FrodoKEM

Halil İbrahim Kaplan

TDBY / Kripto Analiz Laboratuvarı

halil.kaplan@tubitak.gov.tr

2022

Overview

1. **Description of Functions**
2. **FrodoPKE**
3. **FrodoKEM**
4. **Parameters**

Notation. The algorithms in this document are described in terms of the following parameters:

- χ, a probability distribution on $\mathbb{Z}$;
- $q = 2^D$, a power-of-two integer modulus with exponent $D \leq 16$;
- $n, \overline{m}, \overline{n}$, integer matrix dimensions with $n \equiv 0 \pmod 8$;
- $B \leq D$, the number of bits encoded in each matrix entry;
- $\ell = B \cdot \overline{m} \cdot \overline{n}$, the length of bit strings that are encoded as $\overline{m}$-by-$\overline{n}$ matrices;
- $\mathsf{len}_{\mathsf{seed}_\mathbf{A}}$, the bit length of seeds used for pseudorandom matrix generation;
- $\mathsf{len}_{\mathsf{seed}_\mathbf{SE}}$, the bit length of seeds used for pseudorandom bit generation for error sampling.

Additional parameters for specific algorithms accompany the algorithm description.

1. Frodo.Encode – Frodo.Decode

bit strings → mod-q integer matrices

1010100…

$l = B * \bar{m} * \bar{n}$

$$\begin{bmatrix} 2 & \cdots & 3 \\ \vdots & \ddots & \vdots \\ 2 & \cdots & 5 \end{bmatrix}_{\bar{m}\ x\ \bar{n}}$$

Function ec (·) : Encodes an integer $0 \leq k < 2^B$ as an element in Z_q

$$ec(k) := k * q/2^B$$

1. Frodo.Encode – Frodo.Decode

Algorithm 1 Frodo.Encode

Input: Bit string $\mathbf{k} \in \{0,1\}^{\ell}$, $\ell = B \cdot \overline{m} \cdot \overline{n}$.

Output: Matrix $\mathbf{K} \in \mathbb{Z}_q^{\overline{m} \times \overline{n}}$.

1: **for** $(i = 0; i < \overline{m}; i \leftarrow i + 1)$ **do**
2: **for** $(j = 0; j < \overline{n}; j \leftarrow j + 1)$ **do**
3: $k \leftarrow \sum_{l=0}^{B-1} \mathbf{k}_{(i \cdot \overline{n} + j)B + l} \cdot 2^l$
4: $\mathbf{K}_{i,j} \leftarrow \mathrm{ec}(k) = k \cdot q/2^B$
5: **return** $\mathbf{K} = (\mathbf{K}_{i,j})_{0 \le i < \overline{m}, 0 \le j < \overline{n}}$

Apply ec (·) to B-bit sub-strings sequentially and fill the matrix row by row entry-wise

1. Frodo.Encode – Frodo.Decode

EX:

Let B $= 3,\ \bar{m} = 2,\ \bar{n} = 2,\ q = 16$

Frodo.Encode(101 110 001 111) = $\begin{bmatrix} ec(5) & ec(6) \\ ec(1) & ec(7) \end{bmatrix}$

$= \begin{bmatrix} 5 * \frac{16}{8} = 10 & 12 \\ 2 & 14 \end{bmatrix}$

1. Frodo.Encode – Frodo.Decode

Algorithm 2 Frodo.Decode

Input: Matrix $\mathbf{K} \in \mathbb{Z}_q^{\overline{m} \times \overline{n}}$.

Output: Bit string $\mathbf{k} \in \{0,1\}^{\ell}$, $\ell = B \cdot \overline{m} \cdot \overline{n}$.

1: **for** $(i = 0; i < \overline{m}; i \leftarrow i + 1)$ **do**
2: **for** $(j = 0; j < \overline{n}; j \leftarrow j + 1)$ **do**
3: $k \leftarrow \mathrm{dc}(\mathbf{K}_{i,j}) = \lfloor \mathbf{K}_{i,j} \cdot 2^B / q \rceil \bmod 2^B$
4: $k = \sum_{l=0}^{B-1} k_l \cdot 2^l$ where $k_l \in \{0,1\}$
5: **for** $(l = 0; l < B; l \leftarrow l + 1)$ **do**
6: $\mathbf{k}_{(i \cdot \overline{n} + j)B + l} \leftarrow k_l$
7: **return** $\mathbf{k}$

extracts B bits from each entry by applying the function dc(·)

$$\mathrm{dc}(c) = \lfloor c \cdot 2^B / q \rceil \bmod 2^B$$

2. Frodo.Pack – Frodo.Unpack

Matrix with entries in $\mathbb{Z}_q$ → Bit string

$$\begin{bmatrix} 2 & \cdots & 3 \\ \vdots & \ddots & \vdots \\ 2 & \cdots & 5 \end{bmatrix}$$

