# FrodoKEM
# Learning With Errors Key Encapsulation

Post-Quantum Reading Group

Marina Polubelova

April 11, 2019

# FrodoKEM

- Lattice-based key encapsulation mechanism whose security relies on the hardness of the LWE problem
- Six variants determined by their security level (Level 1, Level 3, Level 5) and the primitive used for generating pseudorandomly a public matrix A (AES or SHAKE)

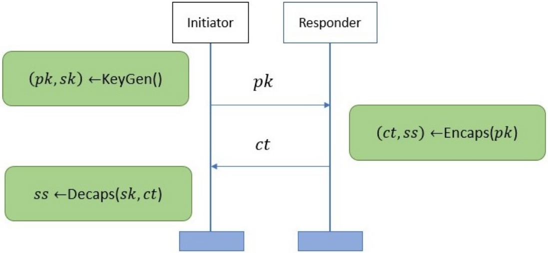
| | Classical security | Quantum security | Examples |
|---|---|---|---|
| Level 1 | 128 bits | 64 bits | AES128 |
| Level 2 | 128 bits | 80 bits | SHA256/SHA3-256 |
| Level 3 | 192 bits | 96 bits | AES192 |
| Level 4 | 192 bits | 128 bits | SHA384/SHA3-384 |
| Level 5 | 256 bits | 128 bits | AES256 |

# FrodoKEM

- Simple design
  - Matrix-vector operations
  - Reduction modulo q can be computed for free
  - Error sampling from a small lookup table
  - No reconciliation mechanism
- Dynamically and pseudorandomly generated public matrix A (to avoid the possibility of backdoors and all-for-the-price-of-one attacks)
- Security of FrodoKEM is supported both by security reductions and by analysis of the best known cryptanalytic attacks

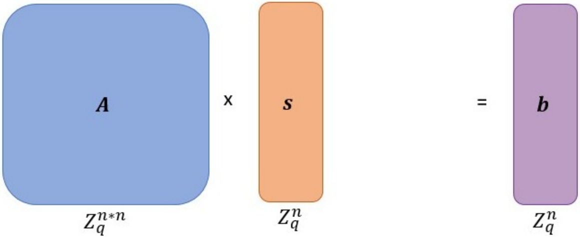
# Key Encapsulation Mechanism (KEM)

**Key Encapsulation Mechanism** is a tuple of algorithms ($KeyGen$, $Encaps$, $Decaps$) along with a finite keyspace $K$




E.g., TLS, SSH

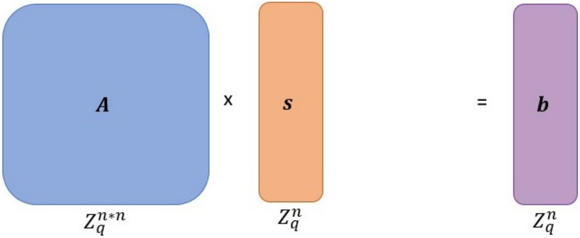
Random matrix $A$, secret $s$, and output $b$




**Find** $s$?

# Learning with Errors (LWE)

Random matrix $A$, secret $s$, and output $b$

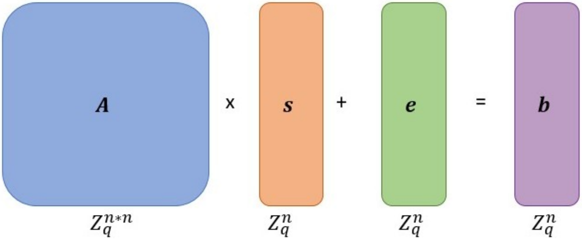



**Find** $s$?

*Easy by Gaussian elimination*

# Learning with Errors (LWE)

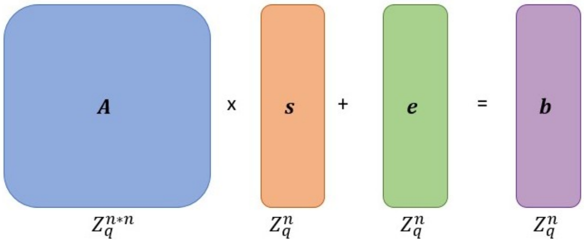
Random matrix $A$, secret $s$, small noise $e$, and output $b$




