Another Look at the Quantum Security of the Vectorization Problem with Shifted Inputs

Joint work with P. Frixons, V. Gilchrist, P. Kutas, S. Merz, C. Petit*, L. Pham

ePrint 2025/376

**slides inspired by Christophe's. Thanks!*

Vectorization problem with shifted inputs

Group actions are attractive for building post-quantum crypto

$$\star : G \times X \to X$$

If G is commutative, then DH follows naturally

Vectorization problem with shifted inputs

Underlying this scheme is the *Vectorization* problem

The Vectorization problem :

Given x and $g \star x$, recover g.

-“core” DLP problem

-underlies CSIDH

Vectorization problem with shifted inputs

The Vectorization problem with shifted inputs:

Given $x, (c_i, [c_i]g \star x)$ and $g \star x$, recover g.

-variant of vectorization that publishes more information

-underlies CSI-SharK and BCP

How does the security compare to pure vectorization?

CSIDH

Choose a prime p

Let $X = \{$supersingular curves defined over $\mathbb{F}_p\}$ (up to isomorphism)

For $E \in X$ let $End_{\mathbb{F}_p}(E)$ be the set of endomorphisms defined over $\mathbb{F}_p$

Let G be the class group of $End_{\mathbb{F}_p}(E)$

CSIDH

Choose a prime p

Let $X = \{\text{supersingular curves defined over } \mathbb{F}_p\}$ (up to isomorphism)

For $E \in X$ let $End_{\mathbb{F}_p}(E)$ be the set of endomorphisms defined over $\mathbb{F}_p$

