Module 8: Concrete Security

Hyeongmin Choe

Seoul National University

July 16-17, 2024

Module 8: Concrete Security

- July 16-17, Tuesday and Wednesday afternoons (7h)
- **Focus**:
  How to estimate concrete security of PQC schemes?
- Lecture + Tutorial
- **16th, Tuesday**
  - Lattice Estimator
    (Lecture + short Tutorial)
  - Kyber (Lecture)
  - Smaug (Tutorial)
- **17th, Wednesday**
  - Falcon (Lecture)
  - Dilithium (Tutorial)

Lattice Estimator

Lattice Estimator

The Lattice Estimator (formerly the LWE Estimator) is a tool used throughout industry and academia to estimate the security level of Learning with Errors-based parameter sets.

- Ben Curtis

Lattice Estimator

- Bunch of existing attacks against Lattice problems

LWE primal
(uSVP, BDD, hybrid, ..)

Arora-GB
(Gröbner bases)

LWE dual
(dual, hybrid, mitm-hybrid, ..)

SIS
(BKZ, ..)

Coded-BKW

NTRU primal
(uSVP, BDD, hybrid, dense sublattice, ..)

Lattice Estimator

- Bunch of existing attacks against Lattice problems, all of them must be considered when setting parameters.

- Cost of lattice attacks: “estimated,” not “measured.” Accuracy of the estimation is important.

⇨ **Lattice estimator:** Publicly available tool for security estimation for schemes using Lattice problems!

Thanks to *Martin R Albrecht, Rachel Player, Sam Scott, Benjamin Curtis,* and may others..

Lattice Estimator

- Available at:

  https://github.com/malb/lattice-estimator

- Keep updated following new State-of-the-Art attacks.

- Not perfectly correct, but reasonable estimation.

- Not targeting “very” insecure parameters
  - e.g., BKZ block-size $\beta \leq 40$.

Lattice Estimator: Distributions

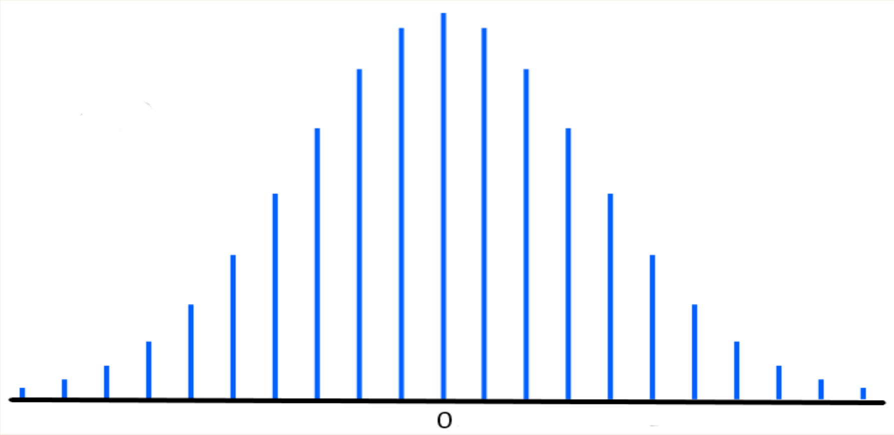
- LWE instance includes two distributions:
  - secret $s \leftarrow \chi_s$
  - error $\mathrm{e} \leftarrow \chi_e$

- Supported distributions *"estimator/nd.py"*
  - Discrete Gaussian
  - Centered Binomial (CBD)
  - Uniform
  - Sparse Binary/Ternary
  - and User-define distributions

Lattice Estimator: Distributions

- Discrete Gaussian Distribution

```
DiscreteGaussian(stddev, mean=0, n=None)
```



- discrete Gaussian with $\sigma =$ stddev, centered at mean
  - tail bound: $\lceil \log_2 n \rceil \sigma$
- Variant: DiscreteGaussianAlpha, where $\alpha q = \sigma$

