Techniques for Masking Saber and Kyber

Michiel Van Beirendonck       Jan-Pieter D’Anvers
June 2021
Techniques for Masking Saber and Kyber

Synthesis presentation of two works


And related approaches

- [OSPG18, SPOG19, BGR+21] ...
Today’s focus

Masking
- Technique to protect against DPA

Saber & Kyber: MLW(E/R)-based KEM finalists
- KeyGen, Encaps, Decaps

In our experiments, we found Saber easier and more efficient to mask

\[
\begin{array}{cc}
\text{Saber} & \text{Kyber} \\
q = 2^{13} & q = 3329 \\
\text{MLWR} & \text{MLWE}
\end{array}
\]

- Due to
Masking

\[
\begin{array}{cccc}
s_0 & s_1 & s_2 & s_3 \\
\end{array}
\]

\[
\begin{array}{cccc}
B_0^0 & B_1^0 & B_2^0 & B_3^0 \\
\end{array}
\]

\[n = 2\]
Masking

\[
s_0 \quad s_1 \quad s_2 \quad s_3 \quad s_4
\]

\[
\sum \quad B_0^0 \quad B_1^0 \quad B_2^0 \quad B_3^0 \quad B_4^0
\]

\[
\mod q
\]

\[
A_0^0 \quad A_1^0 \quad A_2^0 \quad A_3^0 \quad A_4^0
\]
Masking

\[ s_0 \quad s_1 \quad s_2 \quad s_3 \quad s_4 \]

\[ B_0^0 \quad B_1^0 \quad B_2^0 \quad B_3^0 \quad B_4^0 \]

\[ A_0^0 \quad A_1^0 \quad A_2^0 \quad A_3^0 \quad A_4^0 \]

\[ \sum \quad \oplus \quad \text{mod } q \]
B2A and A2B

More efficient for power-of-two $q = 2^k$ than prime $q$

Algorithms

- In [BDK+21]: Goubin’s B2A$_{2^k}$ [Gou01], table-based A2B$_{2^k}$ [Deb12, VBDV21]
  - Efficient first-order software masking
- In [BGR+21]: SecAdd-based B2A, A2B [CGV14]
  - Common hardware for B2A$_{\{2^k,q\}}$, A2B$_{\{2^k,q\}}$
  - Efficient hardware with Threshold Implementations
  - Extensible to higher-order masking
- Additionally in this presentation: SecB2A$_q$ [SPOG19]
Decapsulation: Decrypt and Re-encrypt

\[ s = \frac{A}{U} \times \left( 0 \times 0 + D \times D \times c_1 - c_2 \right) \]

\[ D = H(K', c) \]

\[ K = H(z, c) \]
Polynomial Arithmetic

\[
D \times s - D \times D \times c_1 + D \times s_0 \times b + D \times \mathcal{H}(\mathcal{H}(pk), c) = H(K_0, c)
\]

\[
D \times \mathcal{H}(z, c) = H(z, c)
\]
Polynomial Arithmetic

Easy to protect using arithmetic masking

Small overhead factors

- $(n = 2) : 1.7^* \ [\text{BGR}^{+21}] - 2.0^\dagger \ [\text{BDK}^{+21}]$
- $(n = 3) : 2.96^* \ [\text{BGR}^{+21}]$

* with amortized precomputation
† w/o amortized precomputation, precomputation possible using techniques from [MKV20] or [CHK+21]
SHA-3

\[ K = \mathcal{H}(K', c) \]

\[ K = \mathcal{H}(z, c) \]
SHA-3

Typically protected using Boolean masking [BDPVA10, BBD+16]

Overhead factors

- $n = 2$: 5.9* [BGR+21] - 9.26† [BDK+21]
- $n = 3$: 73.1* [BGR+21]

* w.r.t plain-C
† w.r.t optimized assembly
Binomial Sampling

\[ K = \mathcal{H}(K', c) \]

\[ \tilde{K} = \mathcal{H}(z, c) \]
Binomial Sampling

Add/Sub $2\mu$ Boolean masked bits

- $y = \text{BitAddSub}(\oplus)$

- Naive approach needs $2\mu$ B2A conversions
  - $y = \text{BitAddSub}$
Binomial Sampling

