CRYSTALS-Dilithium: A Lattice-Based Digital Signature Scheme

Léo Ducas (CWI), Eike Kiltz (Ruhr-Universität Bochum), Tancrède Lepoint (SRI International), Vadim Lyubashevsky (IBM Research), Peter Schwabe (Radboud University), Gregor Seiler (IBM Research), Damien Stehlé (ENS de Lyon)

September 10, 2018
Signature scheme submitted to the NIST PQC standardization process
Overview

- Signature scheme submitted to the NIST PQC standardization process
  - One out of 5 lattice-based signature schemes
Summary

- Signature scheme submitted to the NIST PQC standardization process
  - One out of 5 lattice-based signature schemes
- Public key size 1.5 KB, signature size 2.7 KB (recommended parameters)
Overview

- Signature scheme submitted to the NIST PQC standardization process
  - One out of 5 lattice-based signature schemes
- Public key size 1.5 KB, signature size 2.7 KB (recommended parameters)
- Design based on “Fiat-Shamir with Aborts” technique [Lyu09]
Overview

- Signature scheme submitted to the NIST PQC standardization process
  - One out of 5 lattice-based signature schemes
- Public key size 1.5 KB, signature size 2.7 KB (recommended parameters)
- Design based on “Fiat-Shamir with Aborts” technique [Lyu09]
  - Rejection sampling is used to sample signatures that do not reveal secret information
Signature scheme submitted to the NIST PQC standardization process
  - One out of 5 lattice-based signature schemes
Public key size 1.5 KB, signature size 2.7 KB (recommended parameters)
Design based on “Fiat-Shamir with Aborts” technique [Lyu09]
  - Rejection sampling is used to sample signatures that do not reveal secret information
Signature compression as developed in [GLP12], [BG14] (> 50% smaller)
Overview

- Signature scheme submitted to the NIST PQC standardization process
  - One out of 5 lattice-based signature schemes
- Public key size 1.5 KB, signature size 2.7 KB (recommended parameters)
- Design based on “Fiat-Shamir with Aborts” technique [Lyu09]
  - Rejection sampling is used to sample signatures that do not reveal secret information
- Signature compression as developed in [GLP12], [BG14] (> 50% smaller)
- New: Compression of public key (60% smaller, 100 byte larger signature)
Signature scheme submitted to the NIST PQC standardization process
  - One out of 5 lattice-based signature schemes
Public key size 1.5 KB, signature size 2.7 KB (recommended parameters)
Design based on “Fiat-Shamir with Aborts” technique [Lyu09]
  - Rejection sampling is used to sample signatures that do not reveal secret information
Signature compression as developed in [GLP12], [BG14] (> 50% smaller)
New: Compression of public key (60% smaller, 100 byte larger signature)
New: Hardness based on Module-LWE/SIS
Signature scheme submitted to the NIST PQC standardization process
  - One out of 5 lattice-based signature schemes
Public key size 1.5 KB, signature size 2.7 KB (recommended parameters)
Design based on "Fiat-Shamir with Aborts" technique [Lyu09]
  - Rejection sampling is used to sample signatures that do not reveal secret information
Signature compression as developed in [GLP12], [BG14] (> 50% smaller)
New: Compression of public key (60% smaller, 100 byte larger signature)
New: Hardness based on Module-LWE/SIS
New: Very efficient implementation
Principal Design Considerations

- Easy to implement securely – No Gaussian sampling
- Small total size of public key + signature
  - Among the smallest total size of all NIST submissions (Falcon is smaller)
- Conservative parameter selection
- Modular design
  - Use of Module-LWE/SIS allows to work over the same small ring for all security levels:
    Arithmetic needs only be optimized once and for all
Choice of Ring

Strategy: Choose smallest ring dimension $n$ that gives main advantages of Ring-LWE.
Choice of Ring

Strategy: Choose smallest ring dimension $n$ that gives main advantages of Ring-LWE

Dimension $n = 256$ is enough to get sufficiently large set of small norm challenges

