Quantum speedups for lattice sieves are tenuous at best

ePrint: 2019/1161

Martin R. Albrecht, Vlad Gheorghiu,
Eamonn W. Postlethwaite, John M. Schanck

October 18, 2019
Not this talk:
The security of Kyber768.
Not this talk:
The security of Kyber768.
Not this talk:
The security of Kyber768.
The **lattice security** of Kyber768.
Not this talk:
The security of Kyber768.
The **lattice security** of Kyber768.
The **cost of BKZ-k for k that determines** the lattice security of Kyber768.
Not this talk:
The security of Kyber768.
The **lattice security** of Kyber768.
The **cost of BKZ-k for k that determines** the lattice security of Kyber768.
The **core-SVP estimate for** the lattice security of Kyber768.
Not this talk:
The security of Kyber768.
The lattice security of Kyber768.
The cost of BKZ-k for k that determines the lattice security of Kyber768.
The core-SVP estimate for the lattice security of Kyber768.
The cost of the sieving routine inside the SVP solver used in the core-SVP estimate for the lattice security of Kyber768.
Not this talk:
The security of Kyber768.
The **lattice security** of Kyber768.
The **cost of BKZ-k for k that determines** the lattice security of Kyber768.
The **core-SVP estimate for** the lattice security of Kyber768.
The **cost of the sieving routine** inside the SVP solver used in the core-SVP estimate for the lattice security of Kyber768.
The **heuristic cost of the sieving routine** inside the SVP solver used in the core-SVP estimate for the lattice security of Kyber768.
Not this talk:
The security of Kyber768.
The **lattice security** of Kyber768.
The **cost of BKZ-k for k that determines** the lattice security of Kyber768.
The **core-SVP estimate for** the lattice security of Kyber768.
The **cost of the sieving routine** inside the SVP solver used in the core-SVP estimate for the lattice security of Kyber768.
The **heuristic cost of the sieving routine** inside the SVP solver used in the core-SVP estimate for the lattice security of Kyber768.
The **heuristic cost of one call to the sieving routine** inside the SVP solver used in the core-SVP estimate for the lattice security of Kyber768.
Not this talk:
The security of Kyber768.
The **lattice security** of Kyber768.
The **cost of BKZ-k for k that determines** the lattice security of Kyber768.
The **core-SVP estimate for** the lattice security of Kyber768.
The **cost of the sieving routine** inside the SVP solver used in the core-SVP estimate for the lattice security of Kyber768.
The **heuristic cost of the sieving routine** inside the SVP solver used in the core-SVP estimate for the lattice security of Kyber768.
The **heuristic cost of one call to the sieving routine** inside the SVP solver used in the core-SVP estimate for the lattice security of Kyber768.

This talk:
The heuristic cost—classical and quantum—of near neighbor search on spheres in dimension $< 1000$. 

Cost estimates and numerically optimized parameters for the heuristic NNS algorithms underlying:

- Nguyen–Vidick sieve
- bgj1, i.e. Becker–Gama–Joux sieve w/o recursion
- The Becker–Ducas–Gama–Laarhoven sieve
Near neighbor search

A near neighbor search algorithm takes a list of $N$ points, pre-processes it to make neighbor queries more efficient.

I want to find points that are close to $u$ in angular distance.

▶ Angular distance: $\theta(u, v) = \arccos\langle u, v \rangle$.

I want to do this for many different $u$. 
Near neighbor search

A near neighbor search algorithm takes a list of $N$ points, pre-processes it to make neighbor queries more efficient.

I want to find points that are close to $u$ in angular distance.

- Angular distance: $\theta(u, v) = \arccos\langle u, v \rangle$.

I want to do this for many different $u$. 
List-size preserving parameterization

Special case:
- Input consists of $N$ uniformly random points.
- $N$ large enough to ensure that there are $N$ neighboring pairs.

Write $C_d(\theta)$ for the spherical measure of

$$\text{Cap}(u, \theta) = \{ v : \theta(u, v) \leq \theta \}.$$

Then

$$N \approx \binom{N}{2} C_d(\theta),$$

equiv.

$$N \approx \frac{2}{C_d(\theta)}$$
List-size preserving parameterization

Special case:

- Input consists of $N$ uniformly random points.
- $N$ large enough to ensure that there are $N$ neighboring pairs.