**Find** $s$?

# Learning with Errors (LWE)

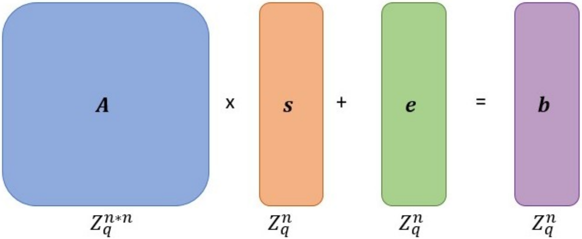
Random matrix $A$, secret $s$, small noise $e$, and output $b$




**Find** $s$?

*Search LWE problem*

Random matrix $A$, secret $s$, small noise $e$, and output $b$



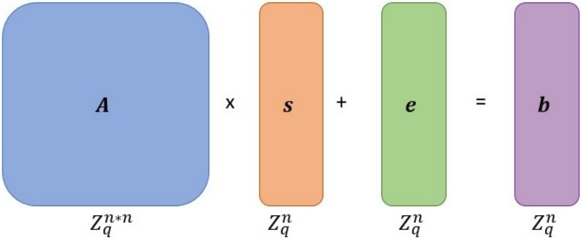

**Find** $s$?

*Search LWE problem*

**Distinguish** $(A, As + e)$ **from** $(A, random)$**?**

# Learning with Errors (LWE)

Random matrix $A$, secret $s$, small noise $e$, and output $b$




**Find** $s$?

*Search LWE problem*

**Distinguish** $(A, As + e)$ **from** $(A, random)$**?**

*Decision LWE problem*

# Learning with Errors (LWE)

- Generic, algebraically unstructured lattices
  - LWE
- Adding structure for better performance
  - Ring-LWE
  - Module-LWE

# Learning with Errors (LWE)

- Generic, algebraically unstructured lattices
  - LWE
- Adding structure for better performance
  - Ring-LWE
  - Module-LWE
- Reminder

**Vectors** $x \in \mathbb{Z}_q^n$

- $x = (x_0, \ldots, x_{n-1})$ where $x_i \in \mathbb{Z}_q$

**Ring elements** $r \in R_q = \mathbb{Z}_q[X]/(X^n + 1)$

- $r = r_0 + r_1 \cdot X + \ldots + r_{n-1} \cdot X^{n-1}$ where $r_i \in \mathbb{Z}_q$
- Coefficient embedding $r = (r_0, \ldots, r_{n-1}) \in \mathbb{Z}_q^n$

**Module elements** $m \in R_q^d$

- $m = (m_0, \ldots, m_{d-1})$ where $m_i \in R_q$
- if $d = 1$, we get Ring-LWE
- if $R_q = \mathbb{Z}_q$ and $d = n$, we get LWE

# LWE in practice

- **FrodoKEM** relies on the LWE problem
- **NewHope** relies on the Ring-LWE problem
- **Kyber** relies on the Module-LWE problem

# FrodoPKE

**Algorithm 9** FrodoPKE.KeyGen.

**Input:** None.
**Output:** Key pair $(pk, sk) \in (\{0,1\}^{\mathsf{len}_{\mathsf{seed}_\mathbf{A}}} \times \mathbb{Z}_q^{n\times\overline{n}}) \times \mathbb{Z}_q^{n\times\overline{n}}$.

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

**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 $c = (\mathbf{C}_1, \mathbf{C}_2) \in \mathbb{Z}_q^{\overline{m}\times n} \times \mathbb{Z}_q^{\overline{m}\times\overline{n}}$.