$n_1 x\, n_2$

1010100…

$D * n_1 * n_2$

Algorithm 3 Frodo.Pack

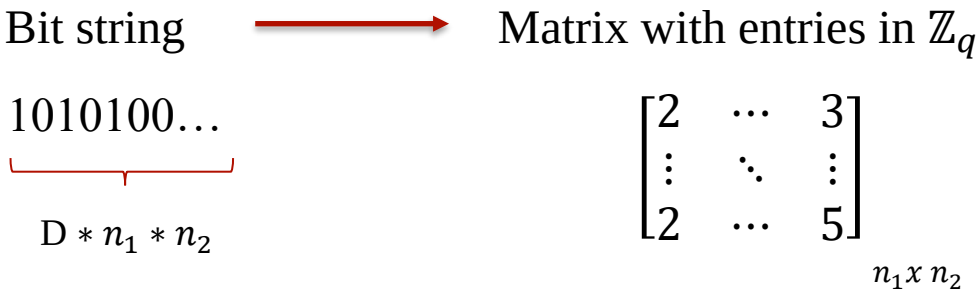
Input: Matrix $\mathbf{C} \in \mathbb{Z}_q^{n_1 \times n_2}$.

Output: Bit string $\mathbf{b} \in \{0,1\}^{D \cdot n_1 \cdot n_2}$.

1: **for** $(i = 0;\ i < n_1;\ i \leftarrow i + 1)$ **do**
2: **for** $(j = 0;\ j < n_2;\ j \leftarrow j + 1)$ **do**
3: $\mathbf{C}_{i,j} = \sum_{l=0}^{D-1} c_l \cdot 2^l$ where $c_l \in \{0,1\}$
4: **for** $(l = 0;\ l < D;\ l \leftarrow l + 1)$ **do**
5: $\mathbf{b}_{(i \cdot n_2 + j)D + l} \leftarrow c_{D-1-l}$
6: **return** $\mathbf{b}$

Packs a matrix into a bit string by simply concatenating the D-bit matrix coefficients

2. Frodo.Pack – Frodo.Unpack



Algorithm 4 Frodo.Unpack

Input: Bit string $\mathbf{b} \in \{0,1\}^{D \cdot n_1 \cdot n_2}$, n_1, n_2.
Output: Matrix $\mathbf{C} \in \mathbb{Z}_q^{n_1 \times n_2}$.

1: **for** $(i = 0; i < n_1; i \leftarrow i + 1)$ **do**
2: **for** $(j = 0; j < n_2; j \leftarrow j + 1)$ **do**
3: $\mathbf{C}_{i,j} \leftarrow \sum_{l=0}^{D-1} \mathbf{b}_{(i \cdot n_2 + j)D + l} \cdot 2^{D-1-l}$
4: **return** $\mathbf{C}$

3. Frodo.Sample

Random bit string, Distribution → Integer

Algorithm 5 Frodo.Sample

Input: A (random) bit string $\mathbf{r} = (\mathbf{r}_0, \mathbf{r}_1, \ldots, \mathbf{r}_{\mathsf{len}_X - 1}) \in \{0,1\}^{\mathsf{len}_X}$, the table $T_\chi = (T_\chi(0), T_\chi(1), \ldots, T_\chi(s))$.
Output: A sample $e \in \mathbb{Z}$.

1: $t \leftarrow \sum_{i=1}^{\mathsf{len}_X - 1} \mathbf{r}_i \cdot 2^{i-1}$
2: $e \leftarrow 0$
3: **for** $(z = 0; z < s; z \leftarrow z + 1)$ **do**
4: **if** $t > T_\chi(z)$ **then**
5: $e \leftarrow e + 1$
6: $e \leftarrow (-1)^{\mathbf{r}_0} \cdot e$
7: **return** e

- Compute t = $r_1 * 2^0 + r_2 * 2^1 + \cdots + r_{len_x - 1} * 2^{len_x - 1}$
- Set e=0
- For each element in table,
- if t > element, then increase e
- Return $(-1)^{r_0} * e$

4. Frodo.SampleMatrix

Random bit strings, Distribution → Matrix $\in \mathbb{Z}^{n_1 * n_2}$

Algorithm 6 Frodo.SampleMatrix

Input: A (random) bit string $(\mathbf{r}^{(0)}, \mathbf{r}^{(1)}, \ldots, \mathbf{r}^{(n_1 n_2 - 1)}) \in \{0,1\}^{n_1 n_2 \cdot \mathsf{len}_\chi}$ (here, each $\mathbf{r}^{(i)}$ is a vector of len_χ bits), the table T_χ.
Output: A sample $\mathbf{E} \in \mathbb{Z}^{n_1 \times n_2}$.

1: **for** $(i = 0; i < n_1; i \leftarrow i + 1)$ **do**
2: **for** $(j = 0; j < n_2; j \leftarrow j + 1)$ **do**
3: $\mathbf{E}_{i,j} \leftarrow$ Frodo.Sample$(\mathbf{r}^{(i \cdot n_2 + j)}, T_\chi)$
4: **return** $\mathbf{E}$

For each row in Matrix

- Fill each entity with Frodo.Sample()

5. Frodo.Gen

$seed_A$,
Dimention n → Matrix $\in \mathbb{Z}_q^{n*n}$

Algorithm 8 Frodo.Gen using SHAKE128

Input: Seed $\mathsf{seed}_{\mathbf{A}} \in \{0,1\}^{\mathsf{len}_{\mathsf{seed}_{\mathbf{A}}}}$.
Output: Pseudorandom matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times n}$.