Fully splitting prime $q$ allows for NTT-based multiplication (more about this later)

$$R = \mathbb{Z}_{2^{23}-2^{13}+1}[X]/(X^{256} + 1)$$
### Simplified Scheme

<table>
<thead>
<tr>
<th>Key generation:</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mathbf{A} \leftarrow R^{5 \times 4}$</td>
</tr>
<tr>
<td>$s_1 \leftarrow S_5^4$, $s_2 \leftarrow S_5^5$</td>
</tr>
<tr>
<td>$\mathbf{t} = \mathbf{A}s_1 + s_2$</td>
</tr>
<tr>
<td>$pk = (\mathbf{A}, \mathbf{t})$, $sk = (\mathbf{A}, \mathbf{t}, s_1, s_2)$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Verification:</th>
</tr>
</thead>
<tbody>
<tr>
<td>$c' = H(\text{High}(\mathbf{A}z - ct), M)$</td>
</tr>
<tr>
<td>If $|z|_\infty \leq \gamma - \beta$ and $c' = c$, accept</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Signing:</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mathbf{y} \leftarrow S_\gamma^4$</td>
</tr>
<tr>
<td>$\mathbf{w} = \mathbf{Ay}$</td>
</tr>
<tr>
<td>$c = H(\text{High}(\mathbf{w}), M) \in B_{60}$</td>
</tr>
<tr>
<td>$z = \mathbf{y} + cs_1$</td>
</tr>
<tr>
<td>If $|z|<em>\infty &gt; \gamma - \beta$ or $|\text{Low}(\mathbf{w} - cs_2)|</em>\infty &gt; \gamma - \beta$, restart</td>
</tr>
<tr>
<td>$\text{sig} = (z, c)$</td>
</tr>
</tbody>
</table>
Public Key Compression

Verification:

\[ c' = H(\text{High}(Az - ct), M) \]

If \( \|z\|_\infty \leq \gamma - \beta \) and \( c' = c \), accept

Decompose \( t = t_12^{14} + t_0 \) and put only \( t_1 \) into public key (23 → 9 bits per coefficient)
Public Key Compression

Verification:

\[ c' = H(\text{High}(Az - ct), M) \]

If \( \|z\|_{\infty} \leq \gamma - \beta \) and \( c' = c \), accept

Decompose \( t = t_12^{14} + t_0 \) and put only \( t_1 \) into public key \((23 \rightarrow 9 \text{ bits per coefficient})\)

For verification we need to compute

\[ \text{High}(Az - ct) = \text{High}(Az - ct_12^{14} - ct_0) \]

Include carries from adding \(-ct_0\) in signature \(\rightarrow \text{High}(Az - ct_12^{14})\) can be corrected
Tight reduction, even in quantum random oracle model, from *SelfTargetMSIS* and Module-LWE/SIS [KLS18]:

\[
\text{Adv}^{\text{SUFCMA}}(A) \leq \text{Adv}^{\text{MLWE}}(B) + \text{Adv}^{\text{SelfTargetMSIS}}(C) + \text{Adv}^{\text{MSIS}}(D) + 2^{-254}
\]

Given matrix \(A\), find short vector \(y\), challenge polynomial \(c\) and message \(M\) such that

\[
H \left( (I \mid A) \begin{pmatrix} y \\ c \end{pmatrix}, M \right) = c
\]

*SelfTargetMSIS* has non-tight reduction with standard forking lemma argument from Module-SIS
Implementation

Reference and AVX2 optimized implementations on

https://github.com/pq-crystals/dilithium

Main Operations:

- Polynomial multiplication in fixed ring $R = \mathbb{Z}_{2^{23}-2^{13}+1}[X](X^{256} + 1)$
- Expansion of the SHAKE XOF
  - Independent sampling of polynomials: Allows for parallel use of SHAKE
Constant Time

Our implementations are fully protected against timing side channel attacks.

In particular: No use of the C ’%’-operator.

*Note:* Sampling of challenge polynomials is not constant-time and does not need to be.
## Speed of Reference Implementation

<table>
<thead>
<tr>
<th></th>
<th>Key generation</th>
<th>Signing</th>
<th>Signing (average)</th>
<th>Verification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multiplication</td>
<td>89,591</td>
<td>987,666</td>
<td>1,280,053</td>
<td>143,924</td>
</tr>
<tr>
<td>SHAKE</td>
<td>178,487</td>
<td>314,570</td>
<td>377,068</td>
<td>161,079</td>
</tr>
<tr>
<td>Modular Reduction</td>
<td>11,944</td>
<td>120,793</td>
<td>163,017</td>
<td>10,626</td>
</tr>
<tr>
<td>Rounding</td>
<td>6,586</td>
<td>108,412</td>
<td>137,324</td>
<td>11,821</td>
</tr>
<tr>
<td>Rejection Sampling</td>
<td>60,740</td>
<td>76,893</td>
<td>94,607</td>
<td>28,082</td>
</tr>
<tr>
<td>Addition</td>
<td>8,008</td>
<td>58,696</td>
<td>79,498</td>
<td>10,723</td>
</tr>
<tr>
<td>Packing</td>
<td>7,114</td>
<td>17,183</td>
<td>18,856</td>
<td>8,883</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>381,178</strong></td>
<td><strong>1,778,148</strong></td>
<td><strong>2,260,429</strong></td>
<td><strong>396,043</strong></td>
</tr>
</tbody>
</table>