Generate $\mathbf{A} \leftarrow \mathsf{Frodo.Gen}(\mathsf{seed}_\mathbf{A})$
Choose a uniformly random seed $\mathsf{seed}_\mathbf{SE} \leftarrow\!\!\$\; U(\{0,1\}^{\mathsf{len}_{\mathsf{seed}_\mathbf{SE}}})$
Generate pseudorandom bit string $(\mathbf{r}^{(0)}, \mathbf{r}^{(1)}, \dots, \mathbf{r}^{(2\overline{m}n+\overline{m}\overline{n}-1)}) \leftarrow \mathrm{SHAKE}(\mathtt{0x96}\|\mathsf{seed}_\mathbf{SE}, 2\overline{m}n+\overline{m}\overline{n}\cdot\mathsf{len}_\chi)$
Sample error matrix $\mathbf{S}' \leftarrow \mathsf{Frodo.SampleMatrix}((\mathbf{r}^{(0)}, \mathbf{r}^{(1)}, \dots, \mathbf{r}^{(\overline{m}n-1)}), \overline{m}, n, T_\chi)$
Sample error matrix $\mathbf{E}' \leftarrow \mathsf{Frodo.SampleMatrix}((\mathbf{r}^{(\overline{m}n)}, \mathbf{r}^{(\overline{m}n+1)}, \dots, \mathbf{r}^{(2\overline{m}n-1)}), \overline{m}, n, T_\chi)$
Sample error matrix $\mathbf{E}'' \leftarrow \mathsf{Frodo.SampleMatrix}((\mathbf{r}^{(2\overline{m}n)}, \mathbf{r}^{(2\overline{m}n+1)}, \dots, \mathbf{r}^{(2\overline{m}n+\overline{m}\overline{n}-1)}), \overline{m}, \overline{n}, T_\chi)$
Compute $\mathbf{B}' = \mathbf{S}'\mathbf{A} + \mathbf{E}'$ and $\mathbf{V} = \mathbf{S}'\mathbf{B} + \mathbf{E}''$
**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 $sk = \mathbf{S} \in \mathbb{Z}_q^{n\times\overline{n}}$.
**Output:** Decrypted message $\mu' \in \mathcal{M}$.

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

# Correctness of IND-CPA PKE

$$
\begin{aligned}
\mathbf{M} &= \mathbf{C_2} - \mathbf{C_1 S} \\
&= \mathbf{V} + \text{Frodo.Encode}(\mu) - (\mathbf{S'A} + \mathbf{E'})\mathbf{S} \\
&= \text{Frodo.Encode}(\mu) + \mathbf{S'B} + \mathbf{E''} - \mathbf{S'AS} - \mathbf{E'S} \\
&= \text{Frodo.Encode}(\mu) + \mathbf{S'AS} + \mathbf{S'E} + \mathbf{E''} - \mathbf{S'AS} - \mathbf{E'S} \\
&= \text{Frodo.Encode}(\mu) + \mathbf{S'E} + \mathbf{E''} - \mathbf{E'S} \\
&= \text{Frodo.Encode}(\mu) + \mathbf{E'''}
\end{aligned}
$$

**Lemma**

*Let $q = 2^D$, $B \leq D$. Then $dc(ec(k) + e) = k$ for any $k, e \in \mathbb{Z}$ s.t. $0 \leq k < 2^B$ and $-\frac{q}{2^{B+1}} \leq e < \frac{q}{2^{B+1}}$*

# Transform from IND-CPA PKE to IND-CCA KEM

- the Fujisaki-Okamoto transform with implicit rejection (with some modifications)

$\mathsf{KEM}^{\not\perp'}.\mathrm{KeyGen}()$:

$(pk, sk) \leftarrow_\$ \mathsf{PKE}.\mathrm{KeyGen}()$
$\mathbf{s} \leftarrow_\$ \{0,1\}^{\mathsf{len_s}}$
$\mathbf{pkh} \leftarrow G_1(pk)$
$sk' \leftarrow (sk, \mathbf{s}, pk, \mathbf{pkh})$
$\textbf{return} \ \ (pk, sk')$

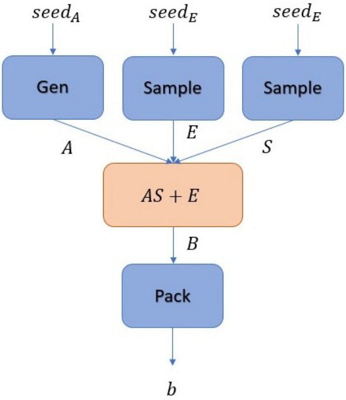
$\mathsf{KEM}^{\not\perp'}.\mathrm{Encaps}(pk)$:

$\mu \leftarrow_\$ \mathcal{M}$
$(\mathbf{r}, \mathbf{k}) \leftarrow G_2(G_1(pk) \| \mu)$
$c \leftarrow \mathsf{PKE}.\mathrm{Enc}(\mu, pk; \mathbf{r})$
$\mathbf{ss} \leftarrow F(c \| \mathbf{k})$
$\textbf{return} \ \ (c, \mathbf{ss})$

