Improved Masking for Tweakable Blockciphers with Applications to Authenticated Encryption

Robert Granger[1], **Philipp Jovanovic**[1], Bart Mennink[2], Samuel Neves[3]

[1]École Polytechnique Fédérale de Lausanne, Switzerland

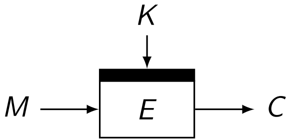
[2]KU Leuven, Belgium

[3]University of Coimbra, Portugal

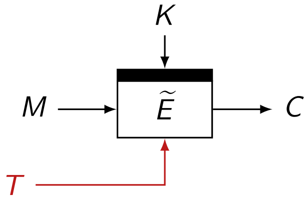
Eurocrypt 2016

Vienna, Austria

Tweakable Blockciphers

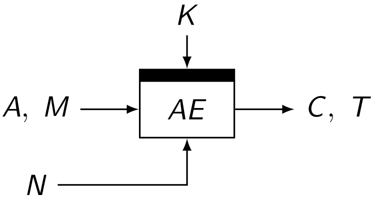


Tweakable Blockciphers



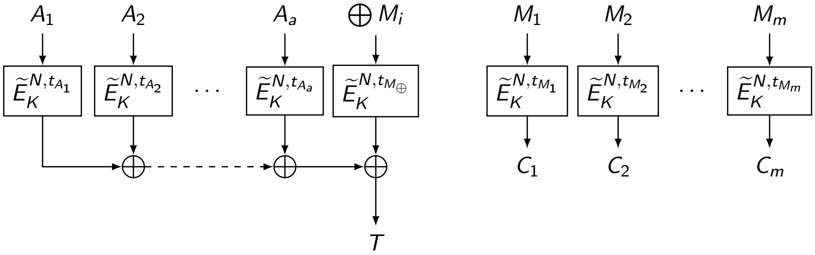
- Tweak T: adds flexibility to the cipher
- Different tweak $\Rightarrow$ different permutation

Authenticated Encryption



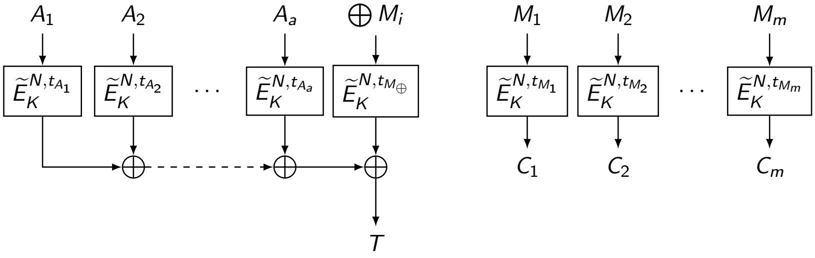
- Ciphertext C is encryption of message M
- Tag T authenticates associated data A and message M
- Nonce N randomizes the scheme (similar to a tweak)

Tweakable Blockciphers in OCBx



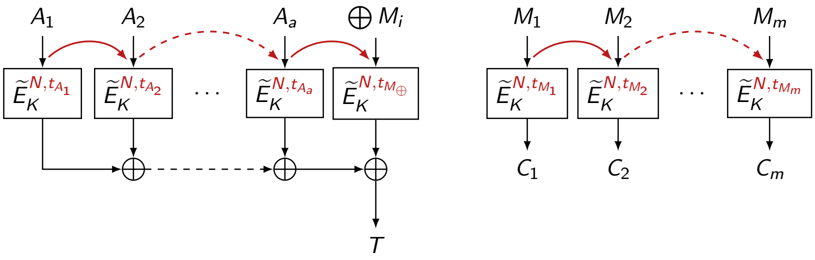
- Generalized OCB by Rogaway et al. [RBBK01,Rog04,KR11]
- Internally based on tweakable blockcipher $\widetilde{E}$

