of LFSR
- Sample LFSRs (state size b as n words of w bits):

b	w	n	φ
128	8	16	$(x_1, \ldots, x_{15}, (x_0 \lll 1) \oplus (x_9 \gg 1) \oplus (x_{10} \ll 1))$
128	32	4	$(x_1, \ldots, x_3, (x_0 \lll 5) \oplus x_1 \oplus (x_1 \ll 13))$
128	64	2	$(x_1, (x_0 \lll 11) \oplus x_1 \oplus (x_1 \ll 13))$
256	64	4	$(x_1, \ldots, x_3, (x_0 \lll 3) \oplus (x_3 \gg 5))$
512	32	16	$(x_1, \ldots, x_{15}, (x_0 \lll 5) \oplus (x_3 \gg 7))$
512	64	8	$(x_1, \ldots, x_7, (x_0 \lll 29) \oplus (x_1 \ll 9))$
1024	64	16	$(x_1, \ldots, x_{15}, (x_0 \lll 53) \oplus (x_5 \ll 13))$
1600	32	50	$(x_1, \ldots, x_{49}, (x_0 \lll 3) \oplus (x_{23} \gg 3))$
$\vdots$	$\vdots$	$\vdots$	$\vdots$

Design Considerations

- Particularly suited for large states (permutations)
- Low operation counts by clever choice of LFSR
- Sample LFSRs (state size b as n words of w bits):

b	w	n	φ
128	8	16	$(x_1, \dots, x_{15}, (x_0 \lll 1) \oplus (x_9 \gg 1) \oplus (x_{10} \ll 1))$
128	32	4	$(x_1, \dots, x_3, (x_0 \lll 5) \oplus x_1 \oplus (x_1 \ll 13))$
128	64	2	$(x_1, (x_0 \lll 11) \oplus x_1 \oplus (x_1 \ll 13))$
256	64	4	$(x_1, \dots, x_3, (x_0 \lll 3) \oplus (x_3 \gg 5))$
512	32	16	$(x_1, \dots, x_{15}, (x_0 \lll 5) \oplus (x_3 \gg 7))$
512	64	8	$(x_1, \dots, x_7, (x_0 \lll 29) \oplus (x_1 \ll 9))$
1024	64	16	$(x_1, \dots, x_{15}, (x_0 \lll 53) \oplus (x_5 \ll 13))$
1600	32	50	$(x_1, \dots, x_{49}, (x_0 \lll 3) \oplus (x_{23} \gg 3))$
$\vdots$	$\vdots$	$\vdots$	$\vdots$

- Work exceptionally well for ARX primitives

Uniqueness of Masking

- Intuitively, masking goes well as long as

$$\varphi_2^{\gamma} \circ \varphi_1^{\beta} \circ \varphi_0^{\alpha} \neq \varphi_2^{\gamma'} \circ \varphi_1^{\beta'} \circ \varphi_0^{\alpha'}$$

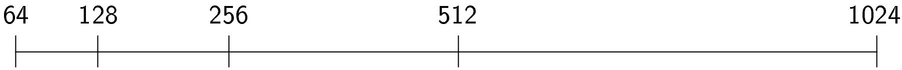
  for any $(\alpha, \beta, \gamma) \neq (\alpha', \beta', \gamma')$
- Challenge: set proper domain for (α, β, γ)
- Requires computation of discrete logarithms

Uniqueness of Masking

- Intuitively, masking goes well as long as

$$\varphi_2^{\gamma} \circ \varphi_1^{\beta} \circ \varphi_0^{\alpha} \neq \varphi_2^{\gamma'} \circ \varphi_1^{\beta'} \circ \varphi_0^{\alpha'}$$

  for any $(\alpha, \beta, \gamma) \neq (\alpha', \beta', \gamma')$
- Challenge: set proper domain for (α, β, γ)
- Requires computation of discrete logarithms

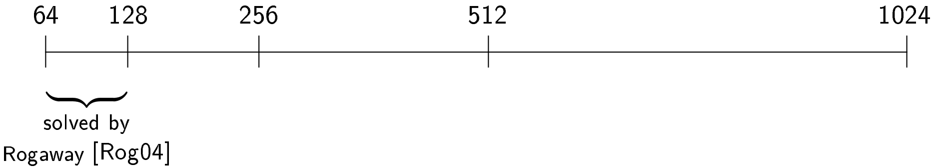


Uniqueness of Masking

- Intuitively, masking goes well as long as

$$\varphi_2^{\gamma} \circ \varphi_1^{\beta} \circ \varphi_0^{\alpha} \neq \varphi_2^{\gamma'} \circ \varphi_1^{\beta'} \circ \varphi_0^{\alpha'}$$

  for any $(\alpha, \beta, \gamma) \neq (\alpha', \beta', \gamma')$
- Challenge: set proper domain for (α, β, γ)
- Requires computation of discrete logarithms