1: **for** $(i = 0; i < n; i \leftarrow i + 1)$ **do**
2: $\quad \mathbf{b} \leftarrow \langle i \rangle \| \mathsf{seed}_{\mathbf{A}} \in \{0,1\}^{16+\mathsf{len}_{\mathsf{seed}_{\mathbf{A}}}}$ where $\langle i \rangle \in \{0,1\}^{16}$
3: $\quad \langle c_{i,0} \rangle \| \langle c_{i,1} \rangle \| \cdots \| \langle c_{i,n-1} \rangle \leftarrow \mathrm{SHAKE128}(\mathbf{b}, 16n)$ where each $\langle c_{i,j} \rangle \in \{0,1\}^{16}$
4: $\quad$ **for** $(j = 0; j < n; j \leftarrow j + 1)$ **do**
5: $\quad\quad \mathbf{A}_{i,j} \leftarrow c_{i,j} \bmod q$
6: **return** $\mathbf{A}$

For each row i of matrix

- Put i||seed inside of SHAKE
- Divide output of SHAKE to 16 bit pieces
- For each piece
  Convert to integer in mod q and put the empty leftmost part of row

5. Frodo.Gen

$seed_A$,
Dimention n → Matrix $\in \mathbb{Z}_q^{n*n}$

Algorithm 7 Frodo.Gen using AES128

Input: Seed $\mathsf{seed}_{\mathbf{A}} \in \{0,1\}^{\mathsf{len}_{\mathsf{seed}_{\mathbf{A}}}}$.
Output: Matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times n}$.

1: **for** $(i = 0; i < n; i \leftarrow i + 1)$ **do**
2: **for** $(j = 0; j < n; j \leftarrow j + 8)$ **do**
3: $\mathbf{b} \leftarrow \langle i \rangle \| \langle j \rangle \| 0 \cdots 0 \in \{0,1\}^{128}$ where $\langle i \rangle, \langle j \rangle \in \{0,1\}^{16}$
4: $\langle c_{i,j} \rangle \| \langle c_{i,j+1} \rangle \| \cdots \| \langle c_{i,j+7} \rangle \leftarrow \mathrm{AES128}_{\mathsf{seed}_{\mathbf{A}}}(\mathbf{b})$ where each $\langle c_{i,k} \rangle \in \{0,1\}^{16}$
5: **for** $(k = 0; k < 8; k \leftarrow k + 1)$ **do**
6: $\mathbf{A}_{i,j+k} \leftarrow c_{i,j+k} \bmod q$
7: **return** $\mathbf{A}$

FrodoPKE

Public-key encryption scheme with fixed-length message space, targeting IND-CPA security

IND-CPA: Indistinguishability under Chosen Plaintext Attack
The adversary generates two messages of equal length. The challenger decides, randomly, to encrypt one of them. The adversary tries to guess which of the messages was encrypted.
Algorithm: • Challenger generates K_E, K_D • Adversary generates two equal length messages m_0, m_1 also does additional operations(like encryption) in poly. time • Challenger randomly chooses b = 0 or b = 1 • Challenger sends to adversary C=E(K_E,m_b) • Adversary: perform additional operations in polynomial time including calls to the encryption oracle. Output guess. •If guess=b the adversary wins

Algorithm 9 FrodoPKE.KeyGen.

Input: None.

Output: Key pair $(pk, sk) \in (\{0,1\}^{\mathsf{len}_{\mathsf{seed}_\mathbf{A}}} \times \mathbb{Z}_q^{n \times \overline{n}}) \times \mathbb{Z}_q^{\overline{n} \times n}$.

1: Choose a uniformly random seed $\mathsf{seed}_\mathbf{A} \leftarrow\!\!\$\ U(\{0,1\}^{\mathsf{len}_{\mathsf{seed}_\mathbf{A}}})$
2: Generate the matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times n}$ via $\mathbf{A} \leftarrow \mathsf{Frodo.Gen}(\mathsf{seed}_\mathbf{A})$
3: Choose a uniformly random seed $\mathsf{seed}_{\mathbf{SE}} \leftarrow\!\!\$\ U(\{0,1\}^{\mathsf{len}_{\mathsf{seed}_{\mathbf{SE}}}})$
4: Generate pseudorandom bit string $(\mathbf{r}^{(0)}, \mathbf{r}^{(1)}, \ldots, \mathbf{r}^{(2n\overline{n}-1)}) \leftarrow \mathrm{SHAKE}(\mathtt{0x5F} \| \mathsf{seed}_{\mathbf{SE}}, 2n\overline{n} \cdot \mathsf{len}_\chi)$
5: Sample error matrix $\mathbf{S}^{\mathrm{T}} \leftarrow \mathsf{Frodo.SampleMatrix}((\mathbf{r}^{(0)}, \mathbf{r}^{(1)}, \ldots, \mathbf{r}^{(n\overline{n}-1)}), \overline{n}, n, T_\chi)$
6: Sample error matrix $\mathbf{E} \leftarrow \mathsf{Frodo.SampleMatrix}((\mathbf{r}^{(n\overline{n})}, \mathbf{r}^{(n\overline{n}+1)}, \ldots, \mathbf{r}^{(2n\overline{n}-1)}), n, \overline{n}, T_\chi)$
7: Compute $\mathbf{B} = \mathbf{AS} + \mathbf{E}$
8: **return** public key $pk \leftarrow (\mathsf{seed}_\mathbf{A}, \mathbf{B})$ and secret key $sk \leftarrow \mathbf{S}^{\mathrm{T}}$

Algorithm 10 FrodoPKE.Enc.