Lattice Estimator: Distributions

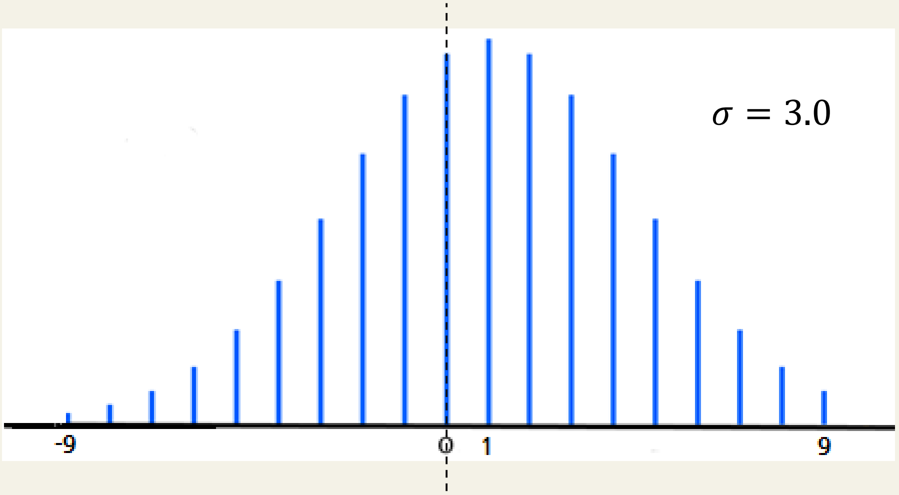
- Discrete Gaussian Distribution (E.g.)

```
>>> from estimator.nd import NoiseDistribution as ND
>>> ND.DiscreteGaussian(3.0, 1.0, 8)
```

- discrete Gaussian with $\sigma = 3.0$, centered at 1.0, tail bound

  $\lceil \log_2 8 \rceil \sigma = 3\sigma = 9.0$

- distribution ranges from -9.0 to 9.0



Lattice Estimator: Distributions

- Centered Binomial (CBD)

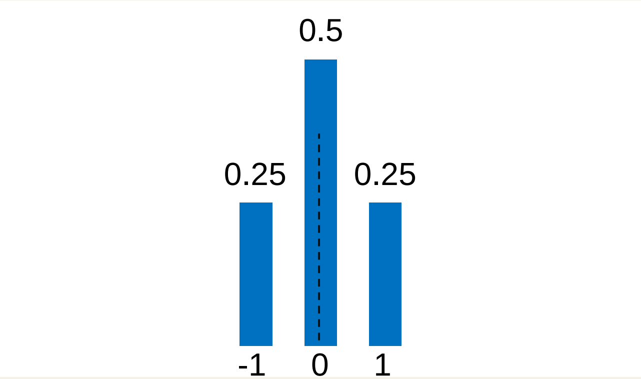
```
CenteredBinomial(eta, n=None)
```

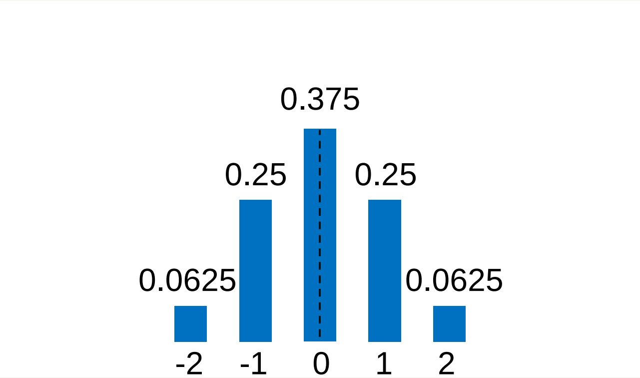
- CBD that samples $a_1, \dots, a_\eta, b_1, \dots, b_\eta$ and return $\Sigma(a_i - b_i)$
  - ranges from - η to η
  - for $k \in [-\eta, \eta]$, probability $= \binom{2\eta}{k+\eta} / 2^{2\eta}$

Lattice Estimator: Distributions

- Centered Binomial (E.g.)

```
>>> ND.CenteredBinomial(1)
>>> ND.CenteredBinomial(2)
```





-1: 1/4, 0: 1/2, 1: 1/4

-2: 1/16, -1: 1/4, 0: 3/8, 1: 1/4, 2: 1/16

Lattice Estimator: Distributions

- Uniform

  Uniform(a, b, n=None)

  - uniform distribution in a *closed* integer interval [a, b]
  - each integer point has probability 1/(b-a+1)

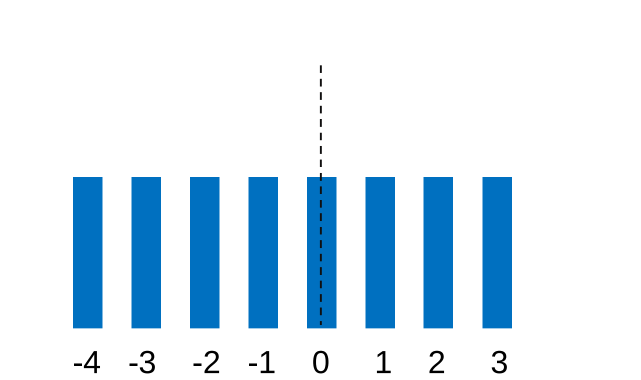
- Variant: UniformMod, which is uniform modulo q
  - ranges [-q/2, q/2]
  - ModSwitch error (from q to p) is UniformMod(q/p)
    - will see in the tutorial!!

