HAETAE: Shorter Fiat-Shamir with Aborts Signature

Jung Hee Cheon¹,², Hyeongmin Choe¹, Julien Devevey³, Tim Güneysu⁴,⁵, Dongyeon Hong², Markus Krausz⁴, Georg Land⁴, Marc Möller⁴, Damien Stehlé², MinJune Yi¹

1. Seoul National University
2. CryptoLab Inc.
3. ANSSI
4. Ruhr University Bochum
5. German Research Centre for Artificial Intelligence
What’s Haetae?

- Website: https://kpqc.cryptolab.co.kr/haetae
- Submission to NIST’s additional PQC round
- Submission to South Korea PQC
- Same framework as Fiat-Shamir with Aborts over lattices
## High-level comparison with Dilithium

<table>
<thead>
<tr>
<th></th>
<th>Dilithium</th>
<th>Haetae</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Goal</strong></td>
<td>Implementer-friendly</td>
<td>Small signature size</td>
</tr>
<tr>
<td><strong>Distribution</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Mode</strong></td>
<td>Unimodal</td>
<td>Bimodal</td>
</tr>
<tr>
<td><strong>Arithmetic operations</strong></td>
<td>1.5</td>
<td>1</td>
</tr>
<tr>
<td><strong>Bit-size of the modulus</strong></td>
<td>23</td>
<td>16</td>
</tr>
<tr>
<td><strong>Number of repetitions</strong></td>
<td>4</td>
<td>6</td>
</tr>
</tbody>
</table>
Performances (Level II)

- Dilithium
- Falcon
- Haetae

Signature Size vs Median Runtime

- HAETAE
- J. Devevey (ANSSI)
Performances (Level II)

Median Runtime

- Dilithium
- Falcon
- Haetae

Implementation Easiness

- Dilithium
- Falcon
- Haetae

Signature Size

J. Devevey (ANSSI)
1. Fiat-Shamir with Aborts for Lattices
2. Hyperballs
   - Rejection Step
   - Hyperball Sampler
3. Minimizing $\|sc\|$
   - Key Generation
4. Signature Compression
5. Security Estimation
1. Fiat-Shamir with Aborts for Lattices
**Σ-Protocols**

\[ P(A, s) \quad \text{VERSUS} \quad V(A, t) \]

- \( \exists s = (A_0 | \text{Id})s = t \mod q \)

- Nothing is revealed on \( s \)

- Convincing \( V(A, t) \) without \( s \) is hard
Σ-Protocols

\[ P(A, s) \quad V(A, t) \]

\[ w \]

\[ c \]

\[ z \text{ or } \bot \]

- \( As = (A_0|\text{Id})s = t \mod q \)
- \( V \) accepts under some condition
- Nothing is revealed on \( s \)
- Convincing \( V \) without \( s \) is hard
Lyubashevsky’s Protocol [Lyu09, Lyu12]

\[ P(A, s) \quad V(A, t) \]

\[ y \leftarrow Q \]

\[ w = Ay \mod q \]

\[ c \]

\[ z = y + sc \text{ w.p. } \frac{P(z)}{MQ(y)} \]

\[ \text{Else } \perp \]

- \( As = (A_0|Id)s = t \mod q \)
- \( y, s \) and \( c \) are small
Lyubashevsky’s Protocol [Lyu09, Lyu12]

\[
P(A, s) \quad V(A, t)
\]

\[y \leftarrow Q\]

\[w = Ay \mod q\]

\[c\]

\[z = y + sc \text{ w.p. } \frac{P(z)}{MQ(y)}\]

Else ⊥

- \[As = (A_0|\text{Id})s = t \mod q\]
- \[y, s \text{ and } c \text{ are small}\]
- \[V \text{ accepts if } Az - tc = w \mod q \text{ and } \|z\| \leq \gamma\]
- \[z \leftarrow P \text{ independent of } s\]
- Convincing \(V\) without \(s\) is hard
Fiat-Shamir with Aborts

\[
\text{Sign}(A, s, \mu):
\]
\[
\text{do}
\]
\[
y \leftarrow Q
\]
\[
c = H(Ay \mod q, \mu)
\]
\[
z = y + sc
\]
\[
w.p. \quad P(z) \quad M \cdot Q(y)
\]
\[
z = \bot
\]
\[
\text{while } z = \bot
\]
\[
\text{return } (z, c)
\]