Input: Message $\mu \in \mathcal{M}$ and public key $pk = (\mathsf{seed}_{\mathbf{A}}, \mathbf{B}) \in \{0,1\}^{\mathsf{len}_{\mathsf{seed}_{\mathbf{A}}}} \times \mathbb{Z}_q^{n \times \overline{n}}$.
Output: Ciphertext $c = (\mathbf{C}_1, \mathbf{C}_2) \in \mathbb{Z}_q^{\overline{m} \times n} \times \mathbb{Z}_q^{\overline{m} \times \overline{n}}$.

1: Generate $\mathbf{A} \leftarrow \mathsf{Frodo.Gen}(\mathsf{seed}_{\mathbf{A}})$
2: Choose a uniformly random seed $\mathsf{seed}_{\mathbf{SE}} \leftarrow\!\!\$\ U(\{0,1\}^{\mathsf{len}_{\mathsf{seed}_{\mathbf{SE}}}})$
3: Generate pseudorandom bit string $(\mathbf{r}^{(0)}, \mathbf{r}^{(1)}, \ldots, \mathbf{r}^{(2\overline{m}n + \overline{m}\overline{n}-1)}) \leftarrow \mathrm{SHAKE}(\mathsf{0x96} \| \mathsf{seed}_{\mathbf{SE}}, (2\overline{m}n + \overline{m}\overline{n}) \cdot \mathsf{len}_{\chi})$
4: Sample error matrix $\mathbf{S}' \leftarrow \mathsf{Frodo.SampleMatrix}((\mathbf{r}^{(0)}, \mathbf{r}^{(1)}, \ldots, \mathbf{r}^{(\overline{m}n-1)}), \overline{m}, n, T_{\chi})$
5: Sample error matrix $\mathbf{E}' \leftarrow \mathsf{Frodo.SampleMatrix}((\mathbf{r}^{(\overline{m}n)}, \mathbf{r}^{(\overline{m}n+1)}, \ldots, \mathbf{r}^{(2\overline{m}n-1)}), \overline{m}, n, T_{\chi})$
6: Sample error matrix $\mathbf{E}'' \leftarrow \mathsf{Frodo.SampleMatrix}((\mathbf{r}^{(2\overline{m}n)}, \mathbf{r}^{(2\overline{m}n+1)}, \ldots, \mathbf{r}^{(2\overline{m}n+\overline{m}\overline{n}-1)}), \overline{m}, \overline{n}, T_{\chi})$
7: Compute $\mathbf{B}' = \mathbf{S}'\mathbf{A} + \mathbf{E}'$ and $\mathbf{V} = \mathbf{S}'\mathbf{B} + \mathbf{E}''$
8: **return** ciphertext $c \leftarrow (\mathbf{C}_1, \mathbf{C}_2) = (\mathbf{B}', \mathbf{V} + \mathsf{Frodo.Encode}(\mu))$

Algorithm 11 FrodoPKE.Dec.

Input: Ciphertext $c = (\mathbf{C}_1, \mathbf{C}_2) \in \mathbb{Z}_q^{\overline{m} \times n} \times \mathbb{Z}_q^{\overline{m} \times \overline{n}}$ and secret key $sk = \mathbf{S}^{\mathrm{T}} \in \mathbb{Z}_q^{\overline{n} \times n}$.
Output: Decrypted message $\mu' \in \mathcal{M}$.

1: Compute $\mathbf{M} = \mathbf{C}_2 - \mathbf{C}_1\mathbf{S}$
2: **return** message $\mu' \leftarrow$ Frodo.Decode($\mathbf{M}$)

Correctness of decryption: The decryption algorithm FrodoPKE.Dec computes

$$\begin{aligned}
\mathbf{M} &= \mathbf{C}_2 - \mathbf{C}_1\mathbf{S} \\
&= \mathbf{V} + \mathsf{Frodo.Encode}(\mu) - (\mathbf{S}'\mathbf{A} + \mathbf{E}')\mathbf{S} \\
&= \mathsf{Frodo.Encode}(\mu) + \mathbf{S}'\mathbf{B} + \mathbf{E}'' - \mathbf{S}'\mathbf{A}\mathbf{S} - \mathbf{E}'\mathbf{S} \\
&= \mathsf{Frodo.Encode}(\mu) + \mathbf{S}'\mathbf{A}\mathbf{S} + \mathbf{S}'\mathbf{E} + \mathbf{E}'' - \mathbf{S}'\mathbf{A}\mathbf{S} - \mathbf{E}'\mathbf{S} \\
&= \mathsf{Frodo.Encode}(\mu) + \mathbf{S}'\mathbf{E} + \mathbf{E}'' - \mathbf{E}'\mathbf{S} \\
&= \mathsf{Frodo.Encode}(\mu) + \mathbf{E}'''
\end{aligned}$$

for some error matrix $\mathbf{E}''' = \mathbf{S}'\mathbf{E} + \mathbf{E}'' - \mathbf{E}'\mathbf{S}$. Therefore, any B-bit substring of the message μ corresponding to an entry of $\mathbf{M}$ will be decrypted correctly if the condition in Lemma 2.18 is satisfied for the corresponding entry of $\mathbf{E}'''$.