Lattice Estimator: Distributions

- Uniform (E.g.)

```
>>> ND.Uniform(-4, 3)
```

- range: $\{-4, -3, -2, -1, 0, 1, 2, 3\}$
- each point has probability 1/8



Lattice Estimator: Distributions

- Sparse Binary/Ternary (Fixed Hamming weight)

SparseTernary(n, p, m=None)

- **Ternary**: length n vector $s \in \mathbb{Z}^n$ that has exactly p "+1"s and m "-1"s, and "0"s for the left:

$$\begin{cases} -1: & m \\ 0: & n-m-p \\ 1: & p \end{cases} \approx \begin{cases} -1: & \frac{m}{n} \\ 0: & 1-\frac{m+p}{n} \\ 1: & \frac{p}{n} \end{cases}$$

- **Binary:** use $m = 0$ (or without an input m)

Lattice Estimator: Distributions

- Sparse Binary/Ternary (Fixed Hamming weight) (E.g.)

```
>>> ND.SparseTernary(256, 64, 64)
>>> ND.SparseTernary(256, 128)
```

- **Ternary**: length=256, Hamming weight = 128

$$\begin{cases} -1: & 64 \\ 0: & 128 \\ 1: & 64 \end{cases}$$

- **Binary:** length=256, Hamming weight = 128

$$\begin{cases} 0: & 128 \\ 1: & 128 \end{cases}$$

Lattice Estimator: Distributions

- User-define distributions *"estimator/nd.py"*

```
@staticmethod
def MyDistribution(n=None):
    # E.g. -1: 1/8, 0: 3/4, 1: 1/8
    # mean = 1/8*(-1) + 3/4*0 + 1/8*1 = 0
    # stddev = sqrt(1/8*(-1-0)**2 + 3/4*(0-0)**2+ 1/8*(1-0)**2) = 1/2
    # density (ratio of non-zero coefficients) = 1/8+1/8 = 1/4
    D = NoiseDistribution(
            n=n,
            stddev=RR(1/2),
            mean=RR(0),
            density=1/RR(4),
            bounds=(-1, 1),
            tag="MyDistribution")
    return D
```

Lattice Estimator: Cost Models

- Supported cost models *“estimator/reduction.py”*
  - ADPS16—New Hope
  - ABFKSW20
  - ABLR21
  - ChaLoy21

    ⋮
- We mainly focus on [ADPS16], with methodology a.k.a. *Core-SVP method*: for BKZ block-size β,

$$\text{Attack cost} \approx \begin{cases} \text{Classical} & : 2^{0.292\beta} \\ \text{Quantum} & : 2^{0.265\beta} \end{cases}$$ (or $2^{0.257\beta}$ following [ChaLoy21])

[ADPS16] Erdem Alkim, Léo Ducas, Thomas Pöppelmann, and Peter Schwabe. Postquantum key exchange – a new hope. In Proceedings of the 25th USENIX Security Symposium, pages 327–343. USENIX Association, 2016.
[ChaLoy21] André Chailloux and Johanna Loyer. Lattice sieving via quantum random walks. In ASIACRYPT 2021, Part IV, volume 13093 of LNCS, pages 63–91. Springer,Heidelberg, December 2021.

Lattice Estimator: Tutorial

- Install Sagemath (already there in 10-10 server: tenten.heaan.info:8000)

- Download lattice estimator from github.com/malb/lattice-estimator or

```
git clone https://github.com/malb/lattice-estimator.git
```

- Make a Python file or a Jupyter notebook in “lattice-estimator” directory using sagemath console.