Median cycles of 5000 executions on Intel Skylake i7-6600U processor
NTT-based multiplication allows for easy reuse of computation:

- In Dilithium on average about 224 multiplications to sign a message
Advantages of NTT Multiplication

NTT-based multiplication allows for easy reuse of computation:

- In Dilithium on average about 224 multiplications to sign a message
- So, naively, 673 NTTs
Advantages of NTT Multiplication

NTT-based multiplication allows for easy reuse of computation:

- In Dilithium on average about 224 multiplications to sign a message
- So, naively, 673 NTTs
- But we only actually perform 172 NTTs
Advantages of NTT Multiplication

NTT-based multiplication allows for easy reuse of computation:

- In Dilithium on average about 224 multiplications to sign a message
- So, naively, 673 NTTs
- But we only actually perform 172 NTTs
Advantages of NTT Multiplication

NTT-based multiplication allows for easy reuse of computation:

- In Dilithium on average about 224 multiplications to sign a message
- So, naively, 673 NTTs
- But we only actually perform 172 NTTs

We immediately get a 4x speed-up in multiplication time from saving NTTs compared to Karatsuba multiplication

*Note:* In our reference implementation NTTs still make up for the most time consuming operation
AVX2 optimized Implementation

Optimizations:

- Vectorized NTT in assembly
- 4-way parallel SHAKE
- Better public key and signature compression
- Faster assembly modular reduction

About 3.5x faster signing compared to reference version

Recent update: >40% faster compared to TCHES paper
AVX2 optimized Implementation

Optimizations:

- Vectorized NTT in assembly
- 4-way parallel SHAKE
- Better public key and signature compression
- Faster assembly modular reduction

About 3.5x faster signing compared to reference version
AVX2 optimized Implementation

Optimizations:
- Vectorized NTT in assembly
- 4-way parallel SHAKE
- Better public key and signature compression
- Faster assembly modular reduction

About 3.5x faster signing compared to reference version

Recent update: > 40% faster compared to TCHES paper
New Fast Vectorized NTT Implementation

Prior state of the art: Double floating point arithmetic as in NewHope

Now: Fast approach with integer arithmetic and same Montgomery reduction strategy as in reference implementation
New Fast Vectorized NTT Implementation

Prior state of the art: Double floating point arithmetic as in NewHope

Now: Fast approach with integer arithmetic and same Montgomery reduction strategy as in reference implementation

Unfortunately not as fast as 16-bit NTT in Kyber because of missing instruction for high product
New Fast Vectorized NTT Implementation

Prior state of the art: Double floating point arithmetic as in NewHope

*Now:* Fast approach with integer arithmetic and same Montgomery reduction strategy as in reference implementation

Unfortunately not as fast as 16-bit NTT in Kyber because of missing instruction for high product

<table>
<thead>
<tr>
<th></th>
<th>Dilithium</th>
<th>Floating point</th>
<th>Kyber (16bit)</th>
<th>Saber (16bit)</th>
</tr>
</thead>
<tbody>
<tr>
<td>NTT</td>
<td>1,382</td>
<td>2,989</td>
<td>393</td>
<td>—</td>
</tr>
<tr>
<td>Inverse NTT</td>
<td>1,292</td>
<td>3,215</td>
<td>366</td>
<td>—</td>
</tr>
<tr>
<td>Full multiplication</td>
<td>4,288</td>
<td>10,042</td>
<td>1,162</td>
<td>3,810</td>
</tr>
</tbody>
</table>

Roughly 2x speed-up over floating point NTT
## Speed of AVX2 optimized Implementation