Tweakable Blockciphers in OCBx



- Generalized OCB by Rogaway et al. [RBBK01,Rog04,KR11]
- Internally based on tweakable blockcipher $\widetilde{E}$
- Tweak (N, t):
  - Unique for every evaluation
  - Different blocks always transformed by different tweaks

Tweakable Blockciphers in OCBx



- Generalized OCB by Rogaway et al. [RBBK01,Rog04,KR11]
- Internally based on tweakable blockcipher $\widetilde{E}$
- Tweak (N, t):
  - Unique for every evaluation
  - Different blocks always transformed by different tweaks
  - Change should be efficient

Tweakable Blockciphers

1998: Hasty Pudding Cipher [Sch98]:

- AES submission
- "first tweakable cipher"

2001: Mercy [Cro01] (disk encryption)

2007: Threefish [FLS+07] in SHA-3 submission Skein

2014: TWEAKEY [JNP14] in CAESAR submissions:

- Deoxys
- Joltik
- KIASU

Tweakable Blockciphers

1998: Hasty Pudding Cipher [Sch98]:

- AES submission
- "first tweakable cipher"

2001: Mercy [Cro01] (disk encryption)

2007: Threefish [FLS+07] in SHA-3 submission Skein

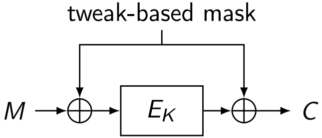
2014: TWEAKEY [JNP14] in CAESAR submissions:

- Deoxys
- Joltik
- KIASU

Our focus: generic tweakable blockcipher design

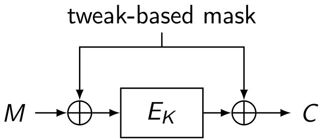
Masking-Based Tweakable Blockciphers

Blockcipher-Based

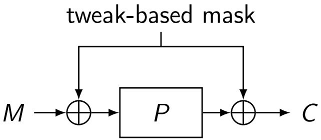


Masking-Based Tweakable Blockciphers

Blockcipher-Based

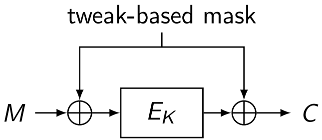


Permutation-Based



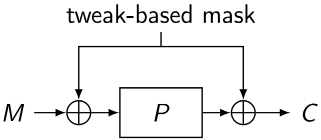
Masking-Based Tweakable Blockciphers

Blockcipher-Based



typically 128 bits

Permutation-Based

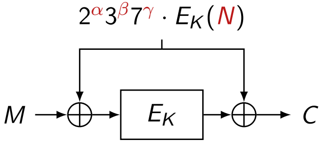


much larger: 256-1600 bits

Powering-Up Masking

XEX

[Rog04]

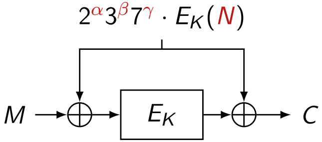


- Tweak (simplified): $(\alpha, \beta, \gamma, N)$

Powering-Up Masking

XEX

[Rog04]

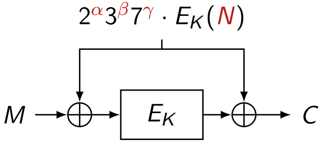


- Tweak (simplified): $(\alpha, \beta, \gamma, N)$
- Used in OCB2 and various CAESAR candidates

Powering-Up Masking

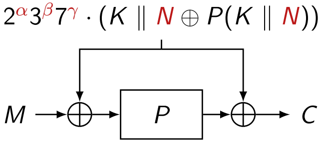
XEX

[Rog04]