Lattice Estimator: Tutorial

- Try rough estimation..

```
from estimator import *
from estimator.nd import NoiseDistribution, stddevf
from estimator.lwe_parameters import LWEParameters
from estimator.lwe import estimate
MyParam = LWEParameters(n=256, q=1024,
            Xs=NoiseDistribution.CenteredBinomial(3),
            Xe=NoiseDistribution.SparseTernary(256, 40, 40),
            m=256,
            tag="MyParam",
            )
r = LWE.estimate.rough(MyParam)              # usvp & dual_hybrid only
```

- And full estimation..

```
r = LWE.estimate(MyParam)                    # more attacks
```

Lattice Estimator: Tutorial

- How to read?

Default: MATZOV (classical)

```
usvp :: rop: ≈2^148.7, red: ≈2^148.7, δ: 1.003818, β: 425, d: 879, tag: usvp
dual  :: rop: ≈2^155.9, mem: ≈2^101.3, m: 414, β: 447, d: 926, ↻: 1, tag: dual
```

- Attack costs
  - rop: number of required ring operations
    - i.e. $\geq 2^{148.7}$ and $\geq 2^{155.9}$ ring operations
  - red: rop for lattice reduction
  - β: BKZ block size
  - mem: memory requirement in integers mod q
    - i.e. $\geq 2^{101.3} \cdot \log q$ bits of memory
  - m: number of required samples
  - ↻: required number of expected repetitions

Lattice Estimator: Tutorial

- Try different models:
  - ADPS16 (classical)

```
from estimator.reduction import *
r = LWE.estimate(MyParam, red_cost_model=ADPS16("classical"))
```

β: BKZ block size $\xrightarrow{\text{Core-SVP}}$ $\begin{cases} \text{Classical} : 2^{0.292\beta} \\ \text{Quantum} : 2^{0.257\beta} \end{cases}$

  - ADPS16 (quantum)

```
r = LWE.estimate(MyParam, red_cost_model=ADPS16("quantum"))
```

  - ChaLoy21 (quantum only)

```
r = LWE.estimate(MyParam, red_cost_model=ChaLoy21)
```

Lattice Estimator: Tutorial

- Try SIS:
  - SIS with 2-norm bound

```
params = SIS.Parameters(n=113, q=2048, length_bound=512, norm=2)
SIS.lattice(params)
```

  - SIS with ∞-norm bound

```
params = SIS.Parameters(n=113, q=2048, length_bound=50, norm=oo)
SIS.lattice(params)
```

Lattice Estimator: Tutorial

- Try NTRU:
  - NTRU usvp

```
params = NTRU.Parameters(n=200, q=7981, Xs=ND.UniformMod(3),
Xe=ND.UniformMod(3))
NTRU.primal_usvp(params, red_shape_model="gsa")
NTRU.primal_usvp(params, red_shape_model=Simulator.CN11)
```

  - NTRU hybrid

```
NTRU.primal_hybrid(params, red_shape_model=Simulator.CN11)
```

  - NTRU overstretched parameters

```
params.possibly_overstretched
NTRU.primal_dsd(params, red_shape_model=Simulator.ZGSA)
# or with different model: NTRU.primal_dsd(params,
red_shape_model=Simulator.CN11)
```

Kyber Security

Kyber Security

- IND-CCA security of Kyber.KEM reduces to MLWE instances.

  - NIST's security levels 1, 3, and 5:
    - Target Core-SVP: 2^{120}, 2^{180}, 2^{250}
    - Hash function entropy: 2^{192}, 2^{225}, 2^{257}

Kyber Security

- IND-CCA security of Kyber.KEM reduces to MLWE instances, if there is no decryption failures.
  - NIST's security levels 1, 3, and 5:
    - Target Core-SVP: 2^{120}, 2^{180}, 2^{250}
    - Hash function entropy: 2^{192}, 2^{225}, 2^{257}
- Decryption failure probability
  - Attacks exploiting failures exists [DKRV18][DGJ+19][DB22]
  - It should be low enough: (ideally) $2^{-120}, 2^{-180}, 2^{-250}$

[DB22] D'Anvers, J.P., Batsleer, S.: Multitarget decryption failure attacks and their application to saber and kyber. In PKC 2022, Part I. LNCS, vol. 13177, pp. 3–33. Springer, Heidelberg (Mar 2022). https://doi.org/10.1007/978-3-030-97121-2_1
[DGJ+19] D'Anvers, J.P., Guo, Q., Johansson, T., Nilsson, A., Vercauteren, F., Verbauwhede, I.: Decryption failure attacks on ind-cca secure lattice-based schemes. In: Public-Key Cryptography – PKC 2019. pp. 565–598. Springer International Publishing, Cham (2019)
[DKRV18] D'Anvers, J.P., Karmakar, A., Roy, S.S., Vercauteren, F.: Saber: Module-LWR based key exchange, CPA-secure encryption and CCA-secure KEM. In AFRICACRYPT 18. LNCS, vol. 10831, pp. 282–305. Springer, Heidelberg (May 2018). https://doi.org/10.1007/978-3-319-89339-6_16

