# Kyber_Estimation_1024

April 12, 2024

## 1 Kyber Estimation

Estimate the drop of bit security caused by multiple information leaked from power side channel.

[1]:
```
!ls
```

```
Kyber1024_bikz.pdf   Kyber768_qbit.pdf              modular_result512.txt
Kyber1024_qbit.pdf   Kyber_Estimation.ipynb         modular_result768.txt
Kyber512_bikz.pdf   'Kyber_Estimation_64*3.ipynb'   stdout.txt
Kyber512_qbit.pdf    PlotDrops.ipynb                untitled.txt
Kyber768_bikz.pdf    modular_result1024.txt         untitled1.txt
```

[2]:
```
load("../framework/instance_gen.sage")
```

[21]:
```
build_centered_binomial_law(2)
```

[21]:
```
{-2: 0.0625000000000000,
 -1: 0.250000000000000,
 0: 0.375000000000000,
 1: 0.250000000000000,
 2: 0.0625000000000000}
```

[4]:
```
## LWattackstance initilizaiton


KYBER_K =4

 ## default as 768

ntt_n = 256

n = KYBER_K *ntt_n
m = n
q = 3329
# D_e = build_centered_binomial?


bit_security_constant = 0.292
```

```
if KYBER_K ==3 or KYBER_K ==4:
    D_e = build_centered_binomial_law(2)
    # D_e = {-2: 0.0625, -1: 0.25, 0: 0.375, 1: 0.25, 2: 0.0625}
elif KYBER_K == 2 or KYBER_K ==1 :
    D_e = build_centered_binomial_law(2)
    # D_e = build_centered_binomial_law(3)
    # D_e = {-2:0.093754,-1:0.2343754,2:0.093754,1:0.2343754, 0:0.31255, -3:␣
 ↪(1- 0.31255 -0.2343754*2 -0.093754 *2)/2, 3: (1- 0.31255 -0.2343754*2 -0.
 ↪093754 *2)/2}
else:
    assert("The input KYBER_N must be in {2,3,4} corresponding to {Kyber_512,␣
 ↪768, 1024}")

D_s = D_e
A, b, dbdd = initialize_from_LWE_instance(DBDD_predict, n, q, m, D_e, D_s)
# _ = dbdd.integrate_q_vectors(q, report_every=20)
beta, delta = dbdd.estimate_attack()
```

```
      Build DBDD from LWE
n=1024    m=1024          q=3329
      Attack Estimation
dim=2049          =1.002255       =877.44
```

[5]:
```
beta*0.292
```

[5]:
```
256.212123610483
```

[6]:
```
R = IntegerModRing(3329)
V = VectorSpace(R,ntt_n)

import numpy as np

def bit_reverse(x):return 2*int( "0b" + bin(x)[2:].rjust(7,'0')[::-1] ,2)+1

NTT_matrix = []

def add (x,y) : return x +y

for x in range(ntt_n/2):
    NTT_matrix.append(V(reduce(add, [[(17)^(x*bit_reverse(i)),0] for i in␣
 ↪range(ntt_n/2)])))
    NTT_matrix.append(V(reduce(add, [[0,(17)^(x*bit_reverse(i))] for i in␣
 ↪range(ntt_n/2)])))

NTT_matrix = matrix(NTT_matrix)
```

```
inv_NTT_matrix = NTT_matrix^-1
```

[ ]:

[7]:
```
s = [[0 for i in range(ntt_n)] for j in range(KYBER_K)]

for i in range(KYBER_K):
    for j in range(ntt_n):
        v0 = [0 for i in range(m + n)]
        v0[i*ntt_n+j]=1
        s[i][j] = dbdd.leak(v0)
```

[8]:
```
s?
```

```
Type:           list
String form:    [[-1, 1, 1, 1, 0, 0, -1, -1, 0, 0, -1, -1, -1, 0, -1, 0, 2, -1,␣
 ↪-1, -1, 0, 1, 1, 1, 0, -2, 0, -1, <…> 1, 0, 0, -1, 0, 0, -1, -1, 1, 0, -1, -1,␣
 ↪2, 0, 0, -1, 1, -1, 0, 1, -1, 0, 0, 0, 1, -1, 0, -2, 0]]
Length:         4
File:
Docstring:
Built-in mutable sequence.

If no argument is given, the constructor creates a new empty list. The
argument must be an iterable if specified.
Init docstring: Initialize self.  See help(type(self)) for accurate signature.
```

[9]:
```
s_hat_list = [V(i)*NTT_matrix for i in s]
```

[10]:
```
# Hint_list
s_relation_list = [ [R(s_hat[2*i])/R(s_hat[2*i+1]) for i in range(ntt_n/2)] ␣
 ↪for s_hat in s_hat_list]

v_list = []
v_list_1 = []
for j in range(KYBER_K):
    for i in range(ntt_n/2):
        s_relation = s_relation_list[j]
        k = s_relation[i]
        v = list((NTT_matrix.column(2*i) - k*NTT_matrix.column(2*i+1)))
        v = [0]*int(ntt_n*j) + v + [0]*int(ntt_n*(KYBER_K-j-1))
        v_prime  = [int(i) for i in list(v) + [0]*(m)]

        v_prime= vec(v_prime)
        v_list.append(v_prime)
        v_list_1.append(v)
        # print(dbdd.leak(v_prime)%3329)
```

[11]:
```
from tqdm import tqdm,tqdm_notebook
```

[12]:
```
# for i in tqdm(range(10)):
#     sleep(1)
```

[13]:
```
# import logging
# import sys
# import datetime

# def init_logger(filename, logger_name):
#     '''
#     @brief:
#         initialize logger that redirect info to a file just in case we lost␣
 ↪connection to the notebook
#     @params:
#         filename: to which file should we log all the info
#         logger_name: an alias to the logger
#     '''

#     # get current timestamp
#     timestamp = datetime.datetime.utcnow().strftime('%Y%m%d_%H-%M-%S')

#     logging.basicConfig(
#         level=logging.INFO,
#         format='[%(asctime)s] %(name)s {%(filename)s:%(lineno)d}␣
 ↪%(levelname)s - %(message)s',
#         handlers=[
#             logging.FileHandler(filename=filename),
#             logging.StreamHandler(sys.stdout)
#         ]
#     )

#     # Test
#     logger = logging.getLogger(logger_name)
#     logger.info('### Init. Logger {} ###'.format(logger_name))
#     return logger

# # Initialize
# my_logger = init_logger("./ml_notebook.log", "ml_logger")
```

[14]:
```
# sys.stdout = open('stdout.txt','w')

for v0 in tqdm(v_list):
    a=dbdd.leak(v0)%3329
    if a!=0:
```

```
        assert("a!=0")
    dbdd.integrate_modular_hint(v0,0,3329)
```

```
100%|
                          | 512/512 [2:37:25<00:00, 18.45s/it]
```

[18]:
```
beta_1, delta_1 = dbdd.estimate_attack()
```

[20]:
```
beta_1*0.292
```

[20]:
```
163.468746212081
```

[15]:
```
# dbdd
```

[15]:
```
<__main__.DBDD_predict object at 0x7ec94a8270d0>
```

[16]:
```
# dbdd.float_type="qd"
# # dbdd.float_type="mpfr"
```

[ ]:
```

```