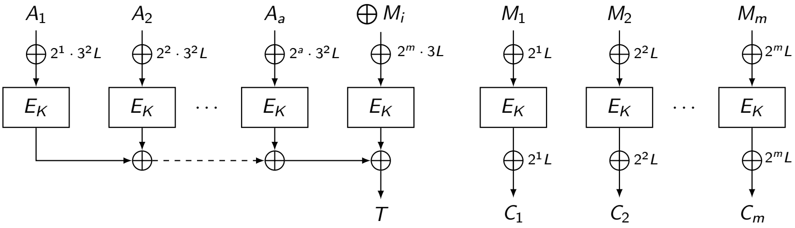
TEM

[STA+14]



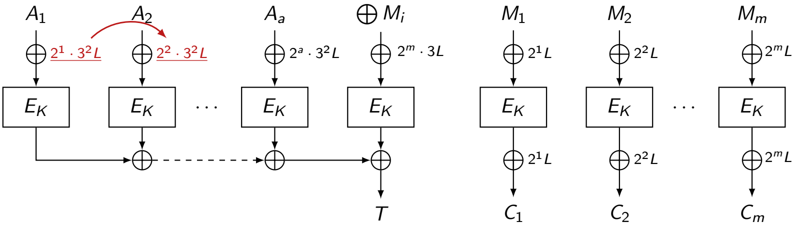
- Tweak (simplified): $(\alpha, \beta, \gamma, N)$
- Used in OCB2 and various CAESAR candidates
- Permutation-based variants: Minalpher and Prøst