Kyber Security

- Kyber Recap:
  - Matrices and vectors over $\mathcal{R}_q = \mathbb{Z}_q[x]/(x^N + 1)$

$$\text{pk: } \left(\overbrace{A}^{k}{}_{k}\ ,\ t = A\,s + e \right)$$

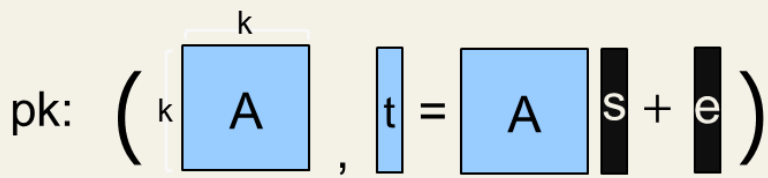
$$\text{(raw) ctxt: } \left(A^t\, r + e_1 \ ,\ t\, r + e_2 + \left\lfloor \frac{q}{2^d} \cdot M \right\rceil \right)$$

$$\text{(compressed) ctxt: } \left(\left\lfloor \frac{2^{d_u}}{q} \cdot A^t\, r + e_1 \right\rceil ,\ \left\lfloor \frac{2^{d_v}}{q} \cdot \left(t\, r + e_2 + \left\lfloor \frac{q}{2^d} \cdot M \right\rceil \right) \right\rceil \right)$$

Kyber Security: MLWE

- **Public key**

Cf. matrices and vectors over $\mathcal{R}_q = \mathbb{Z}_q[x]/(x^N + 1)$



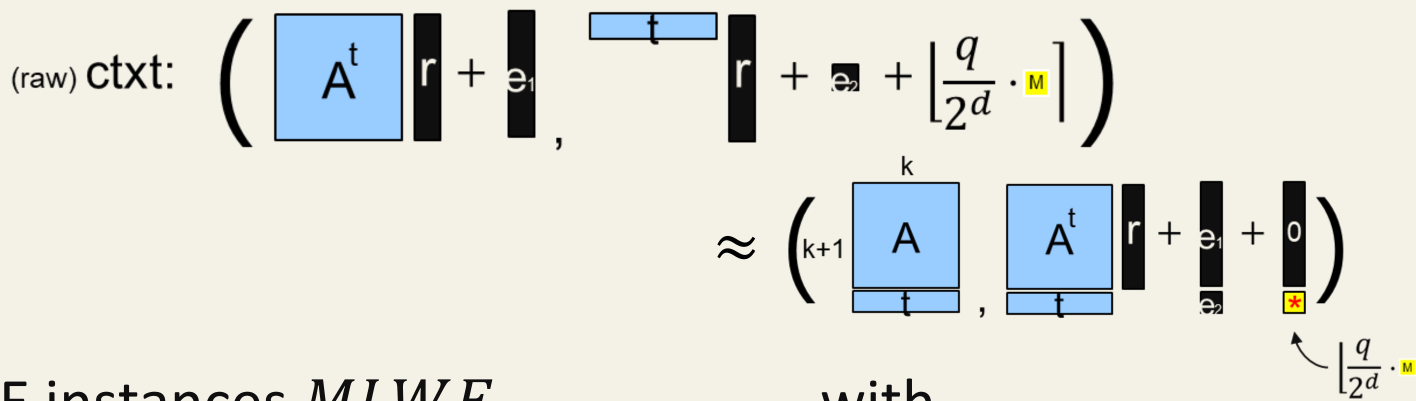
- MLWE instances $MLWE_{n,\,k,\,l,\,q,\,\chi_s,\,\chi_e}$ with
  - $n = 256$ (ring dimension)
  - $k = l = 2, 3, 4$ (depending on security level)
  - $q = 3329$
  - $\chi_s = \mathrm{CBD}(\eta_1),\ \eta_1 = 2$ or 3 (depending on security level)
  - $\chi_e = \mathrm{CBD}(\eta_2),\ \eta_2 = 2$