- Verification: recover \(w\), check if \(c = H(w, \mu)\) and \(\|z\| \leq \gamma\)
Fiat-Shamir with Aborts

\[ \text{Sign}(A, s, \mu) : \]
\[
\begin{align*}
&\text{do} \\
&\quad y \leftarrow Q \\
&\quad c = H(Ay \mod q, \mu) \\
&\quad z = y + sc \\
&\quad \text{w.p. } P(z) \frac{M}{M \cdot Q(y)} \\
&\quad \text{while } z = \perp \\
&\quad \text{return } (z, c)
\end{align*}
\]

- Verification: recover \( w \), check if \( c = H(w, \mu) \) and \( \|z\| \leq \gamma \)

- Unforgeability if [BBD+23]:
  - Large min-entropy for \( w \)
  - aHVZK: simulate accepting transcripts without \( s \)
  - Soundness: \( A(A, t) \) cannot convince \( V(A, t) \)
Security Reduction

Soundness

\[ \downarrow \text{(Only in the ROM)} \]

UF-NMA (Find a forgery given the verification key)

\[ \downarrow \text{(Use the HVZK simulator)} \]

UF-CMA (Find a forgery given \( vk \) and access to a signing oracle)
Optimal Choice of Distribution

Haetae instantiates $Q \propto P = U(\bullet)$
Optimal Choice of Distribution

Haetae instantiates $Q \propto P = U(\bullet)$

- Smallest $\gamma$ as with Gaussians [DFPS22]
- Easier rejection step than Gaussians
Optimal Choice of Distribution

Haetae instantiates $Q \propto P = U(\cdot)$

- Smallest $\gamma$ as with Gaussians [DFPS22]
- Easier rejection step than Gaussians

But we can do better!
Bimodal Lattice-based Fiat-Shamir with Aborts

<table>
<thead>
<tr>
<th>Sign($A_\mu$, $s$, $\mu$):</th>
</tr>
</thead>
<tbody>
<tr>
<td>do</td>
</tr>
<tr>
<td>$y \leftarrow Q$</td>
</tr>
<tr>
<td>$c = H(Ay \mod 2q, \mu)$</td>
</tr>
<tr>
<td>$z = y + (-1)^{u({{0,1}}})sc$</td>
</tr>
<tr>
<td>w.p. $M(Q(z-sc)+Q(z+sc))$</td>
</tr>
<tr>
<td>$z = \bot$</td>
</tr>
<tr>
<td>while $z = \bot$</td>
</tr>
<tr>
<td>return $(z, c)$</td>
</tr>
</tbody>
</table>

- New key equation: $A_\mu s = -A_\mu s = qcj \mod 2q$
Bimodal Lattice-based Fiat-Shamir with Aborts

**Sign(\(A, s, \mu\)):**

do
| \(y \leftarrow Q\)  |
| \(c = H(Ay \mod 2q, \mu)\) |
| \(z = y + (-1)^{U(\{0, 1\})}sc\) |
| w.p. \(\frac{2P(z)}{M(Q(z-sc)+Q(z+sc))}\) |
| \(z = \bot\) |
while \(z = \bot\)  
return \((z, c)\)

- New key equation: \(As = -As = qcj \mod 2q\)
- Verification:
  - Compute \(w = Az - qcj \mod 2q\)
  - Accept if \(\|z\| \leq \gamma\) and \(c = H(w, \mu)\)
Bimodal Lattice-based Fiat-Shamir with Aborts

\[
\text{Sign}(A, s, \mu): \\
\text{do} \\
y \leftarrow Q \\
c = H(Ay \mod 2q, \mu) \\
z = y + (-1)^{u(\{0,1\})} sc \\
\text{w.p. } \frac{2P(z)}{M(Q(z-sc) + Q(z+sc))} \\
z = \bot \\
\text{while } z = \bot \\
\text{return } (z, c)
\]

- New key equation: \( As = -As = qcj \mod 2q \)
- Verification:
  - Compute \( w = Az - qcj \mod 2q \)
  - Accept if \( \|z\| \leq \gamma \) and \( c = H(w, \mu) \)
- Allows for smaller \( \gamma \) at constant \( M \) [DFPS22]

\[ \gamma \approx \frac{\sqrt{\text{dim}(y)} \cdot \|sc\|}{\sqrt{\log M}} \]
Design Rationale

Smaller $\gamma$ (i.e. smaller size)  
\[\downarrow\]
Forging becomes harder  
\[\downarrow\]
More security overall  
\[\downarrow\]
Smaller parameters  
\[\downarrow\]
Smaller size (i.e. smaller $\gamma$)  
\[\downarrow\]
...

