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

\[ E \]

\[ m \rightarrow c \]

- Tweakable flexibility to the cipher
- Each tweak gives different permutation
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 $\tilde{E}$
  - Tweak $(N, \text{tweak})$ is unique for every evaluation
Tweakable Blockciphers in OCBx

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

Blockcipher-Based

Permutation-Based

```
\[
\begin{align*}
m & \rightarrow E_k \rightarrow c \\
\text{tweak-based mask} & \\
m & \rightarrow P \rightarrow c \\
\text{tweak-based mask}
\end{align*}
\]
Masking-Based Tweakable Blockciphers

Blockcipher-Based

- tweak-based mask

\[ m \xrightarrow{E_k} c \]

- typically 128 bits

Permutation-Based

- tweak-based mask

\[ m \xrightarrow{P} c \]

- much larger: 256-1600 bits
Powering-Up Masking (XEX)

- XEX by Rogaway [Rog04]:

\[ 2^{\alpha}3^\beta7^\gamma \cdot E_k(N) \]

- \((\alpha, \beta, \gamma, N)\) is tweak (simplified)
Powering-Up Masking (XEX)

- XEX by Rogaway [Rog04]:

\[ 2^α3^β7^γ \cdot E_k(N) \]

- \((α, β, γ, N)\) is tweak (simplified)
- Used in OCB2 and in various CAESAR candidates
Powering-Up Masking (XEX)

- XEX by Rogaway [Rog04]:

\[ 2^{\alpha} 3^{\beta} 7^{\gamma} \cdot E_k(N) \]

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

\[ L = E_K(N) \]

\[ A_1 \oplus 2 \cdot 3^2 L \]
\[ A_2 \oplus 2^2 3^2 L \]
\[ A_{\alpha} \oplus 2^a 3^2 L \]
\[ \oplus M_i \oplus 2^d 3^2 L \]

\[ E_k \]

\[ M_1 \oplus 2L \]
\[ M_2 \oplus 2^2 L \]
\[ M_d \oplus 2^d L \]

\[ E_k \]

\[ C_1 \]
\[ C_2 \]
\[ C_d \]

• Update of mask:
• Shift and conditional X OR
• Variable time computation
• Expensive on certain platforms
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

$L = E_K(N)$
Word-Based Powering-Up Masking

- Chakraborty and Sarkar [CS06]:

\[
z^i \cdot E_k(N)
\]

- \(z \in \{0, 1\}^w\) 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]:

\[ z^i \cdot E_k(N) \]

\[
\begin{array}{ccc}
 m & \rightarrow & E_k \\
 \quad & \downarrow & \quad \\
 & \rightarrow & c
\end{array}
\]

- \( z \in \{0, 1\}^w \) 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:

\[(i \oplus (i \gg 1)) \cdot E_k(N)\]

- \((i, N)\) is tweak
- Updating: \(G(i) = G(i - 1) \oplus 2^{\text{ntz}(i)}\)
Gray Code Masking

- OCB1 and OCB3 use Gray Codes:

\[(i \oplus (i \gg 1)) \cdot E_k(N)\]

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

\[ \varphi_2 \circ \varphi_1 \circ \varphi_0 \circ P(N\|k) \]

- \( \varphi_i \) are fixed LFSRs, \((\alpha, \beta, \gamma, N)\) is tweak (simplified)
Masked Even-Mansour (MEM)

- Masked Even-Mansour (MEM):

\[ \varphi_2 \circ \varphi_1 \circ \varphi_0 \circ P(N \| k) \]

- \( \varphi_i \) 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):

\[ \varphi_2 \circ \varphi_1 \circ \varphi_0 \circ P(N\|k) \]

- \( \varphi_i \) 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):

<table>
<thead>
<tr>
<th>$b$</th>
<th>$w$</th>
<th>$n$</th>
<th>$\varphi$</th>
</tr>
</thead>
<tbody>
<tr>
<td>128</td>
<td>8</td>
<td>16</td>
<td>$(x_1, \ldots, x_{15}, (x_0 \ll 1) \oplus (x_9 \gg 1) \oplus (x_{10} \ll 1))$</td>
</tr>
<tr>
<td>128</td>
<td>32</td>
<td>4</td>
<td>$(x_1, \ldots, x_3, (x_0 \ll 5) \oplus x_1 \oplus (x_1 \ll 13))$</td>
</tr>
<tr>
<td>128</td>
<td>64</td>
<td>2</td>
<td>$(x_1, (x_0 \ll 11) \oplus x_1 \oplus (x_1 \ll 13))$</td>
</tr>
<tr>
<td>256</td>
<td>64</td>
<td>4</td>
<td>$(x_1, \ldots, x_3, (x_0 \ll 3) \oplus (x_3 \gg 5))$</td>
</tr>
<tr>
<td>512</td>