Let G be the class group of $End_{\mathbb{F}_p}(E)$

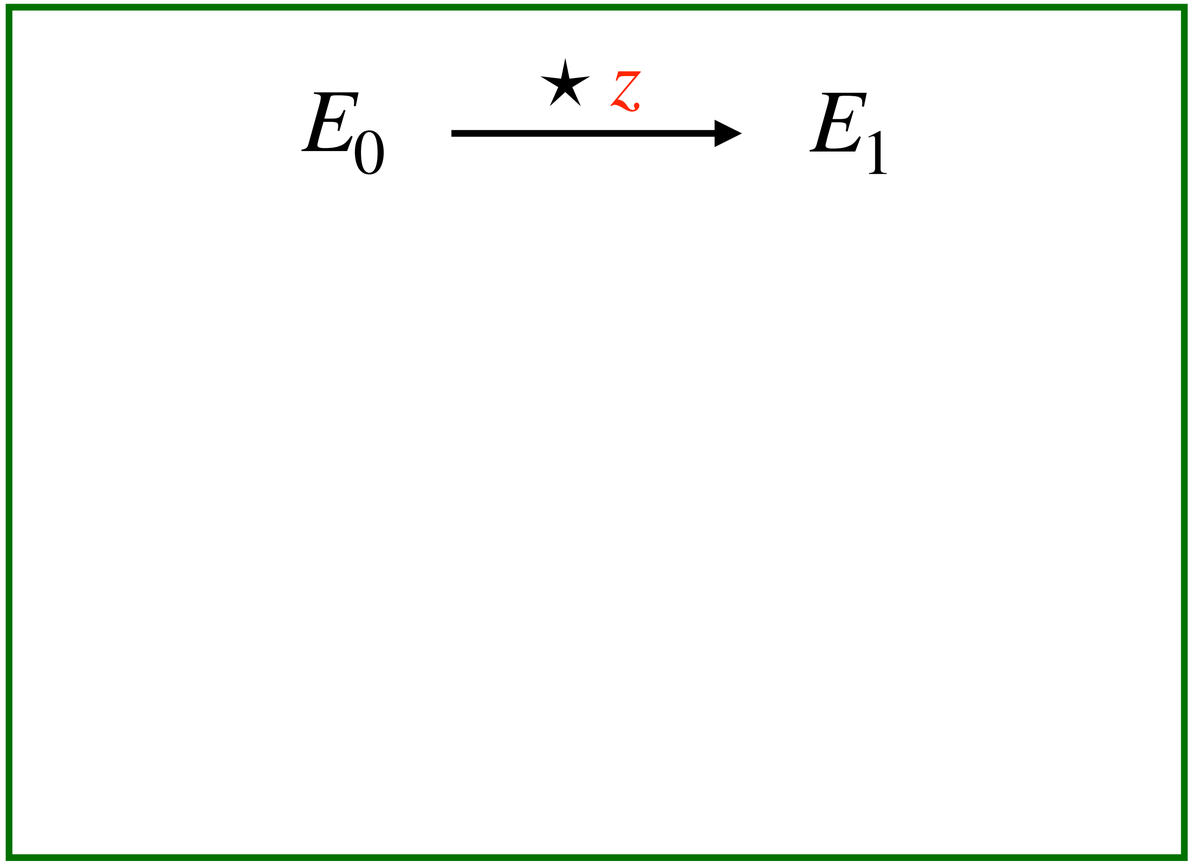
let g be a generator of G

Then we can identify G with $(\mathbb{Z}_N, +)$ where $N := |G|$

$$g^z \star E = E'$$

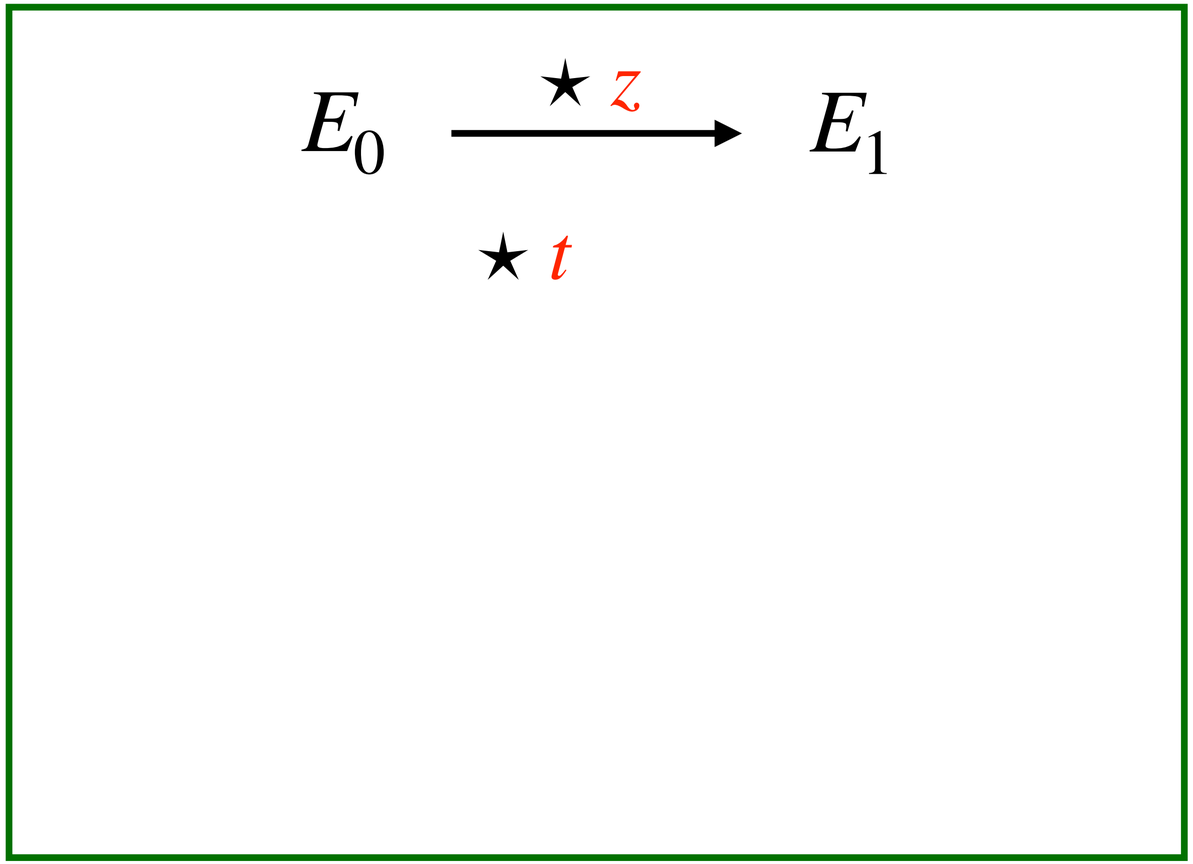
CSI-SharK

Based on the CSI-FiSh signature with id protocol:



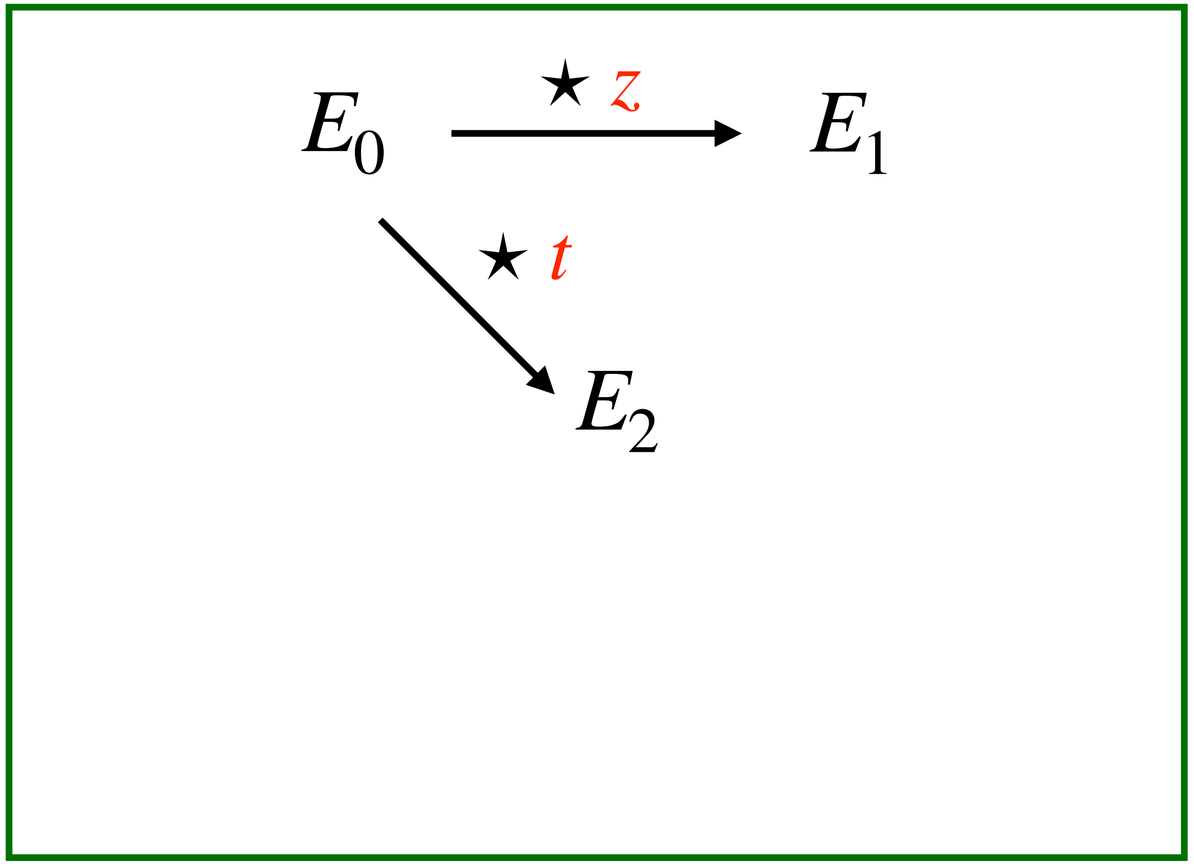
CSI-SharK

Based on the CSI-FiSh signature with id protocol:



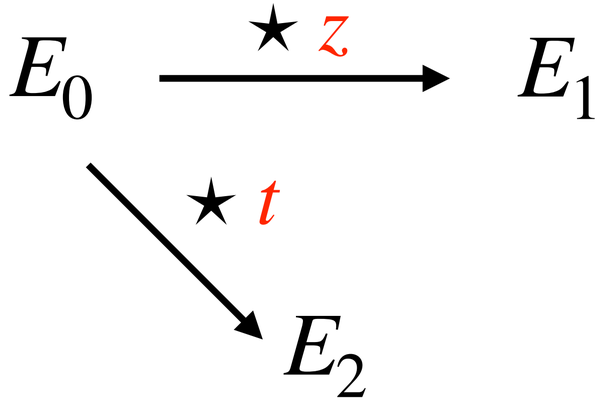
CSI-SharK

Based on the CSI-FiSh signature with id protocol:



CSI-SharK

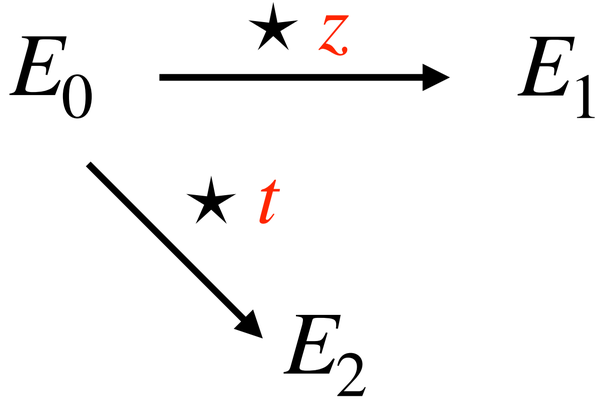
Based on the CSI-FiSh signature with id protocol:



-the verifier sends challenge bit b

CSI-SharK

Based on the CSI-FiSh signature with id protocol:



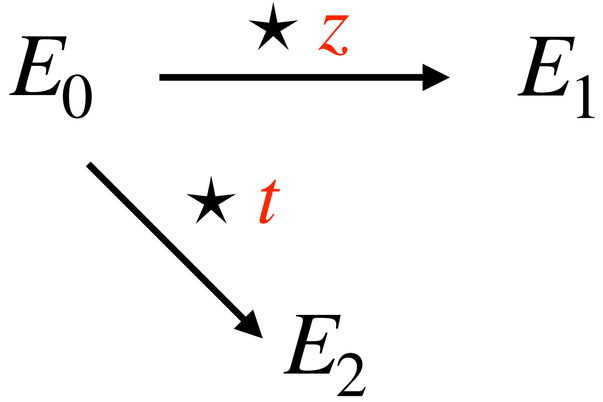
-the verifier sends challenge bit b

-the prover sends u such that

$$E_2 = u \star E_b$$

CSI-SharK

Based on the CSI-FiSh signature with id protocol:



-the verifier sends challenge bit b

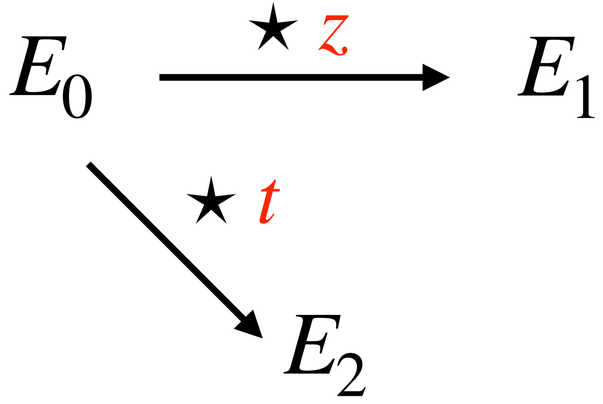
-the prover sends u such that

$$E_2 = u \star E_b$$

-soundness error 1/2

CSI-SharK

Based on the CSI-FiSh signature with id protocol:



-the verifier sends challenge bit b

-the prover sends u such that

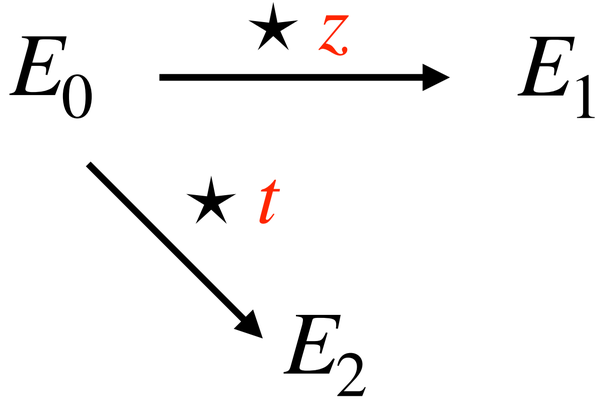
$$E_2 = u \star E_b$$

-soundness error 1/2

-use multiple secret keys z_i to reduce soundness

CSI-SharK

Based on the CSI-FiSh signature with id protocol:



-the verifier sends challenge bit b

-the prover sends u such that

$E_2 = u \star E_b$

-soundness error 1/2

-use multiple secret keys z_i to reduce soundness

-use **related keys** e.g. $z_i = c_i z$ to reduce key size and facilitate secret sharing

CSI-SharK security

Given $(E_i = g^{c_i z} \star E_0, c_i)$ for several i, compute z.

$c_0 = 0$, and it was suggested to use $c_i = i$

CSI-SharK security

Given $(E_i = g^{c_i z} \star E_0, c_i)$ for several i, compute z.

$c_0 = 0$, and it was suggested to use $c_i = i$

Using only (E_0, E_j) for some $j \neq 0$ is exactly Vectorization

CSI-SharK security

Given $(E_i = g^{c_i z} \star E_0, c_i)$ for several i, compute z.

$c_0 = 0$, and it was suggested to use $c_i = i$

Using only (E_0, E_j) for some $j \neq 0$ is exactly Vectorization

If $c_i \mid N$ then the problem can be reduced to a subgroup problem

CSI-SharK security

Given $(E_i = g^{c_i z} \star E_0, c_i)$ for several i, compute z.

$c_0 = 0$, and it was suggested to use $c_i = i$

Using only (E_0, E_j) for some $j \neq 0$ is exactly Vectorization

If $c_i \mid N$ then the problem can be reduced to a subgroup problem

If N is prime, then is the problem as hard as Vectorization?

Quantum security of CSIDH

Shor does not apply and Grover is too slow

Instead, we can frame it as a **hidden shift problem**:

> Let $f_0, f_1 : G \to X$, such that for some z, we have $f_1(x) = f_0(x - z)$ for all x.
>
> Given black box access to f_0, f_1, compute z.

Quantum security of CSIDH

Shor does not apply and Grover is too slow

Instead, we can frame it as a **hidden shift problem**:

> Let $f_0, f_1 : G \to X$, such that for some z, we have $f_1(x) = f_0(x - z)$ for all x.
>
> Given black box access to f_0, f_1, compute z.