Write $C_d(\theta)$ for the spherical measure of

$$\text{Cap}(u, \theta) = \{v : \theta(u, v) \leq \theta\}.$$ 

Then

$$N \approx \binom{N}{2} C_d(\theta),$$
equiv.

$$N \approx \frac{2}{C_d(\theta)}$$
Algorithm: AllPairs / Nguyen–Vidick sieve

Input: list $L$ of size $N$.

Search:
1. Number the points $v_1, v_2, v_3, \ldots, v_N$
2. Test $\theta(v_i, v_j) \leq \theta$ for $1 \leq i < j \leq N$
Cost of AllPairs / Nguyen–Vidick sieve
List-size preserving case

**Classical search**
Nguyen–Vidick (2008): \( \left( \frac{1}{C_d(\theta)} \right)^{2+o(1)} \)

\[
\left( \frac{1}{C_d(\pi/3)} \right)^{2+o(1)} = 2^{c(d)} \quad \text{where} \quad c(d) = (0.4150 \ldots + o(1))d
\]

**Quantum search**
Laarhoven–Mosca–van de Pol (2014): \( \left( \frac{1}{C_d(\theta)} \right)^{1.5+o(1)} \)

\[
\left( \frac{1}{C_d(\pi/3)} \right)^{1.5+o(1)} = 2^{c(d)} \quad \text{where} \quad c(d) = (0.3112 \ldots + o(1))d
\]
Cost of AllPairs / Nguyen–Vidick sieve
List-size preserving case

**Classical search**
Nguyen–Vidick (2008): \( (1/C_d(\theta))^{2+o(1)} \)

\[
(1/C_d(\pi/3))^{2+o(1)} = 2^{c(d)} \text{ where } c(d) = (0.4150\ldots + o(1))d
\]

**Quantum search**
Laarhoven–Mosca–van de Pol (2014): \( (1/C_d(\theta))^{1.5+o(1)} \)

\[
(1/C_d(\pi/3))^{1.5+o(1)} = 2^{c(d)} \text{ where } c(d) = (0.3112\ldots + o(1))d
\]
\[ \log_2(\#\text{ops}) - 0.4150d \]

\[ \log_2(\#\text{ops}) - 0.3112d \]
Why care about the polynomial terms?

- Quantum and classical variants have *different* polynomial factors.
- Quantum advantage is small. Even smaller in more advanced algorithms.
- Polynomial factors are significant in low dimension.
Why care about the polynomial terms?

- Quantum and classical variants have *different* polynomial factors.
- Quantum advantage is small. Even smaller in more advanced algorithms.
- Polynomial factors are significant in low dimension.
Why care about the polynomial terms?

- Quantum and classical variants have *different* polynomial factors.
- Quantum advantage is small. Even smaller in more advanced algorithms.
- Polynomial factors are significant in low dimension.
Why care about the polynomial terms?

- Quantum and classical variants have different polynomial factors.
- Quantum advantage is small. Even smaller in more advanced algorithms.
- Polynomial factors are significant in low dimension.
What are the polynomial factors?

- Volume estimates.
- Cost of testing $\theta(u,v)$. 
What are the polynomial factors?

- Volume estimates.
- Cost of testing $\theta(u,v)$. 
Search predicates

- **Search predicate on \( \mathcal{X} \):**
  \[
  f : \mathcal{X} \to \{0, 1\}
  \]

- **Kernel of \( f \):**
  \[
  \text{Ker}(f) = \{x : f(x) = 0\}
  \]
  \[
  |f| = |\text{Ker}(f)|
  \]

- **Predicate \( f \cap g \) defined by:**
  \[
  \text{Ker}(f \cap g) = \text{Ker}(f) \cap \text{Ker}(g)
  \]
Exhaustive search

\[ g(1) \quad g(2) \quad g(3) \quad g(4) \quad g(5) \quad \ldots \quad \ldots \]
Exhaustive search

1  g(2)  g(3)  g(4)  g(5)  ...  ...
Exhaustive search

|   |   | g(3) | g(4) | g(5) | ... | ... |
Exhaustive search

1 1 1 1 g(4) g(5) ... ...
Exhaustive search

1 1 1 1 1 1 g(5) ... ...