Add/Sub $2\mu$ Boolean masked bits

$y = \text{BitAddSub}(\oplus \quad \quad \quad \quad \quad \quad\quad )$

- Naive approach needs $2\mu$ B2A conversions
  - $y = \text{BitAddSub}(\quad \quad \quad \quad \quad \quad\quad )$

- Use masked half-adders [SPOG19]
  - $y = \text{B2A}(\text{SecBitAddSub}(\quad \quad \quad \quad \quad \quad\quad ))$
## MLWE vs MLWR in Masking

<table>
<thead>
<tr>
<th>Scheme</th>
<th># Keccak-f</th>
<th># poly</th>
<th>SecBitAddSub</th>
<th>B2A</th>
</tr>
</thead>
<tbody>
<tr>
<td>{Light../Fire}Saber</td>
<td>4/5/5</td>
<td>(l)</td>
<td>(\mu = {5/4/3})</td>
<td>(2^k)</td>
</tr>
<tr>
<td>Kyber{512/768/1024}</td>
<td>7/7/9</td>
<td>(2l + 1)</td>
<td>(\mu = {3/2/2})</td>
<td>(q)</td>
</tr>
</tbody>
</table>
MLWE vs MLWR in Masking

<table>
<thead>
<tr>
<th>XOF</th>
<th>CBD(\mu)</th>
</tr>
</thead>
<tbody>
<tr>
<td># Keccak-f</td>
<td># poly</td>
</tr>
<tr>
<td>{Light/Fire} Saber</td>
<td>4/5/5</td>
</tr>
<tr>
<td>Kyber{512/768/1024}</td>
<td>7/7/9</td>
</tr>
</tbody>
</table>

ARM Cortex-M4 Keccak-f PolySecBitAddSub PolyB2A Total

<table>
<thead>
<tr>
<th>Saber [BDK(^+21)]</th>
<th>5 \times 123k</th>
<th>3 \times 50k</th>
<th>3 \times 17k</th>
<th>815k (1.00x)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kyber768</td>
<td>7 \times 123k</td>
<td>7 \times 32k</td>
<td>7 \times 118k(^\dagger)</td>
<td>1914k (2.35x)</td>
</tr>
</tbody>
</table>

\(^\dagger\) we use SecB2A\(_q\) [SPOG19] for this experiment, more efficient than SecAdd-B2A\(_q\) in software
\[ q(x, d) = b \left( \frac{2d}{q} \right) \cdot x \mod 2 \]

Interval comparison with 2 intervals:

\[ I_1 = 0 \quad I_2 = 3 \quad I_3 = 2 \quad I_4 = 1 \]

Compress\(_q\)

\[ \text{XOF} \]

\[ \text{CBD}_\mu \]

\[ K = \mathcal{H}(K', c) \]

\[ K = \mathcal{H}(z, c) \]
$y = \text{Compress}_q'(x, d) = \lfloor (2^d/q) \cdot x \rfloor \mod 2^d$

Interval comparison with $2^d$ intervals

- $2^2$ intervals on the right
Compress\(2^k\) - Saber

\[ y = \text{Compress}'_{2^k}(x, d) = (2^d/2^k) \cdot x \mod 2^d \]

\[ x = \underbrace{x_{\{msb\}}}_\bullet \underbrace{x_{\{lsb\}}}_d \quad (k - d) \]

\[ y = x_{\{msb\}} \]
MaskedCompress\(_{2^k}\) - Saber

\[ y = \text{Compress}'_{2^k}(x, d) = \left[ \frac{2^d}{2^k} \cdot x \right] \mod 2^d \]

\[ x = x_{\text{msb}} \cdot x_{\text{lsb}}, \quad y = x_{\text{msb}} \]
MaskedCompress\(_{2^k}\) - Saber