For CSIDH we may define functions

> $f_0 \colon g' \mapsto g' \star (g^z \star E)$,
> $f_1 \colon g' \mapsto g' \star E$

Kuperberg

High level recipe :

1. Create and label objects

$$y, |f(y)\rangle$$

2. “Combine” “good” objects

$$(x - y), |f(x - y)\rangle$$

3. Extract

Kuperberg

Example [1]: $N = 2^n$

1. Create and label objects

2. “Combine” “good” objects

3. Extract

[1] Regev. *A subexponential time algorithm for the dihedral hidden subgroup problem with polynomial space.* 2004

Kuperberg

Example [1]: $N = 2^n$

1. Create and label objects

$$y, |0\rangle + e^{(2\pi i/N)zy} |1\rangle \text{ } (z \text{ is secret})$$

2. “Combine” “good” objects

3. Extract

[1] Regev. *A subexponential time algorithm for the dihedral hidden subgroup problem with polynomial space.* 2004

[1] Regev. *A subexponential time algorithm for the dihedral hidden subgroup problem with polynomial space.* 2004

Kuperberg

Example [1]: $N = 2^n$

1. Create and label objects

$y, |0\rangle + e^{(2\pi i/N)zy} |1\rangle$ (z is secret)

2. “Combine” “good” objects

Via tensoring : $y_2 - y_1, |0\rangle + e^{(2\pi i/N)z(y_2 - y_1)} |1\rangle$

3. Extract

[1] Regev. *A subexponential time algorithm for the dihedral hidden subgroup problem with polynomial space.* 2004

Kuperberg

Example [1]: $N = 2^n$

1. Create and label objects

$y, |0\rangle + e^{(2\pi i/N)zy} |1\rangle$ (z is secret)

2. “Combine” “good” objects

Via tensoring : $y_2 - y_1, |0\rangle + e^{(2\pi i/N)z(y_2 - y_1)} |1\rangle$

3. Extract

After enough repetitions, we get a label equal to 2^{n-1}, so :

$$|0\rangle + e^{(2\pi i/N)z2^{n-1}} |1\rangle = |0\rangle + e^{\pi i z} |1\rangle$$

Measuring (in the Hadamard basis) gives z

Cost of running Kuperberg

Subexponential complexity

Cost of a quantum attack on CSIDH was estimated by Peikert

— One group action evaluation is expensive,
and requires sub exponentially many

Childs-van Dam algorithm

Context

Suppose access to pairs $(E_i = g^{iz} \star E_0, i)$ for $i \in [0,M]$

Context

Suppose access to pairs $(E_i = g^{iz} \star E_0, i)$ for $i \in [0,M]$

In the CSIDH setting, we can *twist* efficiently: i.e. compute $E_i^t := g^{-iz} \star E_0 = E_{-i}$

Context

Suppose access to pairs $(E_i = g^{iz} \star E_0, i)$ for $i \in [0,M]$

In the CSIDH setting, we can *twist* efficiently: i.e. compute $E_i^t := g^{-iz} \star E_0 = E_{-i}$

This implies access to pairs $(E_i = g^{iz} \star E_0, i)$ for $i \in [-M, M]$

Context

Suppose access to pairs $(E_i = g^{iz} \star E_0, i)$ for $i \in [0,M]$

In the CSIDH setting, we can *twist* efficiently: i.e. compute $E_i^t := g^{-iz} \star E_0 = E_{-i}$

This implies access to pairs $(E_i = g^{iz} \star E_0, i)$ for $i \in [-M, M]$

Define $f : [-M, M] \times \mathbb{Z}_N \to X$

$$(i, x) \mapsto g^x \star E_{-i}$$

Context

Suppose access to pairs $(E_i = g^{iz} \star E_0, i)$ for $i \in [0,M]$

In the CSIDH setting, we can *twist* efficiently: i.e. compute $E_i^t := g^{-iz} \star E_0 = E_{-i}$

This implies access to pairs $(E_i = g^{iz} \star E_0, i)$ for $i \in [-M, M]$

Define $f : [-M, M] \times \mathbb{Z}_N \to X$

$$(i, x) \mapsto g^x \star E_{-i}$$

$$\implies f(i, x) = g^x \star E_{-i}$$

Context

Suppose access to pairs $(E_i = g^{iz} \star E_0, i)$ for $i \in [0,M]$

In the CSIDH setting, we can *twist* efficiently: i.e. compute $E_i^t := g^{-iz} \star E_0 = E_{-i}$

This implies access to pairs $(E_i = g^{iz} \star E_0, i)$ for $i \in [-M, M]$

Define $f : [-M, M] \times \mathbb{Z}_N \to X$

$$(i, x) \mapsto g^x \star E_{-i}$$

$$\implies f(i,x) = g^x \star E_{-i} \ = g^x \star (g^{-iz} \star E_0)$$

Context

Suppose access to pairs $(E_i = g^{iz} \star E_0, i)$ for $i \in [0,M]$

In the CSIDH setting, we can *twist* efficiently: i.e. compute $E_i^t := g^{-iz} \star E_0 = E_{-i}$

This implies access to pairs $(E_i = g^{iz} \star E_0, i)$ for $i \in [-M, M]$

Define $f : [-M, M] \times \mathbb{Z}_N \to X$

$$(i, x) \mapsto g^x \star E_{-i}$$

$$\implies f(i,x) = g^x \star E_{-i} \ = g^x \star (g^{-iz} \star E_0) = g^{x-iz} \star E_0$$

Context

Suppose access to pairs $(E_i = g^{iz} \star E_0, i)$ for $i \in [0,M]$

In the CSIDH setting, we can *twist* efficiently: i.e. compute $E_i^t := g^{-iz} \star E_0 = E_{-i}$

This implies access to pairs $(E_i = g^{iz} \star E_0, i)$ for $i \in [-M, M]$

Define $f : [-M, M] \times \mathbb{Z}_N \to X$

$$(i, x) \mapsto g^x \star E_{-i}$$

$$\implies f(i,x) = g^x \star E_{-i} \; = g^x \star (g^{-iz} \star E_0) = g^{x-iz} \star E_0 = f(0, x - iz)$$

Context

Suppose access to pairs $(E_i = g^{iz} \star E_0, i)$ for $i \in [0,M]$

In the CSIDH setting, we can *twist* efficiently: i.e. compute $E_i^t := g^{-iz} \star E_0 = E_{-i}$

This implies access to pairs $(E_i = g^{iz} \star E_0, i)$ for $i \in [-M, M]$

Define $f : [-M, M] \times \mathbb{Z}_N \to X$

$$(i, x) \mapsto g^x \star E_{-i}$$

"generalized" hidden shift

$$\implies f(i,x) = g^x \star E_{-i} = g^x \star (g^{-iz} \star E_0) = g^{x-iz} \star E_0 = f(0, x - iz)$$

Childs-van Dam algorithm

1. Create and label objects

Apply **BuildSuperposition** to get (with known label, y_j)

$$\sum_{i_1,\ldots i_k \in [-M,M]} \omega^{\sum_j i_j y_j z} \,|\, i_1, \ldots i_k \rangle$$