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

$\mu' \leftarrow \mathsf{PKE}.\mathrm{Dec}(c, sk)$
$(\mathbf{r}', \mathbf{k}') \leftarrow G_2(\mathbf{pkh} \| \mu')$
$\textbf{if} \ c = \mathsf{PKE}.\mathrm{Enc}(\mu', pk; \mathbf{r}') \ \textbf{then}$
$\quad \textbf{return} \ \ \mathbf{ss}' \leftarrow F(c \| \mathbf{k}')$
$\textbf{else}$
$\quad \textbf{return} \ \ \mathbf{ss}' \leftarrow F(c \| \mathbf{s})$

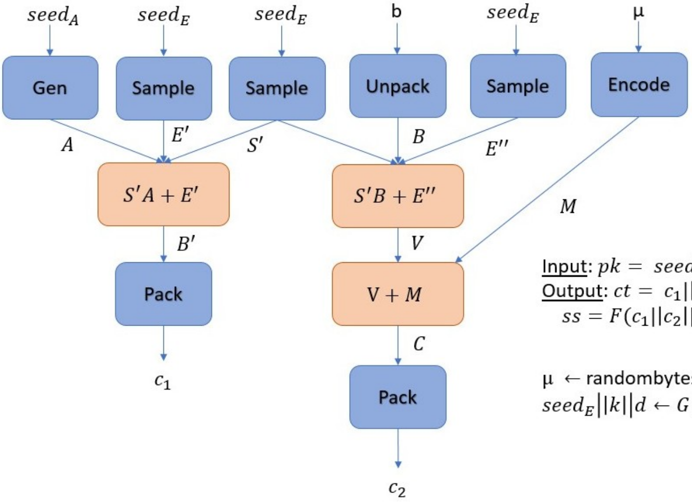
# FrodoKEM.KeyGen




<u>Input</u>: None
<u>Output</u>: $pk = seed_A || b$
$sk = s || seed_A || b || to\_bytes(S)$

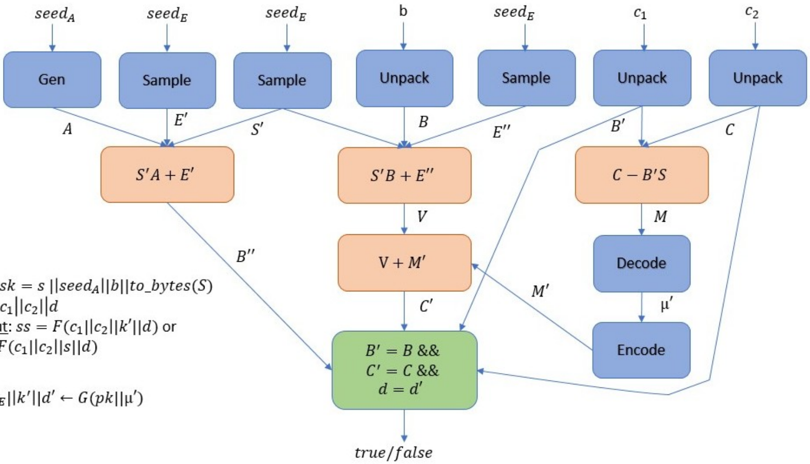
$s || seed_A || z \leftarrow$ randombytes()
$seed_A \leftarrow \mathrm{H(z)}$

# FrodoKEM.Encaps




Input: $pk = seed_A || b$
Output: $ct = c_1 || c_2 || d$
$ss = F(c_1 || c_2 || k || d)$

$\mu \leftarrow$ randombytes()
$seed_E || k || d \leftarrow G(pk || \mu)$




Input $sk = s\,||seed_A||b||to\_bytes(S)$
$ct = c_1||c_2||d$
Output: $ss = F(c_1||c_2||k'||d)$ or
$ss = F(c_1||c_2||s||d)$

$seed'_E||k'||d' \leftarrow G(pk||\mu')$

# FrodoKEM parameters

- $\chi$, a probability distribution on $\mathbb{Z}$
- $q = 2^D$, a power-of-two integer modulus with $D \leq 16$
- $n$, $\bar{m}$, $\bar{n}$, integer matrix dimensions with $n \equiv 0 \pmod 8$