\[ y = \text{Compress}'_{2^k}(x, d) = \lfloor (2^d/2^k) \cdot x \rfloor \mod 2^d \]

\[ x = \begin{array}{c}
\{\text{msb}\} \\
\uparrow \\
\{\text{lsb}\}
\end{array} \quad \cdot \quad \begin{array}{c}
\{\text{msb}\} \\
\uparrow \\
\{\text{lsb}\}
\end{array}, \quad y = x_{\{\text{msb}\}} \]

\[ x = \begin{array}{c}
\text{\textcolor{orange}{msb}} \\
\uparrow \\
\text{\textcolor{orange}{lsb}}
\end{array} \quad \cdot \quad \begin{array}{c}
\text{\textcolor{orange}{msb}} \\
\uparrow \\
\text{\textcolor{orange}{lsb}}
\end{array}, \quad \text{Carry} \]

\[ x = \begin{array}{c}
\textcolor{orange}{\text{msb}} \\
\uparrow \\
\textcolor{orange}{\text{lsb}}
\end{array} \quad \cdot \quad \begin{array}{c}
\textcolor{orange}{\text{msb}} \\
\uparrow \\
\textcolor{orange}{\text{lsb}}
\end{array} \]
$y = \text{Compress}'_{2^k}(x, d) = \left[\frac{2^d}{2^k}\right] \cdot x \mod 2^d$

$\downarrow x = \begin{array}{c} \{\text{msb}\} \\
\end{array} \rightarrow d \rightarrow \begin{array}{c} \{\text{lsb}\} \\
\end{array} \quad \cdots \quad \begin{array}{c} \{\text{msb}\} \\
\end{array} = y$

SecCarry

$\downarrow x = \begin{array}{c} \{\text{msb}\} \\
\end{array} \rightarrow \cdots \rightarrow \begin{array}{c} \{\text{lsb}\} \\
\end{array} \quad \cdots \quad \begin{array}{c} \{\text{msb}\} \\
\end{array} = y + \text{SecCarry}(x_{\{\text{lsb}\}})$

$\uparrow \text{SecCarry is a pruned A2B conversion}$

$\uparrow$[BDK$^+$21]
MaskedDecode ≡ MaskedCompress_\(q(x, 1)\) - Kyber

[OSPG18]: \(\text{Transform}_{2^k}\) and \(A2B_{2^k}\)

[FBR+21]: \(A2B_q\) and \(\text{SecAdd}(-\frac{q}{2})\)

[BGR+21]: \(A2B_q\) and \(\text{BitSliceSecSearch}\)

- \(\text{SecSearch} [BGR+21] \equiv \text{MSB}(\text{SecConstAdd}(x, -\frac{q}{2}))\) [FBR+21]
MaskedCompress\(_q(x, 2)\)? - Kyber

[OSPG18]: Transform\(_{2^k}\) and A2B\(_{2^k}\)

[FBR\(^+\)21]: A2B\(_q\) and SecAdd\((-\frac{q}{2})\)

[BGR\(^+\)21]: A2B\(_q\) and BitSliceSecSearch

\(2^{12} - \frac{q}{4}\) and \(\frac{q}{4}\) no longer spaced at bit-intervals
MaskedCompress_{q}(x, d) - Kyber

\[ y = \text{Compress}'_{q}(x, d) = \lfloor x' \rfloor \mod 2^d, \ x' = \left(2^d/q\right) \cdot x \]

Diagram:

- \( x' = x'_{\{msb\}} \) and \( x'_{\{lsb\}} \)
- Carry
- \( x' = \) 
- ▶️
- \( x = x_{\{msb\}} \) and \( x_{\{lsb\}} \)
\( y = \text{Compress}'_q(x, d) = [x'] \mod 2^d, \quad x' = (2^d/q) \cdot x \)

\( \Downarrow \quad x' = \underbrace{\underbrace{x'_{\text{msb}}}}_{d} \cdot \underbrace{x'_{\text{lsb}_1}}_{f} \cdot \underbrace{x'_{\text{lsb}_2}}_{\infty} \)