<td>32</td>
<td>16</td>
<td>$(x_1, \ldots, x_{15}, (x_0 \ll 5) \oplus (x_3 \gg 7))$</td>
</tr>
<tr>
<td>512</td>
<td>64</td>
<td>8</td>
<td>$(x_1, \ldots, x_7, (x_0 \ll 29) \oplus (x_1 \ll 9))$</td>
</tr>
<tr>
<td>1024</td>
<td>64</td>
<td>16</td>
<td>$(x_1, \ldots, x_{15}, (x_0 \ll 53) \oplus (x_5 \ll 13))$</td>
</tr>
<tr>
<td>1600</td>
<td>32</td>
<td>50</td>
<td>$(x_1, \ldots, x_{49}, (x_0 \ll 3) \oplus (x_{23} \gg 3))$</td>
</tr>
</tbody>
</table>
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):

<table>
<thead>
<tr>
<th>$b$</th>
<th>$w$</th>
<th>$n$</th>
<th>$\varphi$</th>
</tr>
</thead>
<tbody>
<tr>
<td>128</td>
<td>8</td>
<td>16</td>
<td>$(x_1, \ldots, x_{15}, (x_0 \ll 1) \oplus (x_9 \gg 1) \oplus (x_{10} \ll 1))$</td>
</tr>
<tr>
<td>128</td>
<td>32</td>
<td>4</td>
<td>$(x_1, \ldots, x_3, (x_0 \ll 5) \oplus x_1 \oplus (x_1 \ll 13))$</td>
</tr>
<tr>
<td>128</td>
<td>64</td>
<td>2</td>
<td>$(x_1, (x_0 \ll 11) \oplus x_1 \oplus (x_1 \ll 13))$</td>
</tr>
<tr>
<td>256</td>
<td>64</td>
<td>4</td>
<td>$(x_1, \ldots, x_3, (x_0 \ll 3) \oplus (x_3 \gg 5))$</td>
</tr>
<tr>
<td>512</td>
<td>32</td>
<td>16</td>
<td>$(x_1, \ldots, x_{15}, (x_0 \ll 5) \oplus (x_3 \gg 7))$</td>
</tr>
<tr>
<td>512</td>
<td>64</td>
<td>8</td>
<td>$(x_1, \ldots, x_7, (x_0 \ll 29) \oplus (x_1 \ll 9))$</td>
</tr>
<tr>
<td>1024</td>
<td>64</td>
<td>16</td>
<td>$(x_1, \ldots, x_{15}, (x_0 \ll 53) \oplus (x_5 \ll 13))$</td>
</tr>
<tr>
<td>1600</td>
<td>32</td>
<td>50</td>
<td>$(x_1, \ldots, x_{49}, (x_0 \ll 3) \oplus (x_{23} \gg 3))$</td>
</tr>
</tbody>
</table>

- 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 \((\alpha, \beta, \gamma)\)

- Requires computation of discrete logarithms
Uniqueness of Masking

- Intuitively, masking goes well as long as

\[ \varphi_2^\g \circ \varphi_1^\b \circ \varphi_0^\a \neq \varphi_2^\g' \circ \varphi_1^\b' \circ \varphi_0^\a' \]

for any \((\alpha, \beta, \gamma) \neq (\alpha', \beta', \gamma')\)

- Challenge: set proper domain for \((\alpha, \beta, \gamma)\)

- Requires computation of discrete logarithms

64 128 256 512 1024
Uniqueness of Masking

- Intuitively, masking goes well as long as
  \[ \varphi_2 \circ \varphi_1 \circ \varphi_0 \neq \varphi_2' \circ \varphi_1' \circ \varphi_0' \]
  for any \((\alpha, \beta, \gamma) \neq (\alpha', \beta', \gamma')\)
- Challenge: set proper domain for \((\alpha, \beta, \gamma)\)
- Requires computation of discrete logarithms

\[
\begin{align*}
64 & \quad 128 & \quad 256 & \quad 512 & \quad 1024 \\
\end{align*}
\]

\textit{solved by Rogaway [Rog04]}
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 \((\alpha, \beta, \gamma)\)

- Requires computation of discrete logarithms

<table>
<thead>
<tr>
<th>64</th>
<th>128</th>
<th>256</th>
<th>512</th>
<th>1024</th>
</tr>
</thead>
</table>

solved by Rogaway [Rog04]

results implicitly used, e.g., by Prøst (2014)
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 \((\alpha, \beta, \gamma)\)

• Requires computation of discrete logarithms

---

64  128  256  512  1024

\[ \underline{\text{solved by Rogaway [Rog04]}} \]

\[ \underline{\text{results implicitly used, e.g., by Prøst (2014)}} \]

\[ \underline{\text{solved in this work using breakthroughs in discrete log computation}} \]
“Bare” Implementation Results

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

<table>
<thead>
<tr>
<th>Masking</th>
<th>Sandy Bridge</th>
<th>Haswell</th>
</tr>
</thead>
<tbody>
<tr>
<td>Powering-up</td>
<td>13.108</td>