**Find** $(q, n, \chi)$:

- ciphertext's size, which is $D \times \bar{m} \times (n + \bar{n})$
- target security level
- probability of decryption failure
- computation efficiency

# FrodoKEM parameters

| | FrodoKEM-640 | FrodoKEM-976 | FrodoKEM-1344 |
|---|---|---|---|
| $q$ | $2^{15}$ | $2^{16}$ | $2^{16}$ |
| $n$ | 640 | 976 | 1344 |
| $\bar{n}$ | 8 | 8 | 8 |
| $\sigma$ | 2.8 | 2.3 | 1.4 |
| $c$ size | 9,736 | 15,768 | 21,664 |
| failure prob. | $2^{-148.8}$ | $2^{-199.6}$ | $2^{-252.5}$ |
| security C | 143 | 209 | 274 |
| security Q | 103 | 150 | 196 |

$\chi$ is the discrete Gaussian distribution with the standard deviation $\sigma$

# FrodoKEM parameters

Size (in bytes) of inputs and outputs of FrodoKEM

| | secret key | public key | ciphertext | shared secret |
|---|---|---|---|---|
| FrodoKEM-640 | 19,888 | 9,616 | 9,720 | 16 |
| FrodoKEM-976 | 31,296 | 15,632 | 15,744 | 24 |
| FrodoKEM-1344 | 43,088 | 21,520 | 21,632 | 32 |

# Performance

Performance (in thousands of cycles) of FrodoKEM on a 3.4GHz Intel Core i7-6700 processor with matrix A generated using AES128

| Scheme | KeyGen | Encaps | Decaps | Total (Encaps + Decaps) |
|---|---|---|---|---|
| **Optimized Implementation (AES from OpenSSL)** | | | | |
| FrodoKEM-640-AES | 1,384 | 1,858 | 1,749 | 3,607 |
| FrodoKEM-976-AES | 2,820 | 3,559 | 3,400 | 6,959 |
| FrodoKEM-1344-AES | 4,756 | 5,981 | 5,748 | 11,729 |
| **Additional implementation using AVX2 intrinsic instructions (AES from OpenSSL)** | | | | |
| FrodoKEM-640-AES | 1,388 | 1,879 | 1,768 | 3,647 |
| FrodoKEM-976-AES | 2,885 | 3,553 | 3,407 | 6,960 |
| FrodoKEM-1344-AES | 4,744 | 6,026 | 5,770 | 11,796 |
| **Additional implementation using AVX2 intrinsic instructions (standalone AES)** | | | | |
| FrodoKEM-640-AES | 1,388 | 1,878 | 1,767 | 3,645 |
| FrodoKEM-976-AES | 2,829 | 3,599 | 3,447 | 7,046 |
| FrodoKEM-1344-AES | 4,791 | 6,058 | 5,791 | 11,849 |

# Performance

Performance (in thousands of cycles) of FrodoKEM on a 3.4GHz Intel Core i7-6700 processor with matrix A generated using SHAKE128

| Scheme | KeyGen | Encaps | Decaps | Total (Encaps + Decaps) |
|---|---|---|---|---|
| **Optimized Implementation (plain C SHAKE)** | | | | |
| FrodoKEM-640-SHAKE | 7,626 | 8,362 | 8,248 | 16,610 |
| FrodoKEM-976-SHAKE | 16,841 | 18,077 | 17,925 | 36,002 |
| FrodoKEM-1344-SHAKE | 30,301 | 32,611 | 32,387 | 64,998 |
| **Additional implementation using AVX2 intrinsics (SHAKE4x using AVX2)** | | | | |
| FrodoKEM-640-SHAKE | 4,015 | 4,442 | 4,331 | 8,773 |
| FrodoKEM-976-SHAKE | 8,579 | 9,302 | 9,143 | 18,445 |
| FrodoKEM-1344-SHAKE | 15,044 | 16,359 | 16,147 | 32,506 |