Childs-van Dam algorithm

2. “Combine” “good” objects

Compute $\alpha := \sum_j i_j y_j \bmod N$ in a new register to get

$$\sum_{i_1,\ldots i_k \in [-M,M]} \omega^{\alpha z} \,|\, i_1, \ldots i_k\rangle \,|\, \alpha\rangle$$

Childs-van Dam algorithm

2. “Combine” “good” objects

Compute $\alpha := \sum_j i_j y_j \bmod N$ in a new register to get

$$\sum_{i_1,\ldots i_k \in [-M,M]} \omega^{\alpha z} \,|\, i_1, \ldots i_k\rangle \,|\, \alpha\rangle$$

Apply **Knapsack** to get a state close to

$$\sum_{\alpha \in \mathbb{Z}_N} \omega^{\alpha z} \,|\, 0,\ldots 0\rangle \,|\, \alpha\rangle$$

Childs-van Dam algorithm

3. Extract

Apply QFT inverse over $\mathbb{Z}_N$ on last register to get a state close to

$$|0,\ldots 0\rangle\,|z\rangle$$

Measure last register and check answer

Subroutines

BuildSuperposition : can be computed using QFTs

Subroutines

BuildSuperposition : can be computed using QFTs

Knapsack : CvD solves using integer programming

-asymptotic complexity, out-dated solution

Knapsack

Recall,

Given $y_1, \ldots y_k, \alpha, N$, we want to compute

$i_1, \ldots i_k \in [-M, M]$ such that

$$\sum_j i_j y_j = \alpha \bmod N$$

-infinity norm

-average case v.s. worst case

-target α in superposition

-the y_j classical, known

Knapsack

When $\alpha = 0$,

Knapsack

When $\alpha = 0$,

$$\Lambda_0(\mathbf{y}) := \{\mathbf{x} \in \mathbb{Z}^k : \langle \mathbf{x}, \mathbf{y} \rangle = 0 \bmod N\}$$

Knapsack

When $\alpha = 0$,

$$\Lambda_0(\mathbf{y}) := \{\mathbf{x} \in \mathbb{Z}^k : \langle \mathbf{x}, \mathbf{y} \rangle = 0 \bmod N\}$$

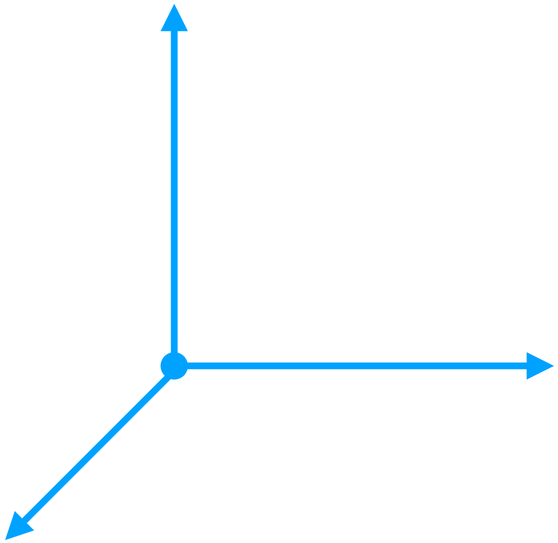
We want $S_\alpha^{\mathbf{y}} := \Lambda_0(\mathbf{y}) \cap [-M, M]^k$

Knapsack

When $\alpha = 0$,

$$\Lambda_0(\mathbf{y}) := \{\mathbf{x} \in \mathbb{Z}^k : \langle \mathbf{x}, \mathbf{y} \rangle = 0 \bmod N\}$$

We want $S_\alpha^{\mathbf{y}} := \Lambda_0(\mathbf{y}) \cap [-M, M]^k$

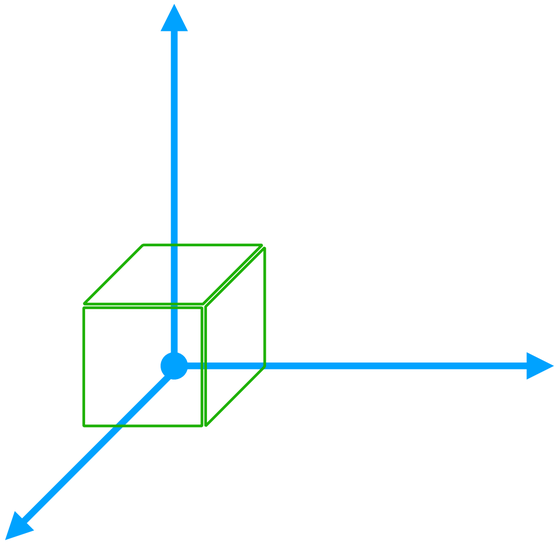


Knapsack

When $\alpha = 0$,

$$\Lambda_0(\mathbf{y}) := \{\mathbf{x} \in \mathbb{Z}^k : \langle \mathbf{x}, \mathbf{y} \rangle = 0 \bmod N\}$$

We want $S_\alpha^{\mathbf{y}} := \Lambda_0(\mathbf{y}) \cap [-M, M]^k$

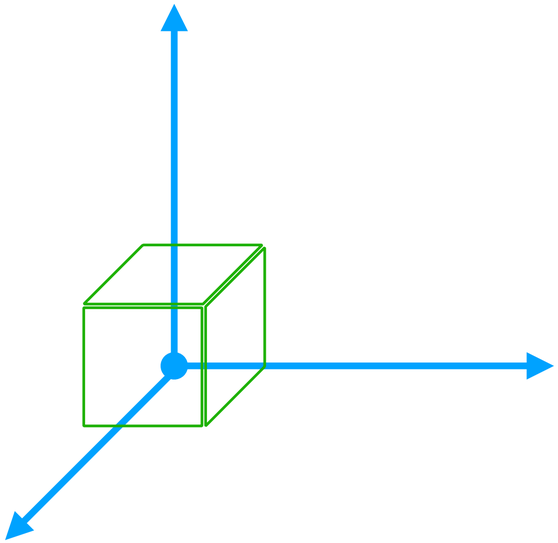


Knapsack

When $\alpha = 0$,

$$\Lambda_0(\mathbf{y}) := \{\mathbf{x} \in \mathbb{Z}^k : \langle \mathbf{x}, \mathbf{y} \rangle = 0 \bmod N\}$$

We want $S_\alpha^{\mathbf{y}} := \Lambda_0(\mathbf{y}) \cap [-M, M]^k$



Shortest vector problem!