<table>
<thead>
<tr>
<th></th>
<th>Key generation</th>
<th>Signing</th>
<th>Signing (average)</th>
<th>Verification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multiplication</td>
<td>15,794</td>
<td>155,721</td>
<td>201,347</td>
<td>25,471</td>
</tr>
<tr>
<td>SHAKE</td>
<td>96,779</td>
<td>170,232</td>
<td>205,847</td>
<td>90,921</td>
</tr>
<tr>
<td>Modular reduction</td>
<td>1,034</td>
<td>7,902</td>
<td>10,541</td>
<td>708</td>
</tr>
<tr>
<td>Rounding</td>
<td>728</td>
<td>7,541</td>
<td>9,904</td>
<td>2,479</td>
</tr>
<tr>
<td>Rejection sampling</td>
<td>62,272</td>
<td>67,193</td>
<td>81,278</td>
<td>27,737</td>
</tr>
<tr>
<td>Addition</td>
<td>8,028</td>
<td>46,755</td>
<td>62,453</td>
<td>8,659</td>
</tr>
<tr>
<td>Packing</td>
<td>6,997</td>
<td>16,200</td>
<td>17,526</td>
<td>8,712</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>199,306</strong></td>
<td><strong>510,298</strong></td>
<td><strong>635,019</strong></td>
<td><strong>174,951</strong></td>
</tr>
</tbody>
</table>
Questions?
Module LWE (aka Generalized LWE)

Polynomial ring: \( R = \mathbb{Z}_q[X]/(X^n + 1) \)

It is hard to distinguish between uniform vector \( \mathbf{t} \in R^k \) and \( \mathbf{t} \) of the form

\[
\mathbf{t} = \begin{pmatrix}
t_1 \\
\vdots \\
t_k
\end{pmatrix} = \begin{pmatrix}
a_{1,1} & \cdots & a_{1,l} \\
\vdots & \ddots & \vdots \\
a_{k,1} & \cdots & a_{k,l}
\end{pmatrix} \begin{pmatrix}
s_{1,1} \\
\vdots \\
s_{1,l}
\end{pmatrix} + \begin{pmatrix}
s_{2,1} \\
\vdots \\
s_{2,k}
\end{pmatrix}
\]

Conservative parameters: Coefficients of \( s_{i,j} \) are from \( \{-5, \ldots, 5\} \)

- \( \mathbf{s}_1 \) lives in a module over \( R \) of rank \( l \)
- Ring-LWE is special case where \( l = 1 \) and \( \mathbf{s}_1 \) lies in the ring \( R \)
- Plain LWE is special case when the dimension \( n \) of the ring is 1 so that \( R = \mathbb{Z}_q \)
- Security: Effective dimension over \( \mathbb{Z}_q \) is \( l \cdot n \)
Suppose $\zeta \in \mathbb{Z}_q$ is a primitive 8-th root of unity, i.e. $\zeta^4 = -1$. 

\[ \mathbb{Z}_q[X]/(X^{256} + 1) \]

\[ \mathbb{Z}_q[X]/(X^{128} - \zeta^2) \]

\[ \mathbb{Z}_q[X]/(X^{64} - \zeta) \]

\[ \mathbb{Z}_q[X]/(X^{64} + \zeta) \]

\[ \mathbb{Z}_q[X]/(X^{128} + \zeta^2) \]

\[ \mathbb{Z}_q[X]/(X^{64} - \zeta^3) \]

\[ \mathbb{Z}_q[X]/(X^{64} + \zeta^3) \]
Advantages of NTT Multiplication

Consider the matrix-vector product

\[
\begin{pmatrix}
  w_1 \\
  w_2 \\
  w_3 \\
  w_4 \\
  w_5
\end{pmatrix}
= 
\begin{pmatrix}
  a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} \\
  a_{2,1} & a_{2,2} & a_{2,3} & a_{2,4} \\
  a_{3,1} & a_{3,2} & a_{3,3} & a_{3,4} \\
  a_{4,1} & a_{4,2} & a_{4,3} & a_{4,4} \\
  a_{5,1} & a_{5,2} & a_{5,3} & a_{5,4}
\end{pmatrix}
\begin{pmatrix}
  y_1 \\
  y_2 \\
  y_3 \\
  y_4
\end{pmatrix}
\]

This needs 20 multiplications or 60 NTTs for full NTT-based multiplications.

With NTT-based multiplication, the $a_{i,j}$ can be directly sampled in their NTT representation.

Also only one inverse NTT per row necessary.

We only need to compute 9 NTTs for the matrix-vector product.