...
Exhaustive search

1 1 1 1 1 1 1

... ...
Exhaustive search

1 1 1 1 1 1 1 1 ... g(57)
Exhaustive search

1 1 1 1 1 1 1 ... 0
Filtered search

\[ f(1) \quad f(2) \quad f(3) \quad f(4) \quad f(5) \quad \ldots \quad f(57) \]
\[ g(1) \quad g(2) \quad g(3) \quad g(4) \quad g(5) \quad \ldots \quad g(57) \]
Filtered search

\[
\begin{array}{cccccccc}
1 & 1 & f(3) & f(4) & f(5) & \ldots & f(57) \\
g(1) & g(2) & g(3) & g(4) & g(5) & \ldots & g(57)
\end{array}
\]
## Filtered search

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th>f(5)</th>
<th>...</th>
<th>f(57)</th>
</tr>
</thead>
<tbody>
<tr>
<td>g(1)</td>
<td>g(2)</td>
<td>g(3)</td>
<td>g(4)</td>
<td>g(5)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Filtered search

\[\begin{array}{ccccccccc}
1 & 1 & 1 & 1 & 1 & 1 & \cdots & f(57) \\
g(1) & g(2) & g(3) & g(4) & g(5) & \cdots & g(57)
\end{array}\]
Filtered search

\[
\begin{array}{cccccccc}
1 & 1 & 1 & 1 & 1 & 1 & \ldots & 0 \\
g(1) & g(2) & g(3) & g(4) & g(5) & \ldots & g(57)
\end{array}
\]
Filtered search

\[
\begin{array}{ccccccccc}
1 & 1 & 1 & 1 & 1 & 1 & \ldots & 0 \\
g(1) & g(2) & g(3) & g(4) & g(5) & \ldots & 0 \\
\end{array}
\]
Quantum search

For any predicate $g$ and unitary $A$, define the amplification operator:

$$G(A, g) := AR_0A^\dagger R_g$$

where

$$R_0 |x\rangle = \begin{cases} -|x\rangle & \text{if } x = 0 \\ |x\rangle & \text{otherwise} \end{cases}$$

$$R_g |x\rangle = (-1)^{g(x)} |x\rangle.$$
Quantum search

Suppose that measuring $A |0\rangle$ yields an element of $\text{Ker}(g)$ with probability $p$.

Grover–Brassard–Høyer–Mosca–Tapp:

$\text{Measuring } G(\mathcal{A}, g)^k \mathcal{A} |0\rangle$

with $k \approx \sqrt{1/p}$ yields a root of $g$ w.p. $\approx 1\ldots$

Boyer–Brassard–Høyer–Tapp:

\ldots even if $p$ is not known.
Quantum search

Suppose that measuring $A |0\rangle$ yields an element of $\text{Ker}(g)$ with probability $p$.

Grover–Brassard–Høyer–Mosca–Tapp:

- Measuring $G(A, g)^k A |0\rangle$
  
  with $k \approx \sqrt{1/p}$ yields a root of $g$ w.p. $\approx 1$.

Boyer–Brassard–Høyer–Tapp:

- ... even if $p$ is not known.
Quantum search

Suppose that measuring $A |0\rangle$ yields an element of $\text{Ker}(g)$ with probability $p$.

Grover–Brassard–Høyer–Mosca–Tapp:

- Measuring $G(A, g)^k A |0\rangle$

  with $k \approx \sqrt{1/p}$ yields a root of $g$ w.p. $\approx 1$.

Boyer–Brassard–Høyer–Tapp:

- ... even if $p$ is not known.
Filtered quantum search

Parameters $m_1$ and $m_2$.

1. Sample $j$ uniformly from $\{0, \ldots, m_1 - 1\}$
2. Sample $k$ uniformly from $\{0, \ldots, m_2 - 1\}$
3. Define

$$A_j = G(D, f)^jD$$
$$B_k = G(A_j, f \cap g)^k$$

4. Prepare and measure the state:

$$B_k A_j |0\rangle$$
Cost of filtered quantum search

Suppose that we know $P/\gamma \leq |g| \leq \gamma P$.

Proposition

We can choose $m_1$ and $m_2$ such that FilteredQuantumSearch finds a root of $f \cap g$ with probability at least $1/8$ and has a cost that is dominated by (approximately)

- $\gamma^{1/2} \sqrt{N}$ times the cost of $G(g)$, or
- $\frac{4}{3} \sqrt{\gamma P}$ times the cost of $R_{f \cap g}$. 
Cost of filtered quantum search

Suppose that we know $P/\gamma \leq |g| \leq \gamma P$.

**Proposition**

We can choose $m_1$ and $m_2$ such that $\text{FilteredQuantumSearch}$ finds a root of $f \cap g$ with probability at least $1/8$ and has a cost that is dominated by (approximately)

- $\gamma \frac{1}{2} \sqrt{N}$ times the cost of $G(g)$, or
- $\frac{4}{3} \sqrt{\gamma P}$ times the cost of $R_{f \cap g}$. 
Cost of filtered quantum search

Suppose that we know $P/\gamma \leq |g| \leq \gamma P$.