<td>10.382</td>
</tr>
<tr>
<td>Gray code</td>
<td>6.303</td>
<td>3.666</td>
</tr>
<tr>
<td>Ours</td>
<td>2.850</td>
<td>2.752</td>
</tr>
</tbody>
</table>

- 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**

- 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:
  \[
  \varphi(x_0, \ldots, x_{15}) = (x_1, \ldots, x_{15}, (x_0 \ll 53) \oplus (x_5 \ll 13))
  \]
- $P$: BLAKE2b permutation with 4 or 6 rounds
Implementation

- State size $b = 1024$
- LFSR on 16 words of 64 bits:
  \[
  \varphi(x_0, \ldots, x_{15}) = (x_1, \ldots, x_{15}, (x_0 \ll 53) \oplus (x_5 \ll 13))
  \]
- $P$: BLAKE2b permutation with 4 or 6 rounds
- Main implementation results (more in paper):

<table>
<thead>
<tr>
<th>Platform</th>
<th>nonce-respecting</th>
<th>misuse-resistant</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>AES-GCM</td>
<td>OCB3</td>
</tr>
<tr>
<td>Cortex-A8</td>
<td>38.6</td>
<td>28.9</td>
</tr>
<tr>
<td>Sandy Bridge</td>
<td>2.55</td>
<td>0.98</td>
</tr>
<tr>
<td>Haswell</td>
<td>1.03</td>
<td>0.69</td>
</tr>
</tbody>
</table>
Implementation

- State size $b = 1024$
- LFSR on 16 words of 64 bits:
  \[
  \varphi(x_0, \ldots, x_{15}) = (x_1, \ldots, x_{15}, (x_0 \ll 53) \oplus (x_5 \ll 13))
  \]
- $P$: BLAKE2b permutation with 4 or 6 rounds

Main implementation results (more in paper):

<table>
<thead>
<tr>
<th>Platform</th>
<th>nonce-respecting</th>
<th>misuse-resistant</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>AES-GCM</td>
<td>OCB3</td>
</tr>
<tr>
<td>Cortex-A8</td>
<td>38.6</td>
<td>28.9</td>
</tr>
<tr>
<td>Sandy Bridge</td>
<td>2.55</td>
<td>0.98</td>
</tr>
<tr>
<td>Haswell</td>
<td>1.03</td>
<td>0.69</td>
</tr>
</tbody>
</table>
Implementation: Parallelizability

- LFSR on 16 words of 64 bits:

\[ \varphi(x_0, \ldots, x_{15}) = (x_1, \ldots, x_{15}, (x_0 \ll 53) \oplus (x_5 \ll 13)) \]
Implementation: Parallelizability

- LFSR on 16 words of 64 bits:
  \[ \varphi(x_0, \ldots, x_{15}) = (x_1, \ldots, x_{15}, (x_0 \ll 53) \oplus (x_5 \ll 13)) \]

- 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:

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

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

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

- \( x_{16} = (x_0 \ll 53) \oplus (x_5 \ll 13) \)
Implementation: Parallelizability

- LFSR on 16 words of 64 bits:

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

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

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

- \( x_{16} = (x_0 \ll 53) \oplus (x_5 \ll 13) \)
- \( x_{17} = (x_1 \ll 53) \oplus (x_6 \ll 13) \)
Implementation: Parallelizability

- LFSR on 16 words of 64 bits:
  \[ \varphi(x_0, \ldots, x_{15}) = (x_1, \ldots, x_{15}, (x_0 \ll 53) \oplus (x_5 \ll 13)) \]

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

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

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

- LFSR on 16 words of 64 bits:
  \[ \varphi(x_0, \ldots, x_{15}) = (x_1, \ldots, x_{15}, (x_0 \ll 53) \oplus (x_5 \ll 13)) \]

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

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

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

- **LFSR on 16 words of 64 bits:**

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

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

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

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