Knapsack

In general,

$$\Lambda_\alpha(\mathbf{y}) := \{\mathbf{x} \in \mathbb{Z}^k : \langle \mathbf{x}, \mathbf{y} \rangle = \alpha \bmod N\}$$

Knapsack

In general,

$$\Lambda_\alpha(\mathbf{y}) := \{\mathbf{x} \in \mathbb{Z}^k : \langle \mathbf{x}, \mathbf{y} \rangle = \alpha \bmod N\}$$

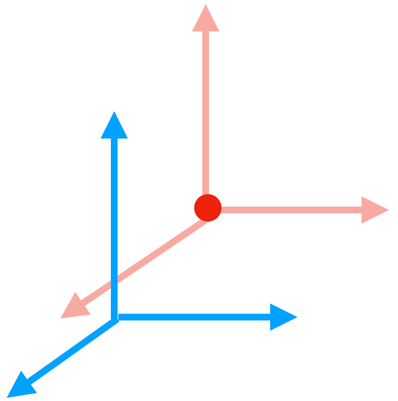
is a lattice **coset** (not a lattice!)

Knapsack

In general,

$$\Lambda_\alpha(\mathbf{y}) := \{\mathbf{x} \in \mathbb{Z}^k : \langle \mathbf{x}, \mathbf{y} \rangle = \alpha \bmod N\}$$

is a lattice **coset** (not a lattice!)



Knapsack

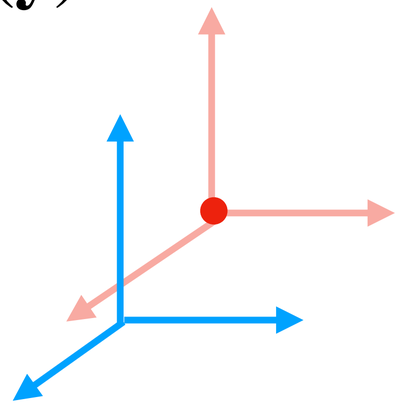
In general,

$$\Lambda_\alpha(\mathbf{y}) := \{\mathbf{x} \in \mathbb{Z}^k : \langle \mathbf{x}, \mathbf{y} \rangle = \alpha \bmod N\}$$

is a lattice **coset** (not a lattice!)

We can characterize $\Lambda_\alpha(\mathbf{y})$ using only one solution, $x_\alpha \in \Lambda_\alpha(\mathbf{y})$:

$$\Lambda_\alpha(\mathbf{y}) = x_\alpha + \Lambda_0(\mathbf{y})$$



Knapsack

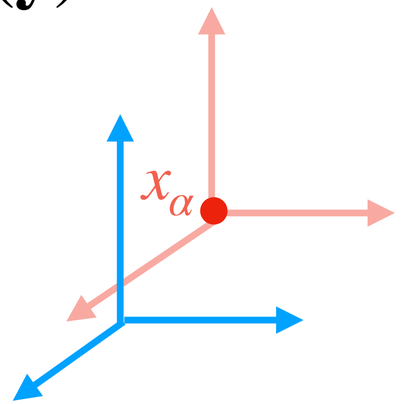
In general,

$$\Lambda_\alpha(\mathbf{y}) := \{\mathbf{x} \in \mathbb{Z}^k : \langle \mathbf{x}, \mathbf{y} \rangle = \alpha \bmod N\}$$

is a lattice **coset** (not a lattice!)

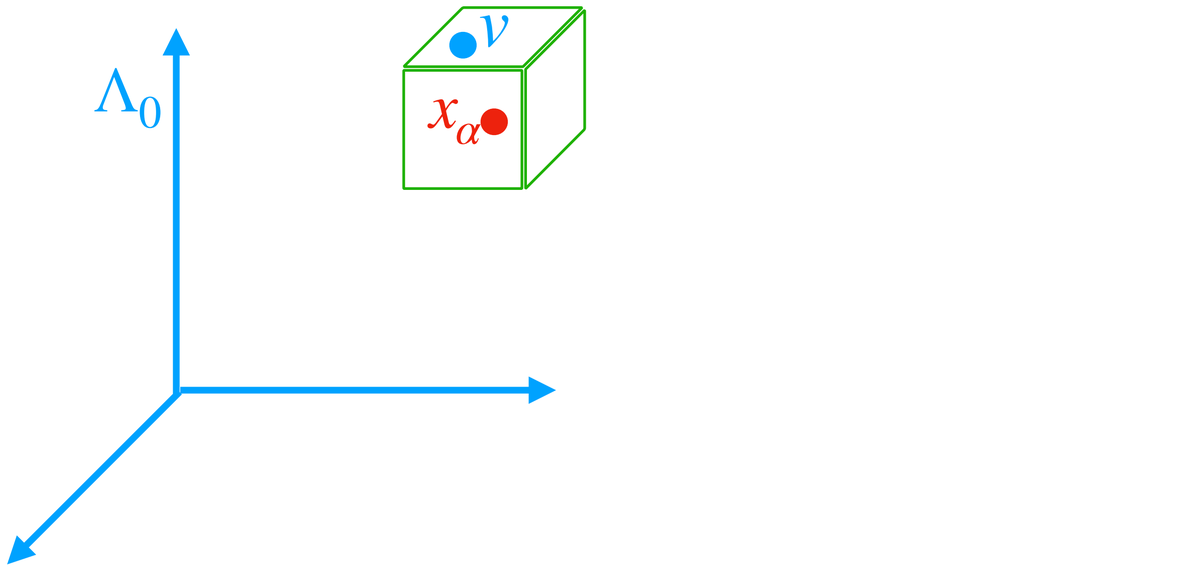
We can characterize $\Lambda_\alpha(\mathbf{y})$ using only one solution, $x_\alpha \in \Lambda_\alpha(\mathbf{y})$:

$$\Lambda_\alpha(\mathbf{y}) = x_\alpha + \Lambda_0(\mathbf{y})$$



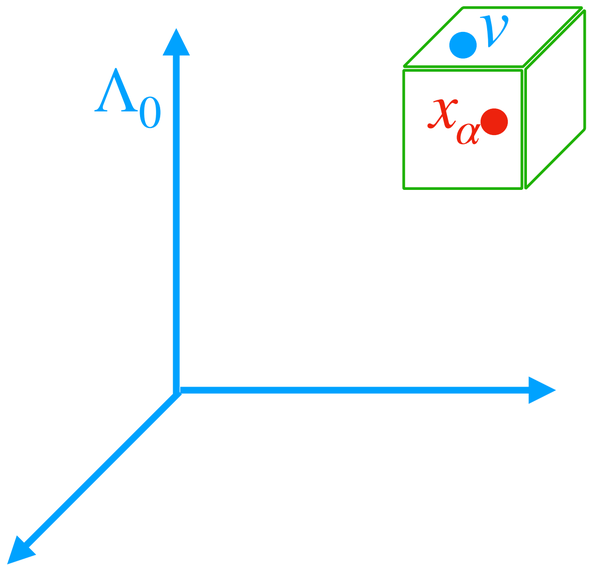
Knapsack

Claim : a “short” solution in $\Lambda_\alpha(\mathbf{y})$ is the same as a vector in $\Lambda_0(\mathbf{y})$ “close” to x_α