Proposition

We can choose $m_1$ and $m_2$ such that FilteredQuantumSearch finds a root of $f \cap g$ with probability at least $1/8$ and has a cost that is dominated by (approximately)

- $\gamma^{\frac{1}{2}} \sqrt{N}$ times the cost of $G(g)$, or
- $\frac{4}{3} \sqrt{\gamma P}$ times the cost of $R_{f \cap g}$. 
Cost of filtered quantum search

Suppose that we know $P/\gamma \leq |g| \leq \gamma P$.

**Idealized Proposition**

We can choose $m_1$ and $m_2$ such that FilteredQuantumSearch finds a root of $f \cap g$ and has a cost that is dominated by

- $\frac{1}{2} \sqrt{N}$ times the cost of $G(g)$, or
- $\frac{4}{3} \sqrt{P}$ times the cost of $R_{f \cap g}$.
Algorithm: AllPairs / Nguyen–Vidick sieve

Input: list $L$ of size $N$

1. Number the points $v_1, v_2, v_3, \ldots, v_N$
2. For $i = 1, \ldots, N$
3. For $j = i + 1, \ldots, N$
4. Test $g_i(v_j)$ where $g_i(v_j) = [\theta(v_i, v_j) > \pi/3]$. 
Algorithm: AllPairs / Nguyen–Vidick sieve

Input: list $L$ of size $N$

1. Number the points $v_1, v_2, v_3, \ldots, v_N$
2. For $i = 1, \ldots, N$
3. For $j = i + 1, \ldots, N$
4. If $f_i(v_j)$ then test $g_i(v_j)$ where $g_i(v_j) = [\theta(v_i, v_j) > \pi/3]$.
Algorithm: AllPairs / Nguyen–Vidick sieve

Input: list \( L \) of size \( N \)

1. Number the points \( v_1, v_2, v_3, \ldots, v_N \)
2. For \( i = 1, \ldots, N \)
3. For \( j = i + 1, \ldots, N \)
4. If \( f_i(v_j) \) then test \( g_i(v_j) \) where \( g_i(v_j) = [\theta(v_i, v_j) > \pi/3] \).

What to use for \( f_i \) in a filtered search?
Define a hash function family:

\[ \mathcal{H} = \{ u \mapsto \text{sgn}( \langle r, u \rangle ) : r \in S \} \]
XOR + population count

Fact: \[ \Pr_{h \leftarrow \mathcal{H}} [h(u) \neq h(v)] = \frac{\theta(u, v)}{\pi}. \]

Let \( H_n(x) = (h_1(x), \ldots, h_n(x)) \) for random \( h_i \in \mathcal{H} \).

For large \( n \), we have

\[
\frac{\text{HammingWeight}(H_n(u) \oplus H_n(v))}{n} \approx \frac{\theta(u, v)}{\pi}
\]
Fact: \[
\Pr[h(u) \neq h(v)] = \frac{\theta(u, v)}{\pi}.
\]

Let \( H_n(x) = (h_1(x), \ldots, h_n(x)) \) for random \( h_i \in \mathcal{H} \).

For large \( n \), we have
\[
\frac{\text{HammingWeight}(H_n(u) \oplus H_n(v))}{n} \approx \frac{\theta(u, v)}{\pi}
\]
**Fact:**  
\[
\Pr_{h \leftarrow \mathcal{H}}[h(u) \neq h(v)] = \frac{\theta(u, v)}{\pi}.
\]

Let \( H_n(x) = (h_1(x), \ldots, h_n(x)) \) for random \( h_i \in \mathcal{H} \).

For large \( n \), we have  
\[
\frac{\text{HammingWeight}(H_n(u) \oplus H_n(v))}{n} \approx \frac{\theta(u, v)}{\pi}
\]
XOR + population count

Used as a filter in implementations of sieving algorithms:

- 2018 Ducas

Earlier algorithmic use

- 1995 Goemans–Williamson
- 2002 Charikar
Algorithm: AllPairs / Nguyen–Vidick sieve

Input: list $L$ of size $N$

Setup:
1. Fix $H_n$
2. Construct a table $(i, H_n(v_i))$

Search:
For all $i$:
1. Load $H_n(v_i)$
2. For $j = i + 1, \ldots, N$
3. Load $H_n(v_j)$
4. If $\text{HammingWt}(H_n(v_i) \oplus H_n(v_j)) \leq k$
5. Test $\theta(v_i, v_j) \leq \theta$. 
This work: New python/mpmath package

Calculates the circuit depth, width, gate count (etc.) for popcount and filtered quantum search subroutines.