Lemma 2.18. *Let* $q = 2^D$, $B \leq D$. *Then* $\mathrm{dc}(\mathrm{ec}(k) + e) = k$ *for any* $k, e \in \mathbb{Z}$ *such that* $0 \leq k < 2^B$ *and* $-q/2^{B+1} \leq e < q/2^{B+1}$.

Proof. This follows directly from the fact that $\mathrm{dc}(\mathrm{ec}(k) + e) = \lfloor k + e2^B/q \rceil \bmod 2^B$. □

$$for\ e = \frac{q}{2^{B+1}},$$

$$\left\lceil k + \frac{q}{2^{B+1}} * \frac{2^B}{q} \right\rfloor mod 2^B = \left\lceil k + \frac{1}{2} \right\rfloor mod 2^B = \mathrm{k} + 1 \bmod 2^B$$

$$for\ e = -\frac{q}{2^{B+1}},$$

$$\left\lceil k - \frac{q}{2^{B+1}} * \frac{2^B}{q} \right\rfloor mod 2^B = \left\lceil k - \frac{1}{2} \right\rfloor mod 2^B = \mathrm{k} - 1 \bmod 2^B$$

$$for\ -\frac{q}{2^{B+1}} \leq e < \frac{q}{2^{B+1}},$$

$$\left\lceil k + e * \frac{2^B}{q} \right\rfloor mod 2^B = \mathrm{k} \bmod 2^B$$

FrodoKEM

Key Encapsulation Mechanism , targeting IND-CCA2 security

IND-CCA2: Indistinguishability Under Chosen Ciphertext Attack

Algorithm:

- Challenger generates K_E, K_D
- Adversary (as many times as he wants): call the encryption or decryption oracle for an arbitrary plaintext/ciphertext
- Adversary generates two equal length messages m_0, m_1 also does additional operations(like encryption) in poly. time
- Challenger randomly chooses b = 0 or b = 1
- Challenger sends to adversary C=E(K_E,m_b)
- Adversary: perform additional operations in polynomial time, including calls to the oracles, for ciphertexts different than C. Output guess
- If guess=b the adversary wins

We will use Hefheinz, Höelmanns and Kiltz (HHK) FO transformation

$$\text{KEM}^{\not\perp\prime} = FO^{\not\perp\prime}(PKE, G_1, G_2, F)$$

Hash functions

$\text{KEM}^{\not\perp\prime}$.KeyGen():

1: $(pk, sk) \leftarrow_{\$} \text{PKE.KeyGen}()$
2: $\mathbf{s} \leftarrow_{\$} \{0,1\}^{\text{len}_\mathbf{s}}$
3: $\mathbf{pkh} \leftarrow G_1(pk)$
4: $sk' \leftarrow (sk, \mathbf{s}, pk, \mathbf{pkh})$
5: **return** (pk, sk')

$\text{KEM}^{\not\perp\prime}$.Encaps($pk$):

1: $\mu \leftarrow_{\$} \mathcal{M}$
2: $(\mathbf{r}, \mathbf{k}) \leftarrow G_2(G_1(pk)\|\mu)$
3: $c \leftarrow \text{PKE.Enc}(\mu, pk; \mathbf{r})$
4: $\mathbf{ss} \leftarrow F(c\|\mathbf{k})$
5: **return** $(c, \mathbf{ss})$

$\text{KEM}^{\not\perp\prime}$.Decaps($c, (sk, \mathbf{s}, pk, \mathbf{pkh})$):

1: $\mu' \leftarrow \text{PKE.Dec}(c, sk)$
2: $(\mathbf{r}', \mathbf{k}') \leftarrow G_2(\mathbf{pkh}\|\mu')$
3: $\mathbf{ss}'_0 \leftarrow F(c\|\mathbf{k}')$
4: $\mathbf{ss}'_1 \leftarrow F(c\|\mathbf{s})$
5: (in constant time) $\mathbf{ss}' \leftarrow \mathbf{ss}'_0$ if $c = \text{PKE.Enc}(\mu', pk; \mathbf{r}')$ else $\mathbf{ss}' \leftarrow \mathbf{ss}'_1$
6: **return** $\mathbf{ss}'$

Algorithm 12 FrodoKEM.KeyGen.

Input: None.

Output: Key pair (pk, sk') with $pk \in \{0,1\}^{\mathsf{len}_{\mathsf{seed}_\mathbf{A}} + D \cdot n \cdot \overline{n}}$, $sk' \in \{0,1\}^{\mathsf{len}_\mathbf{s} + \mathsf{len}_{\mathsf{seed}_\mathbf{A}} + D \cdot n \cdot \overline{n}} \times \mathbb{Z}_q^{\overline{n} \times n} \times \{0,1\}^{\mathsf{len}_{\mathbf{pkh}}}$.