$\Rightarrow$ LWE instances $LWE_{nk,\,nk,\,q,\,CBD(\eta_1),\,CBD(\eta_1)}$

Kyber Security: MLWE

- **Ciphertext**

Cf. matrices and vectors over $\mathcal{R}_q = \mathbb{Z}_q[x]/(x^N+1)$



- MLWE instances $MLWE_{n,\,k,\,l,\,q,\,\chi_s,\,\chi_e}$ with
  - $l = k+1$
  - for all other parameters, the same as before

$\Rightarrow$ LWE instances $LWE_{nk,\,n(k+1),\,q,\,CBD(\eta_1),\,CBD(\eta_1)}$

Kyber Security: MLWE

```
Kyber512 = LWEParameters(n=256*2, q=3329,
               Xs=NoiseDistribution.CenteredBinomial(3),
               Xe=NoiseDistribution.CenteredBinomial(2),
               m=256*3, tag="Kyber512")

Kyber768 = LWEParameters(n=256*3, q=3329,
               Xs=NoiseDistribution.CenteredBinomial(2),
               Xe=NoiseDistribution.CenteredBinomial(2),
               m=256*4, tag="Kyber512")

Kyber1024 = LWEParameters(n=256*4, q=3329,
               Xs=NoiseDistribution.CenteredBinomial(2),
               Xe=NoiseDistribution.CenteredBinomial(2),
               m=256*5, tag="Kyber1024")

r512 =   LWE.estimate(Kyber512,    red_cost_model=ADPS16("classical"))
r768 =   LWE.estimate(Kyber768,    red_cost_model=ADPS16("classical"))
r1024 =  LWE.estimate(Kyber1024,   red_cost_model=ADPS16("classical"))
```

Kyber Security: DFP

- Decryption Failure Probability
  - Many sources of error:
    - inherent MLWE errors: $\boldsymbol{e}$, $\boldsymbol{e}_1$, e_2
    - compression errors: $\boldsymbol{c}_1$, c_2
      - occurs during modulus switching from q to 2^{d_1} or 2^{d_2}
    - combination of the errors
      - multiplied and added
  - Decryption fails (i.e. not output m) with probability

$$\delta = \Pr\left[\|\boldsymbol{e}^t \cdot \boldsymbol{r} + e_2 + c_2 - \boldsymbol{s}^t \cdot \boldsymbol{e}_1 - \boldsymbol{s}^t \cdot \boldsymbol{c}_1\|_\infty \geq \left\lfloor \frac{q}{4} \right\rceil\right]$$

Kyber Security: DFP

- DFP estimator:
  - available at https://github.com/pq-crystals/security-estimates
  - use their “proba_util.py”

```
import operator as op
from math import factorial as fac
from math import sqrt, log
import sys
from proba_util import *

q=3329
eta1=3
eta2=3
eta3=2                    # ctxt error distribution (CBD(2)) for level 1,
n=256                                                          # otherwise eta2=eta3
k=2
d1=10
d2=4
```

Kyber Security: DFP

$$\delta = \Pr\left[\|\boldsymbol{e}^t \cdot \boldsymbol{r} + e_2 + c_2 - \boldsymbol{s}^t \cdot \boldsymbol{e}_1 - \boldsymbol{s}^t \cdot \boldsymbol{c}_1\|_\infty \geq \left\lfloor \frac{q}{4} \right\rceil\right]$$

- DFP estimator (Cont'd)

```
…
# Initialize secrets and errors
Ds = build_centered_binomial_law(eta1)              # identical to Dr
De = build_centered_binomial_law(eta2)              # identical to De1, De2
De1 = build_centered_binomial_law(eta3)             # identical to De2

# Compression errors
Dc1 = build_mod_switching_error_law(q, 2**d1)
Dc2 = build_mod_switching_error_law(q, 2**d2)

# Combinations
Dc1_e1 = law_convolution(Dc1, De1)                                  # c1+e1
Der = iter_law_convolution(law_product(De, Ds), n*k)                # er
Dse1_sc1 = iter_law_convolution(law_product(Ds, Dc1_e1), n*k)       # se1+sc1

D = law_convolution(Der, Dse1_sc1)                          # er-se1-sc1
D = law_convolution(D, De1)                                 # er+e2-se1-sc1
D = law_convolution(D, Dc2)                                 # final

prob = tail_probability(D, q/4)
print("DFP:", log(n*prob)/log(2))
```