# Performance

Performance (in thousands of cycles) of the optimised implementation of FrodoKEM on a 1.992GHz 64-bit ARMv8 processor

| Scheme | KeyGen | Encaps | Decaps | Total (Encaps + Decaps) |
|---|---|---|---|---|
| **Optimized Implementation (AES from OpenSSL)** | | | | |
| FrodoKEM-640-AES | 3,470 | 4,057 | 3,969 | 8,026 |
| FrodoKEM-976-AES | 7,219 | 8,530 | 8,014 | 16,544 |
| FrodoKEM-1344-AES | 12,789 | 14,854 | 14,635 | 29,489 |
| **Optimized implementation (plain C AES)** | | | | |
| FrodoKEM-640-AES | 44,354 | 44,766 | 44,765 | 89,531 |
| FrodoKEM-976-AES | 101,540 | 102,551 | 102,460 | 205,011 |
| FrodoKEM-1344-AES | 191,359 | 193,123 | 192,458 | 385,581 |
| **Optimized implementation (plain C** SHAKE) | | | | |
| FrodoKEM-640-AES | 11,278 | 12,411 | 12,311 | 24,722 |
| FrodoKEM-976-AES | 24,844 | 27,033 | 26,936 | 53,969 |
| FrodoKEM-1344-AES | 44,573 | 48,554 | 48,449 | 97,003 |

# Security reductions

- FrodoKEM is an IND-CCA-secure KEM under the assumption that FrodoKEM is an OW-CPA-secure PKE scheme
- FrodoPKE is an IND-CPA secure PKE scheme under the assumption that the corresponding normal-form learning with errors decision problem is hard
- The normal-form learning with errors decision problem is hard under the assumption that the uniform-secret learning with errors decision problem is hard for the same parameters, except for a small additive loss in the number of samples
- The (*average-case*) uniform-secret learning with errors decision problem, with the particular values of $\sigma$ and an appropriate bound on the number of samples, is hard under the assumption that the *worst-case* bounded distance decoding with discrete Gaussian samples problem (BDDwDGS) is hard for related parameters

# Formal definitions

## Definition (Lattice)

A (full-rank) *n-dimensional lattice* $\mathcal{L}$ is a discrete additive subset of $\mathbb{R}^n$ for which $span_{\mathbb{R}}(\mathcal{L}) = \mathbb{R}^n$.
Any such lattice can be generated by a (non-unique) *basis*
$\mathbf{B} = \{\mathbf{b_1}, \dots, \mathbf{b_n}\} \subset \mathbb{R}^n$ of linearly independent vectors, as
$\mathcal{L} = \mathcal{L}(\mathbf{B}) := \{\sum_{i=1}^{n} z_i \mathbf{b_i} : z_i \in \mathbb{Z}\}$

## Definition (Minimum distance)

For a lattice $\mathcal{L} \subset \mathbb{R}^n$, its *minimum distance* is the length (in the Euclidean norm) of a shortest non-zero lattice vector: $\lambda_1(\mathcal{L}) = \min_{\mathbf{v} \in \mathcal{L} \setminus \{0\}} \|\mathbf{v}\|$

# Formal definitions

**Definition (Discrete Gaussian)**

For a lattice $\mathcal{L} \subset \mathbb{R}^n$, the *discrete Gaussian distribution* over $\mathcal{L}$ with parameter $s$, denoted $D_{\mathcal{L},s}$, is defined as $D_s(\mathbf{x}) = \frac{\rho_s(\mathbf{x})}{\rho_s(\mathcal{L})}$ for $\mathbf{x} \in \mathcal{L}$ (and $D_s(\mathbf{x}) = 0$ otherwise), where $\rho_s(\mathcal{L}) = \sum_{\mathbf{v} \in \mathcal{L}} \rho_s(\mathbf{v})$ is a normalisation factor

**Definition**

For a lattice $\mathcal{L} \subset \mathbb{R}^n$ and positive reals $d < \frac{\lambda_1(\mathcal{L})}{2}$ and $r > 0$, an instance of the *bounded-distance decoding with discrete Gaussian samples* problem $BDDwDGS_{\mathcal{L},d,r}$ is a point $\mathbf{t} \in \mathbb{R}^n$ s.t. $dist(\mathbf{t}, \mathcal{L}) \leq d$, and access to an oracle that samples from $D_{\mathcal{L}*,s}$ for any queried $s \geq r$. The goal is to output the (unique) lattice point $\mathbf{v} \in \mathcal{L}$ closest to $\mathbf{t}$