PKE.KeyGen()

1: Choose uniformly random seeds $\mathbf{s} \| \mathsf{seed}_{\mathbf{SE}} \| \mathbf{z} \leftarrow\!\!\$\ U(\{0,1\}^{\mathsf{len}_\mathbf{s} + \mathsf{len}_{\mathsf{seed}_{\mathbf{SE}}} + \mathsf{len}_\mathbf{z}})$
2: Generate pseudorandom seed $\mathsf{seed}_\mathbf{A} \leftarrow \mathrm{SHAKE}(\mathbf{z}, \mathsf{len}_{\mathsf{seed}_\mathbf{A}})$
3: Generate the matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times n}$ via $\mathbf{A} \leftarrow \mathsf{Frodo.Gen}(\mathsf{seed}_\mathbf{A})$
4: Generate pseudorandom bit string $(\mathbf{r}^{(0)}, \mathbf{r}^{(1)}, \ldots, \mathbf{r}^{(2n\overline{n}-1)}) \leftarrow \mathrm{SHAKE}(\mathsf{0x5F} \| \mathsf{seed}_{\mathbf{SE}}, 2n\overline{n} \cdot \mathsf{len}_\chi)$
5: Sample error matrix $\mathbf{S}^\mathrm{T} \leftarrow \mathsf{Frodo.SampleMatrix}((\mathbf{r}^{(0)}, \mathbf{r}^{(1)}, \ldots, \mathbf{r}^{(n\overline{n}-1)}), \overline{n}, n, T_\chi)$
6: Sample error matrix $\mathbf{E} \leftarrow \mathsf{Frodo.SampleMatrix}((\mathbf{r}^{(n\overline{n})}, \mathbf{r}^{(n\overline{n}+1)}, \ldots, \mathbf{r}^{(2n\overline{n}-1)}), n, \overline{n}, T_\chi)$
7: Compute $\mathbf{B} \leftarrow \mathbf{AS} + \mathbf{E}$
8: Compute $\mathbf{b} \leftarrow \mathsf{Frodo.Pack}(\mathbf{B})$ Converts matrix to the bits
9: Compute $\mathbf{pkh} \leftarrow \mathrm{SHAKE}(\mathsf{seed}_\mathbf{A} \| \mathbf{b}, \mathsf{len}_{\mathbf{pkh}})$ Put public key inside of hash
10: **return** public key $pk \leftarrow \mathsf{seed}_\mathbf{A} \| \mathbf{b}$ and secret key $sk' \leftarrow (\mathbf{s} \| \mathsf{seed}_\mathbf{A} \| \mathbf{b}, \mathbf{S}^\mathrm{T}, \mathbf{pkh})$

Algorithm 13 FrodoKEM.Encaps.

Input: Public key $pk = \mathsf{seed_A} \| \mathbf{b} \in \{0,1\}^{\mathsf{len}_{\mathsf{seed_A}} + D \cdot n \cdot \overline{n}}$.
Output: Ciphertext $\mathbf{c_1} \| \mathbf{c_2} \in \{0,1\}^{(\overline{m} \cdot n + \overline{m} \cdot \overline{n}) D}$ and shared secret $\mathbf{ss} \in \{0,1\}^{\mathsf{len}_{\mathsf{ss}}}$.

1: Choose a uniformly random key $\mu \leftarrow_{\$} U(\{0,1\}^{\mathsf{len}_\mu})$
2: Compute $\mathbf{pkh} \leftarrow \mathrm{SHAKE}(pk, \mathsf{len}_{\mathbf{pkh}})$ Puts public key inside of 1. hash
3: Generate pseudorandom values $\mathsf{seed_{SE}} \| \mathbf{k} \leftarrow \mathrm{SHAKE}(\mathbf{pkh} \| \mu, \mathsf{len}_{\mathsf{seed_{SE}}} + \mathsf{len}_{\mathbf{k}})$
4: Generate pseudorandom bit string $(\mathbf{r}^{(0)}, \mathbf{r}^{(1)}, \ldots, \mathbf{r}^{(2\overline{m}n + \overline{m}\overline{n} - 1)}) \leftarrow \mathrm{SHAKE}(\mathtt{0x96} \| \mathsf{seed_{SE}}, (2\overline{m}n + \overline{m}\overline{n}) \cdot \mathsf{len}_\chi)$
5: Sample error matrix $\mathbf{S'} \leftarrow \mathsf{Frodo.SampleMatrix}((\mathbf{r}^{(0)}, \mathbf{r}^{(1)}, \ldots, \mathbf{r}^{(\overline{m}n-1)}), \overline{m}, n, T_\chi)$
6: Sample error matrix $\mathbf{E'} \leftarrow \mathsf{Frodo.SampleMatrix}((\mathbf{r}^{(\overline{m}n)}, \mathbf{r}^{(\overline{m}n+1)}, \ldots, \mathbf{r}^{(2\overline{m}n-1)}), \overline{m}, n, T_\chi)$
7: Generate $\mathbf{A} \leftarrow \mathsf{Frodo.Gen}(\mathsf{seed_A})$
8: Compute $\mathbf{B'} \leftarrow \mathbf{S'A} + \mathbf{E'}$
9: Compute $\mathbf{c_1} \leftarrow \mathsf{Frodo.Pack}(\mathbf{B'})$ Converts matrix to the bits
10: Sample error matrix $\mathbf{E''} \leftarrow \mathsf{Frodo.SampleMatrix}((\mathbf{r}^{(2\overline{m}n)}, \mathbf{r}^{(2\overline{m}n+1)}, \ldots, \mathbf{r}^{(2\overline{m}n+\overline{m}\overline{n}-1)}), \overline{m}, \overline{n}, T_\chi)$
11: Compute $\mathbf{B} \leftarrow \mathsf{Frodo.Unpack}(\mathbf{b}, n, \overline{n})$ Converts bits to the matrix
12: Compute $\mathbf{V} \leftarrow \mathbf{S'B} + \mathbf{E''}$
13: Compute $\mathbf{C} \leftarrow \mathbf{V} + \mathsf{Frodo.Encode}(\mu)$ Converts key to matrix and sums with V
14: Compute $\mathbf{c_2} \leftarrow \mathsf{Frodo.Pack}(\mathbf{C})$ Converts matrix to the bits
15: Compute $\mathbf{ss} \leftarrow \mathrm{SHAKE}(\mathbf{c_1} \| \mathbf{c_2} \| \mathbf{k}, \mathsf{len}_{\mathbf{ss}})$
16: **return** ciphertext $\mathbf{c_1} \| \mathbf{c_2}$ and shared secret $\mathbf{ss}$