Calculates the accuracy of random popcount filters given
▶ points uniformly distributed on sphere;
▶ points uniformly distributed in a cap of angle $\beta$.

Calculates the (normalized) spherical measure of
▶ caps, using $_2F_1$ representation of $C_d(\theta)$
▶ intersections of caps, using an integral representation.
\[ \log_2(\#\text{ops}) \]

- **AllPair (c: RAM)**
  - \[ 0.4150d \]

- **AllPair (q: depth-width)**
  - \[ 0.3112d \]
Error correction

Image: Fowler, Mariantoni, Martinis, Cleland. (2012)
Error correction

We consider the added cost of reading syndromes, but not processing them.

(Under the same physical assumptions as Gidney–Ekera (2019))
Algorithm: RandomBucketSearch / bg.j1

Parameters: $t$, $\theta_1$

Input: list $L$ of size $N$

Search:
1. Repeat $t$ times:
2. Pick a random point $f$.
3. Run AllPairs on $L_f = L \cap Cap(f, \theta_1)$.

Note: Optimal choice of $t$ and $\theta_1$ is based on volume of the intersection of caps of angle $\theta_1$ with centers at distance $\pi/3$. 
Cost of RandomBucketSearch
List-size preserving case

**Classical search**
Albrecht–Ducas–Herold–Kirshanova–Postlethwaite–Stevens

\[ 2^{c(d)} \text{ where } c(d) = (0.3494 \ldots + o(1))d \]

**Quantum search**

\[ 2^{c(d)} \text{ where } c(d) = (0.3013 \ldots + o(1))d \]
\[ \log_2(\# \text{ops}) \]

- \(0.3494 \cdot d\)
- \(0.3013 \cdot d\)
RandomBucket (c: RAM)

0.3494d

RandomBucket (q: depth-width)

0.3013d
**RandomBucket (c: RAM)**

**RandomBucket (q: GE19)**

$$\log_2(\#\text{ops})$$

- $0.3494d$
- $0.3013d$
Algorithm: ListDecodingSearch / BDGL

Parameters: $t$, $\theta_1$, $\theta_2$

Input: list $L$ of size $N$

Setup:
Pick a set of $t$ random points $F$
Initialize $t$ buckets $\{L_f : f \in F\}$

Fill:
1. For each $v$ in $L$
2. insert $v$ into $L_f$ if $f \in \text{Cap}(v, \theta_2)$

Query:
1. For each $v$ in $L$
2. $F_i = F \cap \text{Cap}(v, \theta_1)$
3. Run AllPairs on $L_F = \bigsquare\{L_f : f \in F_i\}$. 
Cost of ListDecodingSearch / BDGL

**Classical search**
Becker–Ducas–Gama–Laarhoven:

\[ 2^{c(d)} \text{ where } c(d) = (0.2924 \ldots + o(1))d \]

**Quantum search**
Laarhoven:

\[ 2^{c(d)} \text{ where } c(d) = (0.2652 \ldots + o(1))d \]
\[ \log_2(\text{#ops}) \]

- \[ 0.2924d \]
- \[ 0.2652d \]
\[
\log_2(\# \text{ops})
\]

- ListDecoding (c: RAM)
- \(0.2924 \, d\)
- ListDecoding (q: depth-width)
- \(0.2652 \, d\)
ListDecoding (c: RAM)

ListDecoding (q: GE19)

\[ \log_2(\text{#ops}) \]

\[ d \]

- \[ 0.2924d \]
- \[ 0.2652d \]
Barriers to a quantum speedup

qRAM

- Known constructions have some cost that grows like $N^{O(1)}$.
- qRAM computations are not necessarily “localizable”.
Barriers to a quantum speedup

Error correction overhead

- Cost of processing syndromes
- Cost of state distillation
- Locality constraints introduced by code
- Probability of failure from logical errors
Barriers to a quantum speedup

Poor parallelization
ListDecoding (q: depth per search)
Cost underestimates

- “Idealized proposition”:
  \[ \frac{P}{\gamma} \leq |g| \leq \gamma P; \quad \Pr[\text{success}] \geq 1/8. \]

- Use of \( G(H, f) \).
  “Run AllPairs on \( L_F = \bigsqcup \{B_f : f \in F_i\} \).”
\[ \log_2(\# \text{ops}) \]

\[ \pi d / 10 \]