\( \Rightarrow \) Only need \( f \) fractional bits \( x'_{\text{lsb}_1} \) to determine carry\(^\dagger\) \([\text{FBR}^+21]\)

\( \dagger \) \( f \) increases (logarithmically) with the number of shares
\( y = \text{Compress}^\prime_q(x, d) = \lfloor x' \rfloor \mod 2^d, \quad x' = \left( \frac{2^d}{q} \right) \cdot x \)

\( x' = \underbrace{x'_{\{msb\}}}_{d} \cdot \underbrace{x'_{\{lsb\}}}_{f} \cdot \underbrace{x'_{\{lsb\}}}_{\infty} \)

- Only need \( f \) fractional bits \( x'_{\{lsb\}} \) to determine carry\(^\dagger\) [FBR\textsuperscript{+}21]
  - Since \( x'_{\{lsb\}} = \left(2^d \cdot x \mod q \right)/q \) takes only \( q \) discrete values

\(^\dagger f \) increases (logarithmically) with the number of shares
\[
\text{MaskedCompress}(x, d)
\]

\[x' = (2^d / 2^k) \cdot x = \underbrace{\text{SecCarry} x'_{\text{msb}}}_{\text{SecCarry}} \cdot \underbrace{x'_{\text{lsb1}}}_{(k - d) \in \{3, 6, 9\}} \cdot \underbrace{x'_{\text{lsb2}}}_{d \cdot f \cdot \infty},
\]

\[\text{Saber}
\]

\[x' = (2^d / 2^k) \cdot x = \underbrace{\text{SecCarry} x'_{\text{msb}}}_{\text{SecCarry}} \cdot \underbrace{x'_{\text{lsb1}}}_{(k - d) \in \{3, 6, 9\}} \cdot \underbrace{x'_{\text{lsb2}}}_{d \cdot f},
\]

\[\text{Kyber}
\]

\[x' = (2^d / q) \cdot x = \underbrace{\text{SecCarry} x'_{\text{msb}}}_{\text{SecCarry}} \cdot \underbrace{x'_{\text{lsb1}}}_{d} \cdot \underbrace{x'_{\text{lsb2}}}_{f = 13},
\]

\[\underbrace{\text{SecCarry} x'_{\text{msb}}}_{\text{SecCarry}} \cdot \underbrace{x'_{\text{lsb1}}}_{d} \cdot \underbrace{x'_{\text{lsb2}}}_{f = 13},
\]
MaskedCompress\((x, d)\)

<table>
<thead>
<tr>
<th>ARM Cortex-M4 cycles ((n = 2))</th>
<th>PolyMaskedCompress</th>
</tr>
</thead>
<tbody>
<tr>
<td>((k - d) = 3)</td>
<td>(3 \times 14.5k)</td>
</tr>
<tr>
<td>((k - d) = 6)</td>
<td>(17k)</td>
</tr>
<tr>
<td>((k - d) = 9)</td>
<td>(19k)</td>
</tr>
<tr>
<td>(f = 13)</td>
<td>(79.5k(1.00x))</td>
</tr>
</tbody>
</table>

\[†\] unoptimized reference implementation

\[\text{Saber}\]

\[\text{Kyber}\]
I. HashComparison [OSPG18] with $fx_{BDK+21, BDH+21}$.

II. MaskedSum [BPO+20] with ReduceComparisons $fx_{BDH+21}$.

- Not a full comparison
- Doesn't work with compression

III. DecompressedComparison [BGR+21]

- One of the motivations: no existing MaskedCompress

IV. Interesting to consolidate approaches

\[
\begin{align*}
K &= \mathcal{H}(K', c) \\
K &= \mathcal{H}(z, c)
\end{align*}
\]
MaskedComparison

\[ n = 2: \]
- HashComparison \([\text{OSPG18}]\) with fix \([\text{BDK}^+21, \text{BDH}^+21]\).

\[ n > 2: \]
- MaskedSum \([\text{BPO}^+20]\) with ReduceComparisons fix \([\text{BDH}^+21]\)