Powering-Up Masking in OCB2



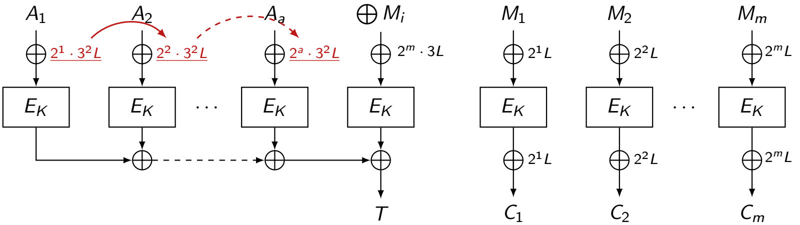
$L = E_K(N)$

Powering-Up Masking in OCB2



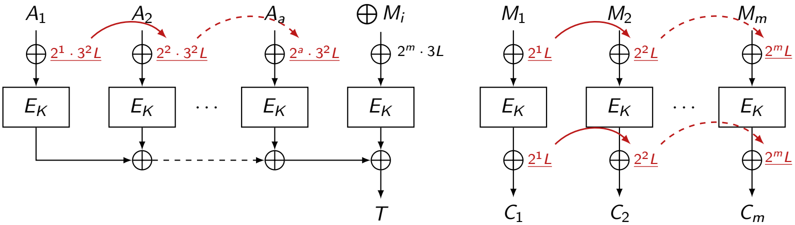
$L = E_K(N)$

Powering-Up Masking in OCB2



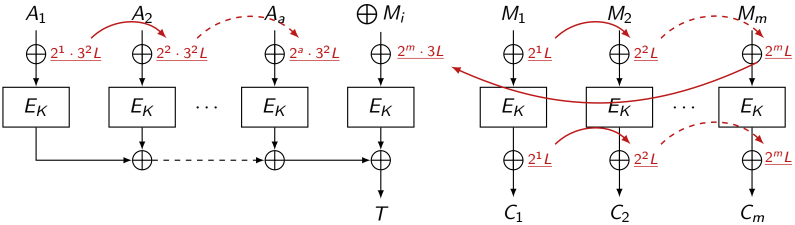
$L = E_K(N)$

Powering-Up Masking in OCB2



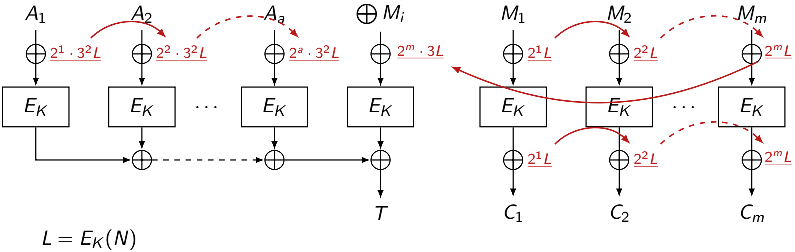
$L = E_K(N)$

Powering-Up Masking in OCB2



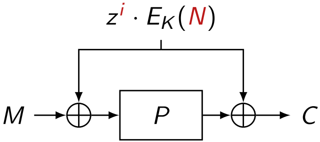
$L = E_K(N)$

Powering-Up Masking in OCB2



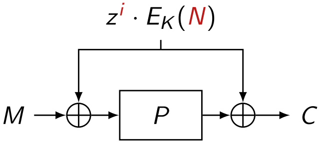
- Update of mask: shift and conditional XOR
- Variable time computation
- Expensive on certain platforms

Word-based Powering-Up Masking



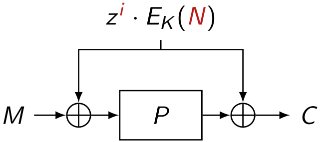
- By Chakraborty and Sarkar [CS06]
- Tweak: (i, N)

Word-based Powering-Up Masking



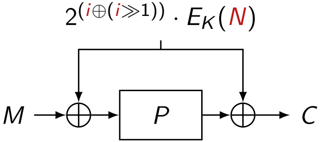
- By Chakraborty and Sarkar [CS06]
- Tweak: (i, N)
- Tower of fields:
  - $z^i \in \mathbb{F}_{2^w}[z]/g$ for $z \in \{0,1\}^w$ ...
  - ... instead of $x^i \in \mathbb{F}_2[x]/f$

Word-based Powering-Up Masking



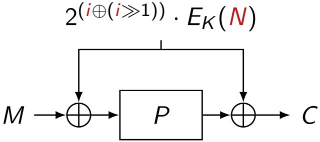
- By Chakraborty and Sarkar [CS06]
- Tweak: (i, N)
- Tower of fields:
  - $z^i \in \mathbb{F}_{2^w}[z]/g$ for $z \in \{0,1\}^w$ ...
  - ... instead of $x^i \in \mathbb{F}_2[x]/f$
- Similar drawbacks as regular powering-up

Gray Code Masking



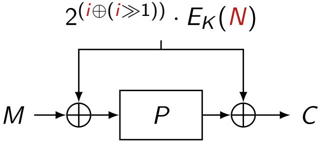
- Used in OCB1 and OCB3
- Tweak: (i, N)
- Updating: $G(i) = G(i-1) \oplus 2^{\mathrm{ntz}(i)} \cdot E_K(N)$

Gray Code Masking



- Used in OCB1 and OCB3
- Tweak: (i, N)
- Updating: $G(i) = G(i-1) \oplus 2^{\mathrm{ntz}(i)} \cdot E_K(N)$
  - Single XOR
  - $\log_2 i$ field doublings (precomputation possible)

Gray Code Masking



- Used in OCB1 and OCB3
- Tweak: (i, N)
- Updating: $G(i) = G(i-1) \oplus 2^{\mathrm{ntz}(i)} \cdot E_K(N)$
  - Single XOR
  - $\log_2 i$ field doublings (precomputation possible)
- 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

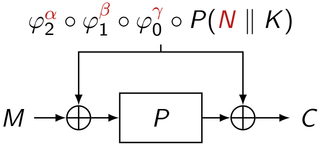
Masked Even-Mansour

- 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

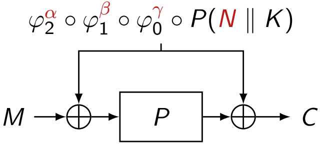
- Nonce-respecting AE in 0.55 cpb
- Misuse-resistant AE in 1.06 cpb

Masked Even-Mansour (MEM)