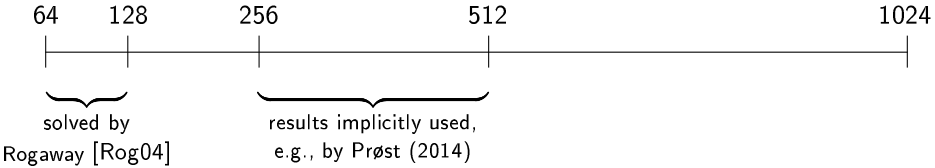


Uniqueness of Masking

- Intuitively, masking goes well as long as

$$\varphi_2^{\gamma} \circ \varphi_1^{\beta} \circ \varphi_0^{\alpha} \neq \varphi_2^{\gamma'} \circ \varphi_1^{\beta'} \circ \varphi_0^{\alpha'}$$

  for any $(\alpha, \beta, \gamma) \neq (\alpha', \beta', \gamma')$
- Challenge: set proper domain for (α, β, γ)
- Requires computation of discrete logarithms

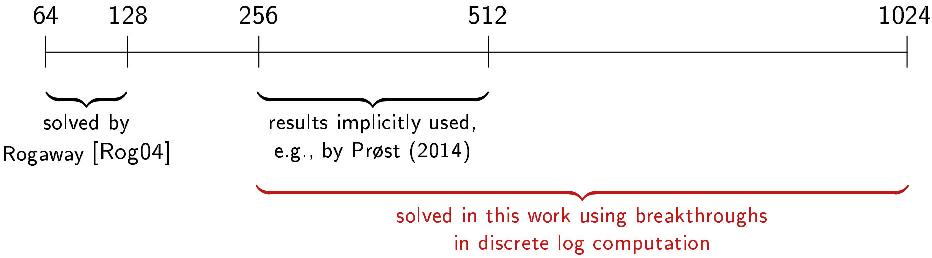


Uniqueness of Masking

- Intuitively, masking goes well as long as

$$\varphi_2^{\gamma} \circ \varphi_1^{\beta} \circ \varphi_0^{\alpha} \neq \varphi_2^{\gamma'} \circ \varphi_1^{\beta'} \circ \varphi_0^{\alpha'}$$

  for any $(\alpha, \beta, \gamma) \neq (\alpha', \beta', \gamma')$
- Challenge: set proper domain for (α, β, γ)
- Requires computation of discrete logarithms



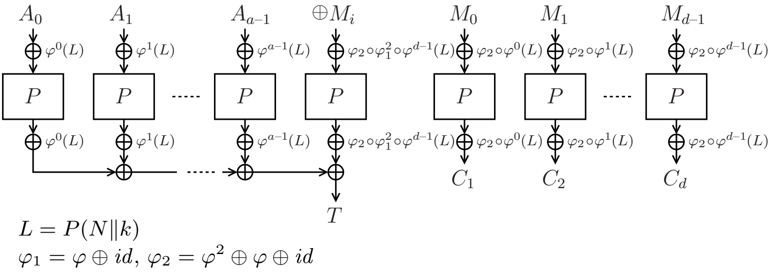
“Bare” Implementation Results

- Mask computation in cycles per update
- In most pessimistic scenario (for ours):

Masking	Sandy Bridge	Haswell
Powering-up	13.108	10.382
Gray code	6.303	3.666
Ours	2.850	2.752

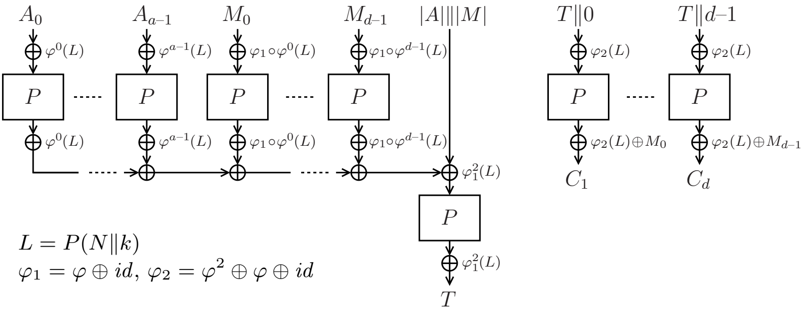
- Differences may amplify/diminish in a mode