Knapsack

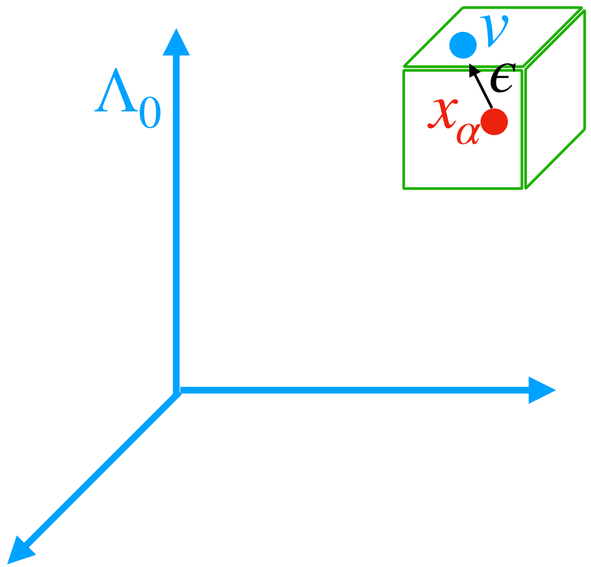
Claim : a “short” solution in $\Lambda_\alpha(\mathbf{y})$ is the same as a vector in $\Lambda_0(\mathbf{y})$ “close” to x_α



Let $v \in (x_\alpha + [-M, M]^k) \cap \Lambda_0(\mathbf{y})$

Knapsack

Claim : a “short” solution in $\Lambda_\alpha(\mathbf{y})$ is the same as a vector in $\Lambda_0(\mathbf{y})$ “close” to x_α

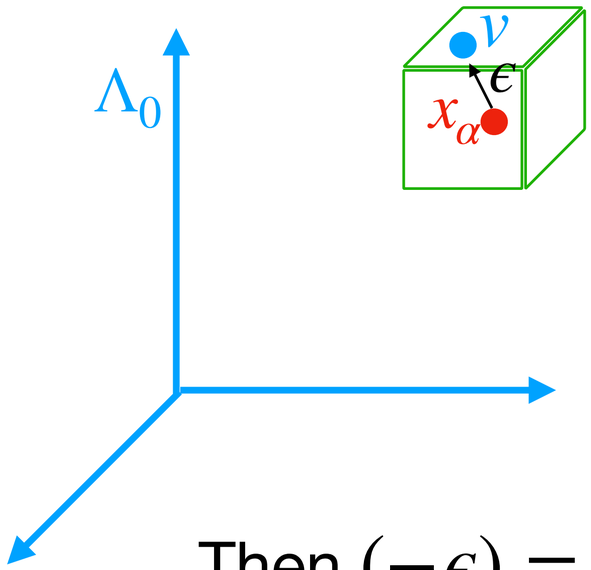


Let $v \in (x_\alpha + [-M, M]^k) \cap \Lambda_0(\mathbf{y})$

Write $v = x_\alpha + \epsilon,$ with $\epsilon \in [-M, M]^k$

Knapsack

Claim : a “short” solution in $\Lambda_\alpha(\mathbf{y})$ is the same as a vector in $\Lambda_0(\mathbf{y})$ “close” to x_α



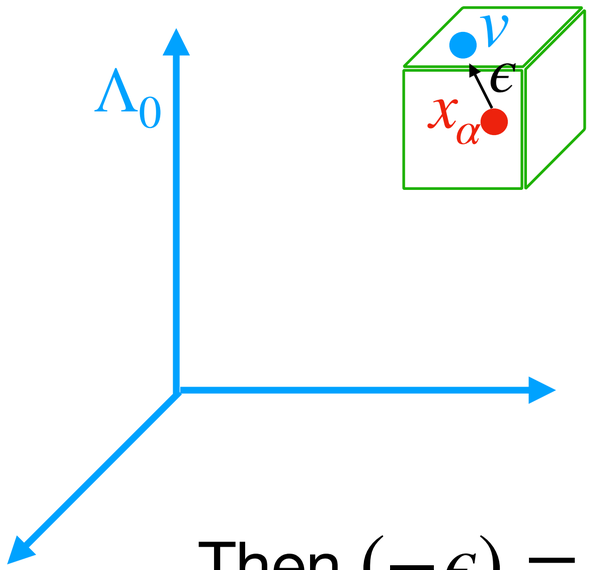
Let $v \in (x_\alpha + [-M, M]^k) \cap \Lambda_0(\mathbf{y})$

Write $v = x_\alpha + \epsilon,$ with $\epsilon \in [-M, M]^k$

Then $(-\epsilon) = x_\alpha - v$

Knapsack

Claim : a “short” solution in $\Lambda_\alpha(\mathbf{y})$ is the same as a vector in $\Lambda_0(\mathbf{y})$ “close” to x_α



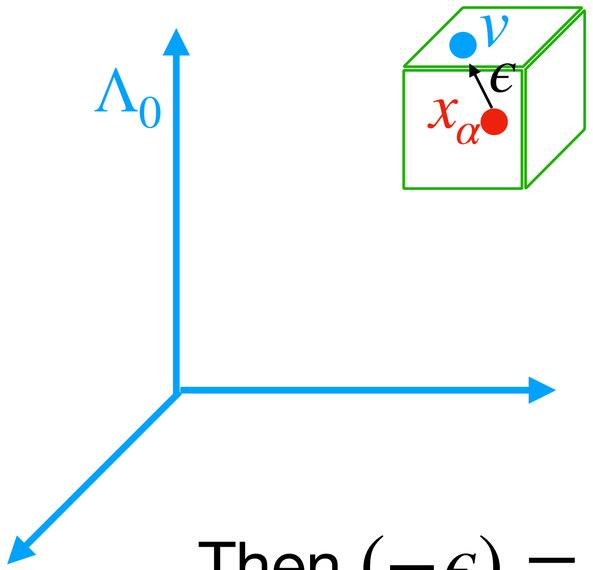
Let $v \in (x_\alpha + [-M, M]^k) \cap \Lambda_0(\mathbf{y})$