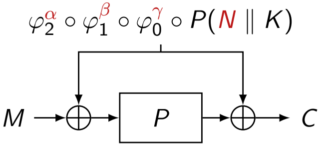
- Fixed LFSRs: φ_i
- Tweak (simplified): $(\alpha, \beta, \gamma, N)$

Masked Even-Mansour (MEM)



- Fixed LFSRs: φ_i
- Tweak (simplified): $(\alpha, \beta, \gamma, N)$
- Combines advantages of:
  - Powering-up masking
  - Word-based LFSRs

Masked Even-Mansour (MEM)



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

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	$(x_1, \ldots, x_{15}, (x_0 \lll 2) \oplus ((x_4 \parallel x_3) \gg 3)$
128	32	4	$(x_1, \ldots, x_3, (x_0 \lll 5) \oplus x_1 \oplus (x_1 \ll 13))$
128	64	2	$(x_1, (x_0 \lll 4) \oplus ((x_1 \parallel x_0) \gg 25)$
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	$(x_1, \ldots, x_{49}, (x_0 \lll 3) \oplus (x_{23} \gg 3))$
1600	64	25	$(x_1, \ldots, x_{24}, (x_0 \lll 14) \oplus ((x_1 \parallel x_0) \gg 13)$
$\vdots$	$\vdots$	$\vdots$	$\vdots$

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	$(x_1, \ldots, x_{15}, (x_0 \lll 2) \oplus ((x_4 \parallel x_3) \gg 3)$
128	32	4	$(x_1, \ldots, x_3, (x_0 \lll 5) \oplus x_1 \oplus (x_1 \ll 13))$
128	64	2	$(x_1, (x_0 \lll 4) \oplus ((x_1 \parallel x_0) \gg 25)$
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	$(x_1, \ldots, x_{49}, (x_0 \lll 3) \oplus (x_{23} \gg 3))$
1600	64	25	$(x_1, \ldots, x_{24}, (x_0 \lll 14) \oplus ((x_1 \parallel x_0) \gg 13)$
$\vdots$	$\vdots$	$\vdots$	$\vdots$

- Work exceptionally well for ARX primitives

Uniqueness of Masking

- Intuitively, masking goes well as long as

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

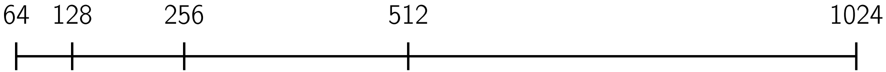
  for any $(\alpha, \beta, \gamma) \neq (\alpha', \beta', \gamma')$
- Challenge: set proper domain for (α, β, γ)
- Requires computation of discrete logarithms