PKE.Enc()

Algorithm 14 FrodoKEM.Decaps.

Input: Ciphertext $\mathbf{c}_1\|\mathbf{c}_2 \in \{0,1\}^{(\overline{m}\cdot n+\overline{m}\cdot\overline{n})D}$, secret key $sk' = (\mathbf{s}\|\mathsf{seed}_\mathbf{A}\|\mathbf{b},\mathbf{S}^\mathrm{T},\mathbf{pkh}) \in \{0,1\}^{\mathsf{len_s}+\mathsf{len}_{\mathsf{seed}_\mathbf{A}}+D\cdot n\cdot\overline{n}} \times \mathbb{Z}_q^{\overline{n}\times n} \times \{0,1\}^{\mathsf{len}_\mathbf{pkh}}$.

Output: Shared secret $\mathbf{ss} \in \{0,1\}^{\mathsf{len}_\mathbf{ss}}$.

PKE.Dec()

1: $\mathbf{B}' \leftarrow \mathsf{Frodo.Unpack}(\mathbf{c}_1, \overline{m}, n)$
2: $\mathbf{C} \leftarrow \mathsf{Frodo.Unpack}(\mathbf{c}_2, \overline{m}, \overline{n})$
3: Compute $\mathbf{M} \leftarrow \mathbf{C} - \mathbf{B}'\mathbf{S}$
4: Compute $\mu' \leftarrow \mathsf{Frodo.Decode}(\mathbf{M})$
5: Parse $pk \leftarrow \mathsf{seed}_\mathbf{A}\|\mathbf{b}$ why ?
6: Generate pseudorandom values $\mathsf{seed}_{\mathbf{SE}}{}'\|\mathbf{k}' \leftarrow \mathrm{SHAKE}(\mathbf{pkh}\|\mu', \mathsf{len}_{\mathsf{seed}_\mathbf{SE}} + \mathsf{len}_\mathbf{k})$

PKE.Enc()

Encrypts the retreived message μ'

7: Generate pseudorandom bit string $(\mathbf{r}^{(0)}, \mathbf{r}^{(1)}, \ldots, \mathbf{r}^{(2\overline{m}n+\overline{m}\overline{n}-1)}) \leftarrow \mathrm{SHAKE}(\mathtt{0x96}\|\mathsf{seed}_\mathbf{SE}{}', (2\overline{m}n + \overline{m}\overline{n}) \cdot \mathsf{len}_\chi)$
8: Sample error matrix $\mathbf{S}' \leftarrow \mathsf{Frodo.SampleMatrix}((\mathbf{r}^{(0)}, \mathbf{r}^{(1)}, \ldots, \mathbf{r}^{(\overline{m}n-1)}), \overline{m}, n, T_\chi)$
9: Sample error matrix $\mathbf{E}' \leftarrow \mathsf{Frodo.SampleMatrix}((\mathbf{r}^{(\overline{m}n)}, \mathbf{r}^{(\overline{m}n+1)}, \ldots, \mathbf{r}^{(2\overline{m}n-1)}), \overline{m}, n, T_\chi)$
10: Generate $\mathbf{A} \leftarrow \mathsf{Frodo.Gen}(\mathsf{seed}_\mathbf{A})$
11: Compute $\mathbf{B}'' \leftarrow \mathbf{S}'\mathbf{A} + \mathbf{E}'$
12: Sample error matrix $\mathbf{E}'' \leftarrow \mathsf{Frodo.SampleMatrix}((\mathbf{r}^{(2\overline{m}n)}, \mathbf{r}^{(2\overline{m}n+1)}, \ldots, \mathbf{r}^{(2\overline{m}n+\overline{m}\overline{n}-1)}), \overline{m}, \overline{n}, T_\chi)$
13: Compute $\mathbf{B} \leftarrow \mathsf{Frodo.Unpack}(\mathbf{b}, n, \overline{n})$
14: Compute $\mathbf{V} \leftarrow \mathbf{S}'\mathbf{B} + \mathbf{E}''$
15: Compute $\mathbf{C}' \leftarrow \mathbf{V} + \mathsf{Frodo.Encode}(\mu')$
16: (in constant time) $\overline{\mathbf{k}} \leftarrow \mathbf{k}'$ if $(\mathbf{B}'\|\mathbf{C} = \mathbf{B}''\|\mathbf{C}')$ else $\overline{\mathbf{k}} \leftarrow \mathbf{s}$
17: Compute $\mathbf{ss} \leftarrow \mathrm{SHAKE}(\mathbf{c}_1\|\mathbf{c}_2\|\overline{\mathbf{k}}, \mathsf{len}_\mathbf{ss})$
18: **return** shared secret $\mathbf{ss}$