- DecompressedComparison \([\text{BGR}^+21]\)
MaskedComparison

\[ n = 2: \]
- HashComparison [OSPG18] with fix [BDK+21, BDH+21].

\[ n > 2: \]
- MaskedSum [BPO+20] with ReduceComparisons fix [BDH+21]
  - Not a full comparison
  - Doesn't work with compression
- DecompressedComparison [BGR+21]
  - One of the motivations: no existing MaskedCompress
- Interesting to consolidate approaches
Results

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Device</th>
<th>Decapsulation unmasked</th>
<th>Decapsulation masked</th>
</tr>
</thead>
<tbody>
<tr>
<td>Saber [BDK+21]</td>
<td>ARM M4</td>
<td>1,123,280</td>
<td>2,833,348 (×2.52)</td>
</tr>
<tr>
<td>Kyber [BGR+21]</td>
<td>ARM M0+</td>
<td>5,530,000</td>
<td>12,208,000 (×2.21)*</td>
</tr>
<tr>
<td>Saber [FBR+21]</td>
<td>RISC-V</td>
<td>347,323</td>
<td>914,925 (×2.63)</td>
</tr>
<tr>
<td>Kyber [FBR+21]</td>
<td>RISC-V</td>
<td>338,746</td>
<td>1,402,650 (×4.14)†</td>
</tr>
</tbody>
</table>

Future work

- More efficient higher-order methods
- Saber on M0+, Kyber on M4 for a better comparison

*randomness sampling not included
†randomness sampling included: 167k cycles (17.5x Saber due to more random bits and rejection sampling)
Gilles Barthe, Sonia Belaïd, François Dupressoir, Pierre-Alain Fouque, Benjamin Grégoire, Pierre-Yves Strub, and Rébecca Zucchini.

Strong non-interference and type-directed higher-order masking.

Shivam Bhasin, Jan-Pieter D’Anvers, Daniel Heinz, Thomas Pöppelmann, and Michiel Van Beirendonck.

Attacking and defending masked polynomial comparison for lattice-based cryptography.

Michiel Van Beirendonck, Jan-Pieter D’anvers, Angshuman Karmakar, Josep Balasch, and Ingrid Verbauwhede.

A side-channel-resistant implementation of saber.

Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche.

Building power analysis resistant implementations of Keccak.
Joppe W. Bos, Marc Gourjon, Joost Renes, Tobias Schneider, and Christine van Vredendaal.  
Masking kyber: First- and higher-order implementations.  

Florian Bache, Clara Paglialonga, Tobias Oder, Tobias Schneider, and Tim Güneysu.  
High-speed masking for polynomial comparison in lattice-based KEMs.  

Jean-Sébastien Coron, Johann Großschädl, and Praveen Kumar Vadnala.  
Secure conversion between Boolean and arithmetic masking of any order.  
In International Workshop on Cryptographic Hardware and Embedded Systems, pages 188–205.  

Chi-Ming Marvin Chung, Vincent Hwang, Matthias J Kannwischer, Gregor Seiler, Cheng-Jhih Shih,  
and Bo-Yin Yang.  
NTT multiplication for NTT-unfriendly rings.  

Blandine Debraize.  
Efficient and provably secure methods for switching from arithmetic to Boolean masking.  
In International Workshop on Cryptographic Hardware and Embedded Systems, pages 107–121.  
Masked accelerators and instruction set extensions for post-quantum cryptography.

Louis Goubin.
A sound method for switching between Boolean and arithmetic masking.

Jose Maria Bermudo Mera, Angshuman Karmakar, and Ingrid Verbauwhede.
Time-memory trade-off in Toom-Cook multiplication: an application to module-lattice based cryptography.

Tobias Oder, Tobias Schneider, Thomas Pöppelmann, and Tim Güneysu.
Practical CCA2-secure and masked ring-LWE implementation.

Tobias Schneider, Clara Paglialonga, Tobias Oder, and Tim Güneysu.
Efficiently masking binomial sampling at arbitrary orders for lattice-based crypto.