J. Devevey (ANSSI)
2. Hyperballs
2.1. Rejection Step

**KeyGen**(1^λ):

1: return $$\mathbf{A}, s$$ with $$\mathbf{A}s = qj \mod 2q$$

**Sign**(\(\mathbf{A}, s, \mu\)):

do

1: \(y \leftarrow U(\bullet)\)
2: \(w = Ay \mod 2q\)
3: \(c = H(HB(w), \text{LSB}(w), \mu)\)
4: \(z = y + (-1)^b sc\)
5: w.p. \(p(z)\), set \(z = \perp\)

while \(z = \perp\)

6: \(x = \text{compress}(z)\)
7: return \((x, c)\)
Rejection Probability

\[ \text{−SC} + \quad \text{SC} + \]

1
1/2
0
Rejection Probability

\[ s_c - s_c^1 = 2^0 \]

Check \( \|z\| \) and \( \|2z - y\| \)

- 1
- 1/2
- 0
2.2. Hyperball Sampler

**KeyGen(1^\lambda):**
1: return \( A, s \)
   with \( As = qj \mod 2q \)

**Sign(A, s, \mu):**

do
1: \( y \leftarrow U(\bullet) \)
2: \( w = Ay \mod 2q \)
3: \( c = H(HB(w), LSB(w), \mu) \)
4: \( z = y + (-1)^b sc \)
5: w.p. \( p(z) \), set \( z = \perp \)
while \( z = \perp \)
6: \( x = \text{compress}(z) \)
7: return \( (x, c) \)
Main Theorem

Back to normal distributions [VG17]

\[ \frac{n}{n+2} =_{D} U(\cdot) \]

- Works for continuous distributions
Main Theorem

Back to normal distributions [VG17]

- Works for continuous distributions
- Implemented using fixed-point arithmetic
- Requires ≈ 90 bits of precision
Implementation with Fixed-point Arithmetic

- (i) Discrete Gaussian to normal distribution “for free”
- (iii) Discretization step to balance rejection probability and efficiency
Setting the Discretization Step

\textit{Card}(\bigotimes) ?
Setting the Discretization Step

**Card(□)?**

- Counting the number of points would help setting parameters
- Well-known for *continuous* hyperballs
- Choose a step making the comparison meaningful
- **Other solution:** empirical approach
Performances

Hyperball Sampler

Sign

Up to 80% of signing runtime!
3. Minimizing $\|sc\|$
3.1. Key Generation

**KeyGen**(1^λ):
1: return A, s
   with $As = qj \mod 2q$

**Sign**(A, s, μ):
- do
  1: $y \leftarrow U(\bullet)$
  2: $w = Ay \mod 2q$
  3: $c = H(HB(w), LSB(w), μ)$
  4: $z = y + (-1)^b sc$
  5: w.p. $p(z)$, set $z = \perp$
  while $z = \perp$
  6: $x = \text{compress}(z)$
  7: return $(x, c)$
Key Generation

1: $A_0 \leftarrow U(R_q^{k \times \ell-1})$
2: $s_0, e_0 \leftarrow U([-\eta \ldots \eta])^{\ell-1+k}$
3: $b \leftarrow A_0 s_0 + e_0 \mod q$
Key Generation

1: \( \mathbf{A}_0 \leftarrow U(\mathcal{R}_{q}^{k \times \ell - 1}) \)
2: \( \mathbf{s}_0, \mathbf{e}_0 \leftarrow U([-\eta \ldots \eta])^{\ell - 1 + k} \)
3: \( \mathbf{b} \leftarrow \mathbf{A}_0 \mathbf{s}_0 + \mathbf{e}_0 \mod q \)
4: \( \mathbf{A} \leftarrow (-2\mathbf{b} + q\mathbf{j} | 2\mathbf{A}_0 | 2\mathbf{l}_k) \mod 2q \)
5: \( \mathbf{s} \leftarrow (1|\mathbf{s}_0^\top|\mathbf{e}_0^\top)^\top \)

\begin{itemize}
  \item \( \mathbf{j} = (1, 0 \ldots 0)^\top \)
  \item Add a trapdoor in the public matrix
\end{itemize}
Key Generation