Application to AE: OPP



- Offset Public Permutation (OPP)
- Generalization of OCB3:
  - Permutation-based
  - More efficient MEM masking
- Security against nonce-respecting adversaries
- 0.55 cpb with reduced-round BLAKE2b

Application to AE: MRO



$L = P(N\|k)$

$\varphi_1 = \varphi \oplus id$, $\varphi_2 = \varphi^2 \oplus \varphi \oplus id$

- Misuse-Resistant OPP (MRO)
- Fully nonce-misuse resistant version of OPP
- 1.06 cpb with reduced-round BLAKE2b

Implementation

- State size $b = 1024$
- LFSR on 16 words of 64 bits:

$$\varphi(x_0, \ldots, x_{15}) = (x_1, \ldots, x_{15}, (x_0 \lll 53) \oplus (x_5 \ll 13))$$

- P: BLAKE2b permutation with 4 or 6 rounds

Implementation

- State size $b = 1024$
- LFSR on 16 words of 64 bits:

$$\varphi(x_0, \ldots, x_{15}) = (x_1, \ldots, x_{15}, (x_0 \lll 53) \oplus (x_5 \ll 13))$$

- P: BLAKE2b permutation with 4 or 6 rounds

- Main implementation results (more in paper):

	nonce-respecting					misuse-resistant
Platform	AES-GCM	OCB3	Deoxys$^{\neq}$	OPP$_4$	OPP$_6$	
Cortex-A8	38.6	28.9	-	4.26	5.91	
Sandy Bridge	2.55	0.98	1.29	1.24	1.91	
Haswell	1.03	0.69	0.96	0.55	0.75	

Implementation

- State size $b = 1024$
- LFSR on 16 words of 64 bits:

$$\varphi(x_0, \ldots, x_{15}) = (x_1, \ldots, x_{15}, (x_0 \lll 53) \oplus (x_5 \ll 13))$$

- P: BLAKE2b permutation with 4 or 6 rounds

- Main implementation results (more in paper):

	nonce-respecting					misuse-resistant			
Platform	AES-GCM	OCB3	Deoxys$^{\neq}$	OPP$_4$	OPP$_6$	GCM-SIV	Deoxys$^{=}$	MRO$_4$	MRO$_6$
Cortex-A8	38.6	28.9	-	4.26	5.91	-	-	8.07	11.32
Sandy Bridge	2.55	0.98	1.29	1.24	1.91	-	2.58	2.41	3.58
Haswell	1.03	0.69	0.96	0.55	0.75	1.17	1.92	1.06	1.39

Implementation: Parallelizability

- LFSR on 16 words of 64 bits:

$$\varphi(x_0, \ldots, x_{15}) = (x_1, \ldots, x_{15}, (x_0 \lll 53) \oplus (x_5 \ll 13))$$

Implementation: Parallelizability

- LFSR on 16 words of 64 bits:

$$\varphi(x_0, \ldots, x_{15}) = (x_1, \ldots, x_{15}, (x_0 \lll 53) \oplus (x_5 \ll 13))$$

- Begin with state $L_i = [x_0, \ldots, x_{15}]$ of 64-bit words

$$\begin{matrix} x_0 & x_1 & x_2 & x_3 \\ x_4 & x_5 & x_6 & x_7 \\ x_8 & x_9 & x_{10} & x_{11} \\ x_{12} & x_{13} & x_{14} & x_{15} \end{matrix}$$

Implementation: Parallelizability

- LFSR on 16 words of 64 bits:

$$\varphi(x_0, \ldots, x_{15}) = (x_1, \ldots, x_{15}, (x_0 \lll 53) \oplus (x_5 \ll 13))$$

- Begin with state $L_i = [x_0, \ldots, x_{15}]$ of 64-bit words

$$\begin{array}{cccc} x_0 & x_1 & x_2 & x_3 \\ x_4 & x_5 & x_6 & x_7 \\ x_8 & x_9 & x_{10} & x_{11} \\ x_{12} & x_{13} & x_{14} & x_{15} \\ x_{16} & & & \end{array}$$