Kyber Security: DFP

```
import operator as op
from math import factorial as fac
from math import sqrt, log
import sys
from proba_util import *

q=3329
eta1=3
eta2=3
eta3=2                                           # ctxt error distribution (CBD(2)) for level 1, otherwise eta2=eta3
n=256
k=2
d1=10
d2=4

# Initialize secrets and errors
Ds = build_centered_binomial_law(eta1)           # equal to Dr
De = build_centered_binomial_law(eta2)           # equal to De1, De2
De1 = build_centered_binomial_law(eta3)          # equal to De2

# Compression errors
Dc1 = build_mod_switching_error_law(q, 2**d1)
Dc2 = build_mod_switching_error_law(q, 2**d2)

# Combinations
Dc1_e1 = law_convolution(Dc1, De1)                              # c1+e1
Der = iter_law_convolution(law_product(De, Ds), n*k)            # er
Dse1_sc1 = iter_law_convolution(law_product(Ds, Dc1_e1), n*k)   # se1+sc1

D = law_convolution(Der, Dse1_sc1)               # er-se1-sc1
D = law_convolution(D, De1)                      # er+e2-se1-sc1
D = law_convolution(D, Dc2)                      # final

prob = tail_probability(D, q/4)
print("DFP:", log(n*prob)/log(2))
```

Kyber Security: DFP

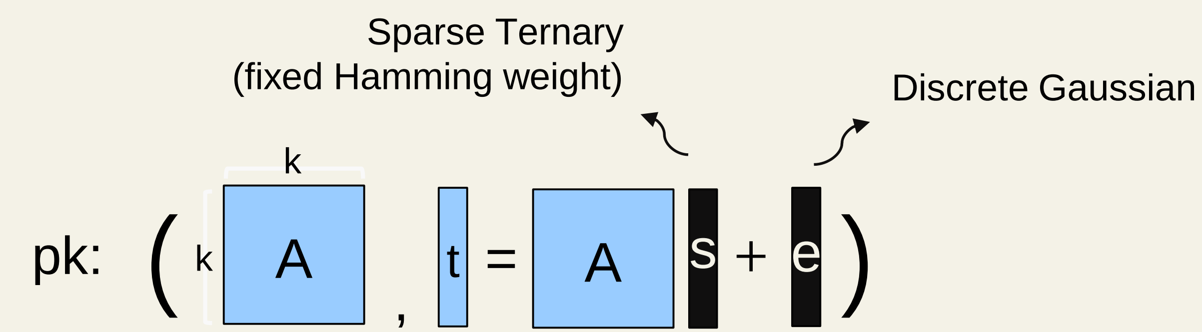
```
ps_light = KyberParameterSet(256, 2, 3, 3, 3329, 2**12, 2**10, 2**4, ke_ct=2)
ps_recommended = KyberParameterSet(256, 3, 2, 2, 3329, 2**12, 2**10, 2**4)
ps_paranoid = KyberParameterSet(256, 4, 2, 2, 3329, 2**12, 2**11, 2**5)
```

Smaug Security

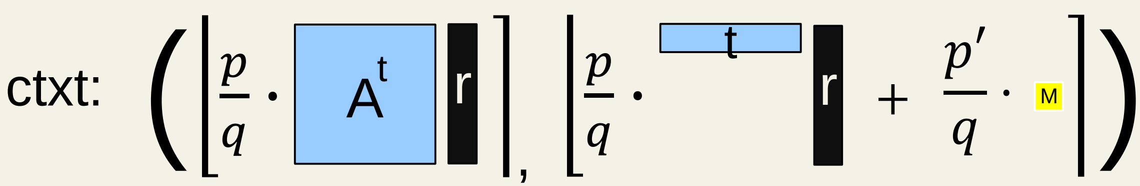
Note, we will take a look on Smaug v4.0, which is not yet public ☺

Smaug Security

- IND-CCA security of Smaug.KEM reduces to MLWE and MLWR instances with low enough DFPs.



$$\Rightarrow LWE_{nk,\, nk,\, q,\, SparseTernary\left(n, \frac{h}{2}, \frac{h}{2}\right),\, dGaussian(\sigma)}$$



$$\Rightarrow LWR_{nk,\, n(k+1),\, q,\, p,\, mCBD}$$

$$\Rightarrow LWE_{nk,\, n(k+1),\, q,\, mCBD,\, UnifMod(\frac{q}{p})}$$

Smaug Security

- Modified CBD, base 16:
  - CBD(1)=mCBD(4): -1: 4/16, 0: 8/16, 1: 4/16
  - mCBD(3): -1: 3/16, 0: 10/16, 1: 3/16
  - mCBD(2): -1: 2/16, 0: 12/16, 1: 2/16
  - mCBD(1): -1: 1/16, 0: 14/16, 1: 1/16