1. \( A_0 \leftarrow U(\mathcal{R}_q^{k \times \ell - 1}) \)
2. \( s_0, e_0 \leftarrow U([-\eta \ldots \eta])^{\ell - 1 + k} \)
3. \( b \leftarrow A_0 s_0 + e_0 \mod q \)
4. \( A \leftarrow (-2b + qj|2A_0|2I_k) \mod 2q \)
5. \( s \leftarrow (1|s_0^T|e_0^T)^T \)
6. restart if \( f_{\tau}(s) > n\beta^2 / \tau \)
7. return \( vk = A, sk = s \)

- \( j = (1, 0 \ldots 0)^T \)
- Add a trapdoor in the public matrix
- \( f_{\tau} \) ensures that \( \|sc\| \leq \beta \) for any \( c \) with Hamming weight \( \leq \tau \)
- Acceptance rate from 10 to 25%
The $f_\tau$ Function

- Challenge $c$: binary polynomial with $\tau$ 1s
- $f_\tau$ uses canonical embedding to bound $\max_c \|sc\|
- Finer-grained than other upper bounds

<table>
<thead>
<tr>
<th>Challenge Entropy Level II $\tau$</th>
<th>Ternary $\left(\frac{256}{\tau}\right) + \tau$</th>
<th>Binary $\left(\frac{256}{\tau}\right)$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>39</td>
<td>58</td>
</tr>
</tbody>
</table>
Idea: transmit only $\mathbf{b} - \text{LeastSignificantBit}(\mathbf{b}) \Rightarrow$ saves 1 bit per coordinate
Idea: transmit only $b - \text{LeastSignificantBit}(b)$ ⇒ saves 1 bit per coordinate

Downside:

- Set $s^T = (1|s_0^T|e_0^T + \text{LSB}(b))$

- Adapt KeyGen to keep $A$ pseudo-uniform

- $\text{LSB}$ is modified to keep $s$ balanced
4. Signature Compression

**KeyGen**(1^λ):
1. return \( A, s \) with \( As = qj \mod 2q \)

**Sign**(\( A, s, \mu \)):
1. \( y \leftarrow U(\bullet) \)
2. \( w = Ay \mod 2q \)
3. \( c = H(HB(w), LSB(w), \mu) \)
4. \( z = y + (-1)^bsc \)
5. w.p. \( p(z) \), set \( z = \perp \)
6. while \( z = \perp \), break
7. \( x = \text{compress}(z) \)
8. return \((x, c)\)
Low Bits Truncation

- Truncation technique from Bai and Galbraith

- \( Ay = A_1 z_1 + 2z_2 - q cj \mod 2q \) for some \( A_1 \)

- Exclude \( LB(z_2) \) from the signature

- Hash \( HB(w) \) and \( LSB(w) \)
Transmitting the Signature

- Signature encoded using tANS (similar to [ETWY22])
- Low bits are sent as they are for reduced memory usage
- Allows for a signature size close to its entropy

Empirical encoding

Empirical encoding

Incompressible

Not transmitted

\[ HB(z_1) \quad LB(z_1) \quad z_1 \]

\[ HB(z_2) \quad LB(z_2) \quad z_2 \]
5. Security Estimation
Security Assumptions

MSIS (find a kernel element of $A$ with norm $\leq \gamma$)

\[ \Downarrow (in \ the \ ROM) \]

“BimodalSelfTargetMSIS” (MSIS with hash collision)

\[ \Downarrow (with \ MLWE) \]

Unforgeability of Haetae
Security Assumptions

**MSIS** *(find a kernel element of $A$ with norm $\leq \gamma$)*

\[\downarrow (\text{in the ROM})\]

*“BimodalSelfTargetMSIS” (MSIS with hash collision)*

\[\downarrow (\text{with MLWE})\]

Unforgeability of Haetae
• Best approach for BimodalSelfTargetMSIS: solve MSIS

• MSIS and MLWE security estimated via the CoreSVP approach

• MLWE has refined estimates using [DSDGR20]

• This follows Dilithium’s approach for easy comparison
Wrapping up

**Median Runtime**

- **Signature Size**
  - 0
  - 2M
  - 4M
  - 6M
  - 8M
  - 10M
  - 12M
  - 14M
  - 16M
  - 18M
  - 20M

**Signature Size**

- 500
- 1000
- 1500
- 2000
- 2500
- 3000

**Implementation Easiness**

- **Dilithium**
- **Falcon**
- **Haetae**

Any question?