See Section 6.1 for a discussion on implementing this step 16 in constant time.

If $\mu' = \mu$ in 4 , then,

- $seed_{SE'}||k' = seed_{SE}||k$ in 6
- Pseudorandom bit string in 7 is same as FrodoKEM.Enc()
- Will return k' in 16
- ss in Encaps and Decaps will be same

Parameters

ALGORITHM	NIST SECURITY LEVEL	SECURITY LEVEL EQUIVALANCE
Frodo-640	Level 1	AES-128
Frodo-976	Level 3	AES-192
Frodo-1344	Level 5	AES-256

‘The task of parameter selection is framed as a combinatorial optimization problem, where the objective function is the ciphertext’s size, and the constraints are dictated by the target security level, probability of decryption failure, and computational efficiency.

The optimization problem is solved by sweeping the parameter space, subject to simple pruning techniques.

We perform this sweep of the parameter space using the Python scripts that accompany the submission, in the folder Additional Implementations/Parameter Search Scripts.’

Table 1: **Parameters at a glance**

	n	q	σ	support of χ	B	$\bar{m} \times \bar{n}$	c size (bytes)
Frodo-640	640	2^{15}	2.8	$[-12 \ldots 12]$	2	8×8	9,720
Frodo-976	976	2^{16}	2.3	$[-10 \ldots 10]$	3	8×8	15,744
Frodo-1344	1344	2^{16}	1.4	$[-6 \ldots 6]$	4	8×8	21,632

Table 2: **Security bounds**

	target level	failure rate	LWE security			IND-CCA security
			C	Q	P	C
Frodo-640	1	$2^{-138.7}$	145	132	104	141
Frodo-976	3	$2^{-199.6}$	210	191	150	206
Frodo-1344	5	$2^{-252.5}$	275	250	197	268

Table 3: **Error distributions**

	σ	Probability of (in multiples of 2^{-16})													Rényi	
		0	±1	±2	±3	±4	±5	±6	±7	±8	±9	±10	±11	±12	order	divergence
$\chi_{\text{Frodo-640}}$	2.8	9288	8720	7216	5264	3384	1918	958	422	164	56	17	4	1	200	0.324×10^{-4}
$\chi_{\text{Frodo-976}}$	2.3	11278	10277	7774	4882	2545	1101	396	118	29	6	1			500	0.140×10^{-4}
$\chi_{\text{Frodo-1344}}$	1.4	18286	14320	6876	2023	364	40	2							1000	0.264×10^{-4}

Table 4: **Cryptographic parameters for Frodo-640, Frodo-976, and Frodo-1344**

	Frodo-640	Frodo-976	Frodo-1344
D	15	16	16
q	32768	65536	65536
n	640	976	1344
$\overline{m} = \overline{n}$	8	8	8
B	2	3	4
$\text{len}_{\text{seed}_\mathbf{A}}$	128	128	128
$\text{len}_{\mathbf{z}}$	128	128	128
$\text{len}_{\mu} = \ell$	128	192	256
$\text{len}_{\text{seed}_\mathbf{SE}}$	128	192	256
$\text{len}_{\mathbf{s}}$	128	192	256
$\text{len}_{\mathbf{k}}$	128	192	256
$\text{len}_{\mathbf{pkh}}$	128	192	256
$\text{len}_{\mathbf{ss}}$	128	192	256
len_{χ}	16	16	16
χ	$\chi_{\text{Frodo-640}}$	$\chi_{\text{Frodo-976}}$	$\chi_{\text{Frodo-1344}}$
SHAKE	SHAKE128	SHAKE256	SHAKE256

Scheme	secret key sk	public key pk	ciphertext c	shared secret ss
FrodoKEM-640	19,888 (10,272 + 9,616)	9,616	9,720	16
FrodoKEM-976	31,296 (15,664 + 15,632)	15,632	15,744	24
FrodoKEM-1344	43,088 (21,568 + 21,520)	21,520	21,632	32

References

1. Naehrig, M., Alkim, E., Bos, J., Ducas, L., Easterbrook, K., LaMacchia, B., Longa, P., Mironov, I., Nikolaenko, V., Peikert, C., Raghunathan, A., Stebila, D.: FrodoKEM.