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

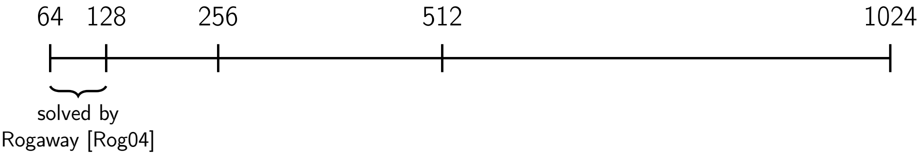


Uniqueness of Masking

- 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

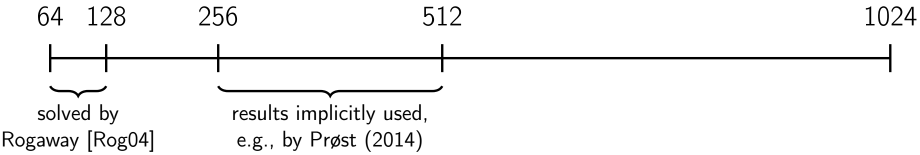


Uniqueness of Masking

- 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

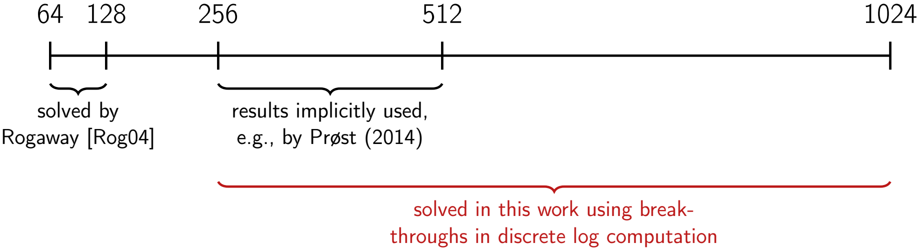


Uniqueness of Masking

- 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

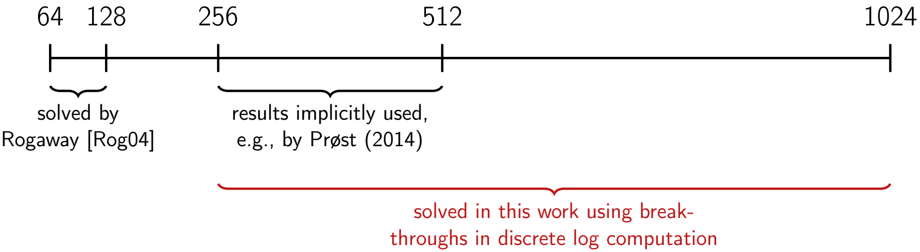


Uniqueness of Masking

- 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



- Logs for 2^{11}, 2^{12}, 2^{13} easily doable with latest techniques

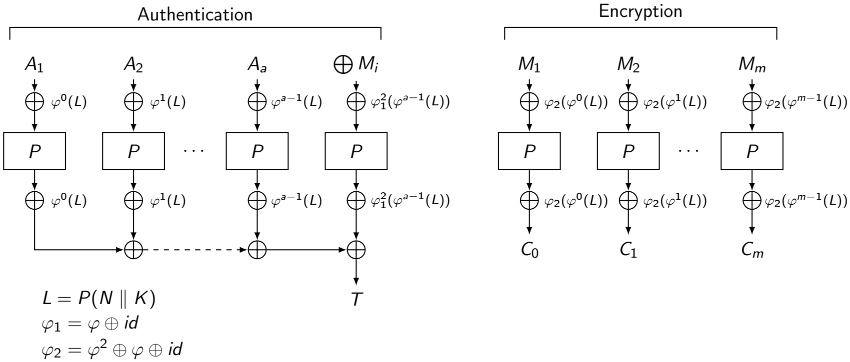
“Bare” Implementation Results

- 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

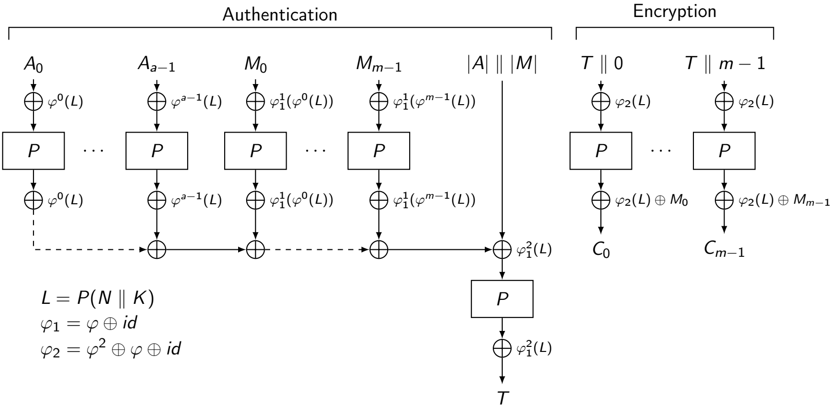
- Differences may amplify/diminish in a mode

Application to AE: OPP



- Offset Public Permutation (OPP)
- Security against nonce-respecting adversaries

Application to AE: MRO



- Misuse-Resistant OPP (MRO)
- Fully nonce-misuse resistant version of OPP

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:

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

- 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:

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

- 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

- 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

- OPP: $\approx$ 6.36 GiBps, MRO: $\approx$ 3.30 GiBps

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

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

Implementation: Parallelizability

- LFSR on 16 words of 64 bits:

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

- Begin with state $L_i = [x_0, \dots, x_{15}]$ 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

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

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

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

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}

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

Conclusion

Masked Even-Mansour

- Simple, efficient, constant-time (by default)
- Justified by breakthroughs in discrete log computation
- MEM-based AE is able to outperform its closest competitors

Conclusion

Masked Even-Mansour

- Simple, efficient, constant-time (by default)
- Justified by breakthroughs in discrete log computation
- MEM-based AE is able to outperform its closest competitors

More Info

- `https://eprint.iacr.org/2015/999` (full version)
- `https://github.com/MEM-AEAD`

Conclusion

Masked Even-Mansour

- Simple, efficient, constant-time (by default)
- Justified by breakthroughs in discrete log computation
- MEM-based AE is able to outperform its closest competitors

More Info

- `https://eprint.iacr.org/2015/999` (full version)
- `https://github.com/MEM-AEAD`

`// Thank You`

Support: Masking Function Search

- Basis:

$$M = \begin{pmatrix} 0 & I & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & I \\ X_0 & X_1 & \cdots & X_{n-1} \end{pmatrix} \in \mathbb{F}_{2^{nw}} \times \mathbb{F}_{2^{nw}}$$

  with $X_i \in \{0, I, \mathrm{SHL}_c, \mathrm{SHR}_c, \mathrm{ROT}_c, \mathrm{AND}_c\}$, $\dim(X_i) = w$
- Check: minimal polynomial of M is primitive of degree b
- Then: $\varphi^i(L) = M^i \cdot L$ has period $2^b - 1$
- Note:

$$\varphi : (x_0, \ldots, x_{n-1}) \mapsto (x_1, \ldots, x_{n-1}, f(x_0, \ldots, x_{n-1}))$$

Support: Tweak Space Domain Separation

Lemma

- $\varphi : \{0,1\}^{1024} \mapsto \{0,1\}^{1024}$, *with*

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

and associated transformation matrix M

- $\varphi_0^{i_0}(L) = M^{i_0} \cdot L$,
- $\varphi_1^{i_1}(L) = (M + I)^{i_1} \cdot L$
- $\varphi_2^{i_2}(L) = (M^2 + M + I)^{i_2} \cdot L$

The tweak space

$$\mathcal{T} = \mathcal{T}_0 \times \mathcal{T}_1 \times \mathcal{T}_2 = \{0, 1, \ldots, 2^{1020} - 1\} \times \{0, 1, 2, 3\} \times \{0, 1\}$$

is b-proper relative to the function set $\{\varphi_0^{i_0}, \varphi_1^{i_1}, \varphi_2^{i_2}\}$.

Support: Tweak Space Domain Separation via Lattices

- Lattice spanned by rows of

$$\begin{pmatrix} K \cdot 1 & w_0 & 0 & 0 \\ K \cdot l_1 & 0 & w_1 & 0 \\ K \cdot l_2 & 0 & 0 & w_2 \\ K \cdot m & 0 & 0 & 0 \end{pmatrix}$$

  for integers K, $m = 2^b - 1$, weights w_i, and dlogs l_1, l_2
- Then

$$(\Delta i_0 + \Delta i_1 l_1 + \Delta i_2 l_2 + km, \Delta i_0 w_0, \Delta i_1 w_1, \Delta i_2 w_2)$$

  is shortest vector if

$$\Delta i_0 + \Delta i_1 l_1 + \Delta i_2 l_2 \equiv 0 \pmod{2^n - 1}$$

- For $(w_0, w_1, w_2) = (1, 2^{1019}, 2^{1022})$, similar tweak space as in Lemma on last slide