- ModSwitch error (when $p|q$)
  - $a \mapsto a' \coloneqq \left(\left\lfloor \frac{p}{q} \cdot (a \bmod q) \right\rceil \bmod p \right)$
  - $\left(a - \frac{q}{p} \cdot a' \right) \sim Unif\left\{ -\frac{q}{2p} + 1, \cdots, \frac{q}{2p} \right\}$ = UniformMod(q/p)

Smaug Security

- Level 1:
  - n=256, k=2, q=1024, p=256, p'=32
  - Public key:
    - Secret (s): fixed Hamming weight of h=140
      - Ds=SparseTernary(n, 70, 70)
    - Er
      - D
  - Cipl

$$LWE_{nk,\, nk,\, q,\, SparseTernary\left(n, \frac{h}{2}, \frac{h}{2}\right),\, dGaussian(\sigma)} \quad \& \quad LWE_{nk,\, n(k+1),\, q,\, mCBD,\, UnifMod\left(\frac{q}{p}\right)}$$

  - Secret (r): mCBD(2)
    - Need to define this distribution!
  - Error: ModSwitch error from q to p (and p')
    - UniformMod(q/p) and UniformMod(q/p')

Smaug Security: DFP

- Decryption Failure Probability
  - Sources of error:
    - MLWE error:
      - $\boldsymbol{e} \sim$ dGaussian(sigma)
    - MLWR ModSwitch errors:
      - $\boldsymbol{e}_1 \sim$ UniformMod(q/p)
      - $e_2 \sim$ UniformMod(q/p’)
  - Decryption fails (i.e. not output m) with probability

$$\delta = \Pr\left[\|\boldsymbol{e}^t \cdot \boldsymbol{r} + e_2 - \boldsymbol{s}^t \cdot \boldsymbol{e}_1\|_\infty \geq \frac{q}{4}\right],$$

where

- $\boldsymbol{s} \sim$ SparseTernary(nk, h/2, h/2),
- $\boldsymbol{r} \sim$ mCBD(2)

Smaug Security: Tutorial Guideline 1

- User-define modules for mCBD

CBD(1)=mCBD(4):	-1: 4/16, 0: 8/16, 1: 4/16
mCBD(3):	-1: 3/16, 0: 10/16, 1: 3/16
mCBD(2):	-1: 2/16, 0: 12/16, 1: 2/16
mCBD(1):	-1: 1/16, 0: 14/16, 1: 1/16

  - ‘ModifiedCBD(numCBD, n=None)’ in “estimator/nd.py”
    - for security estimation
  - ‘build_mCBD_law(numCBD)’ in “security-estimates/proba_util.py”
    - for DFP calculation

Smaug Security: Tutorial Guideline 2

- Security Estimation
  - Lvl 1.
    - n=256, k=2, q=1024, p=256, p'=32, sig=1.0625, h=140, numCBD=2
  - Lvl 3.
    - n=256, k=3, q=2048, p=512, p'=16, sig=1.0625, h=264 , numCBD=4
  - Lvl 5.
    - n=256, k=4, q=2048, p=512, p'=128, sig=1.0625, h=348, numCBD=3

$$LWE_{nk,\,nk,\,q,\,SparseTernary(nk,\,h/2,\,h/2),\,dGaussian(\sigma)}$$
$$\&\quad LWE_{nk,\,n(k+1),\,q,\,mCBD,\,UnifMod(q/p)}$$

Smaug Security: Tutorial Guideline 3

- Decryption Failures
  - MLWE error $\boldsymbol{e} \sim$ dGaussian(sigma)
  - MLWR ModSwitch errors:
    - $\boldsymbol{e}_1 \sim$ UniformMod(q/p)
    - $e_2 \sim$ UniformMod(q/p’)
  - Secret vectors:
    - $\boldsymbol{s} \sim$ SparseTernary(nk, h/2, h/2),
    - $\boldsymbol{r} \sim$ mCBD(*)

$$\delta = \Pr\left[\|\boldsymbol{e}^t \cdot \boldsymbol{r} + e_2 - \boldsymbol{s}^t \cdot \boldsymbol{e}_1\|_\infty \geq \frac{q}{4}\right]$$

Thank You!