- $x_{16} = (x_0 \lll 53) \oplus (x_5 \ll 13)$

Implementation: Parallelizability

- LFSR on 16 words of 64 bits:

$$\varphi(x_0, \ldots, x_{15}) = (x_1, \ldots, x_{15}, (x_0 \lll 53) \oplus (x_5 \ll 13))$$

- Begin with state $L_i = [x_0, \ldots, x_{15}]$ of 64-bit words

$$\begin{array}{cccc} x_0 & x_1 & x_2 & x_3 \\ x_4 & x_5 & x_6 & x_7 \\ x_8 & x_9 & x_{10} & x_{11} \\ x_{12} & x_{13} & x_{14} & x_{15} \\ x_{16} & x_{17} & & \end{array}$$

- $x_{16} = (x_0 \lll 53) \oplus (x_5 \ll 13)$
- $x_{17} = (x_1 \lll 53) \oplus (x_6 \ll 13)$

Implementation: Parallelizability

- LFSR on 16 words of 64 bits:

$$\varphi(x_0, \ldots, x_{15}) = (x_1, \ldots, x_{15}, (x_0 \lll 53) \oplus (x_5 \ll 13))$$

- Begin with state $L_i = [x_0, \ldots, x_{15}]$ of 64-bit words

$$\begin{array}{cccc} x_0 & x_1 & x_2 & x_3 \\ x_4 & x_5 & x_6 & x_7 \\ x_8 & x_9 & x_{10} & x_{11} \\ x_{12} & x_{13} & x_{14} & x_{15} \\ x_{16} & x_{17} & x_{18} & \end{array}$$

- $x_{16} = (x_0 \lll 53) \oplus (x_5 \ll 13)$
- $x_{17} = (x_1 \lll 53) \oplus (x_6 \ll 13)$
- $x_{18} = (x_2 \lll 53) \oplus (x_7 \ll 13)$

Implementation: Parallelizability

- LFSR on 16 words of 64 bits:

$$\varphi(x_0, \ldots, x_{15}) = (x_1, \ldots, x_{15}, (x_0 \lll 53) \oplus (x_5 \ll 13))$$

- Begin with state $L_i = [x_0, \ldots, x_{15}]$ of 64-bit words

$$\begin{matrix} x_0 & x_1 & x_2 & x_3 \\ x_4 & x_5 & x_6 & x_7 \\ x_8 & x_9 & x_{10} & x_{11} \\ x_{12} & x_{13} & x_{14} & x_{15} \\ x_{16} & x_{17} & x_{18} & x_{19} \end{matrix}$$

- $x_{16} = (x_0 \lll 53) \oplus (x_5 \ll 13)$
- $x_{17} = (x_1 \lll 53) \oplus (x_6 \ll 13)$
- $x_{18} = (x_2 \lll 53) \oplus (x_7 \ll 13)$
- $x_{19} = (x_3 \lll 53) \oplus (x_8 \ll 13)$

Implementation: Parallelizability

- LFSR on 16 words of 64 bits:

$$\varphi(x_0, \ldots, x_{15}) = (x_1, \ldots, x_{15}, (x_0 \lll 53) \oplus (x_5 \ll 13))$$

- Begin with state $L_i = [x_0, \ldots, x_{15}]$ of 64-bit words

$$\begin{matrix} x_0 & x_1 & x_2 & x_3 \\ x_4 & x_5 & x_6 & x_7 \\ x_8 & x_9 & x_{10} & x_{11} \\ x_{12} & x_{13} & x_{14} & x_{15} \\ x_{16} & x_{17} & x_{18} & x_{19} \end{matrix}$$

- $x_{16} = (x_0 \lll 53) \oplus (x_5 \ll 13)$
- $x_{17} = (x_1 \lll 53) \oplus (x_6 \ll 13)$
- $x_{18} = (x_2 \lll 53) \oplus (x_7 \ll 13)$
- $x_{19} = (x_3 \lll 53) \oplus (x_8 \ll 13)$
- Parallelizable (AVX2) and word-sliceable

Conclusion

Masked Even-Mansour

- Simpler, constant-time (by default), more efficient
- Justified by breakthroughs in discrete log computation
- MEM-based AE outperforms its closest competitors

More Info

- `https://eprint.iacr.org/2015/999`
- `https://github.com/MEM-AEAD`

Thank you for your attention!