Write $v = x_\alpha + \epsilon,$ with $\epsilon \in [-M, M]^k$

Then $(-\epsilon) = x_\alpha - v \quad \in (x_\alpha + \Lambda_0(y)) \cap [-M, M]^k$

Knapsack

Claim : a “short” solution in $\Lambda_\alpha(\mathbf{y})$ is the same as a vector in $\Lambda_0(\mathbf{y})$ “close” to x_α



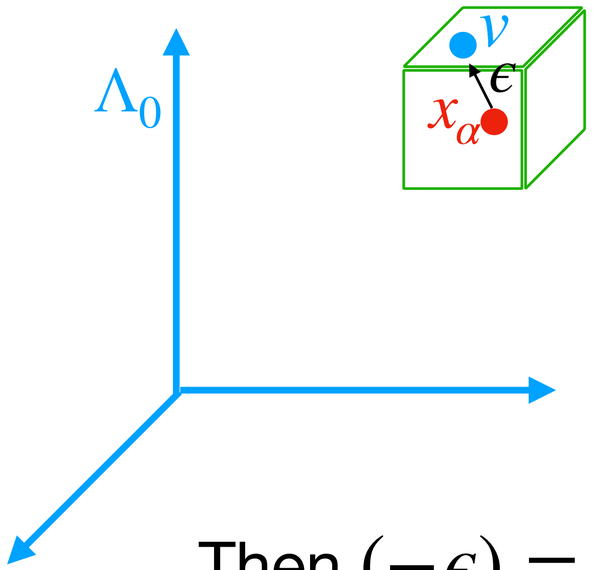
Let $v \in (x_\alpha + [-M, M]^k) \cap \Lambda_0(\mathbf{y})$

Write $v = x_\alpha + \epsilon,$ with $\epsilon \in [-M, M]^k$

Then $(-\epsilon) = x_\alpha - v \quad \in (x_\alpha + \Lambda_0(y)) \cap [-M, M]^k \quad = \Lambda_\alpha(y) \cap [-M, M]^k$

Knapsack

Claim : a “short” solution in $\Lambda_\alpha(\mathbf{y})$ is the same as a vector in $\Lambda_0(\mathbf{y})$ “close” to x_α



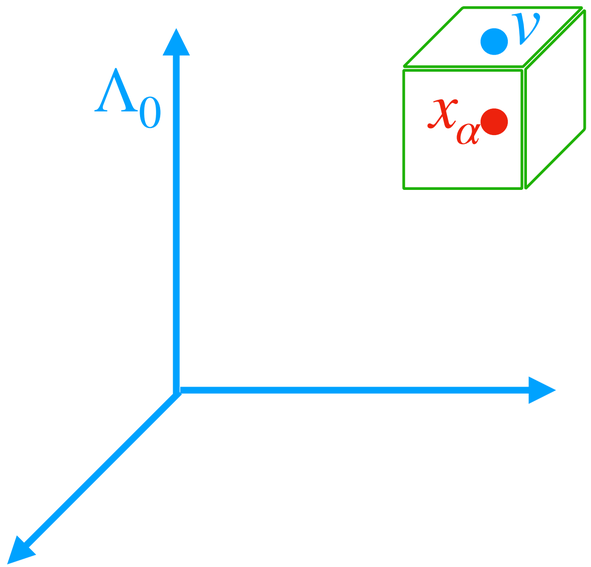
Let $v \in (x_\alpha + [-M, M]^k) \cap \Lambda_0(\mathbf{y})$

Write $v = x_\alpha + \epsilon,$ with $\epsilon \in [-M, M]^k$

Then $(-\epsilon) = x_\alpha - v \quad \in (x_\alpha + \Lambda_0(y)) \cap [-M, M]^k \quad = \Lambda_\alpha(y) \cap [-M, M]^k$

closest vector problem!

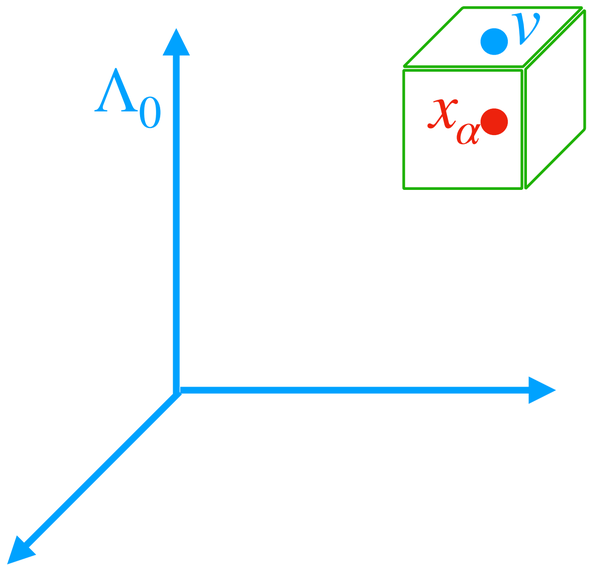
Enumeration



With A a basis of Λ_0, target vector x_α,

we want $v \in \Lambda_0$ such that $\|v - x_\alpha\|_\infty \leq M$

Enumeration

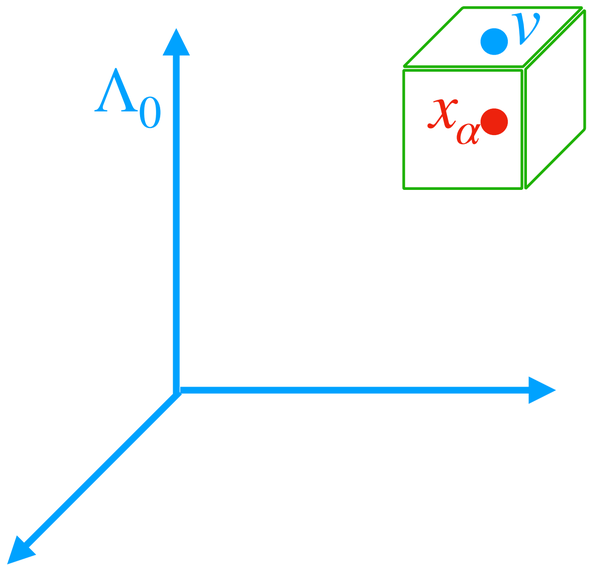


With A a basis of Λ_0, target vector x_α,

we want $v \in \Lambda_0$ such that $\|v - x_\alpha\|_\infty \leq M$

Define $v_0 := \lfloor A^{-1} x_\alpha \rceil$ (not a solution)

Enumeration



With A a basis of Λ_0, target vector x_α,

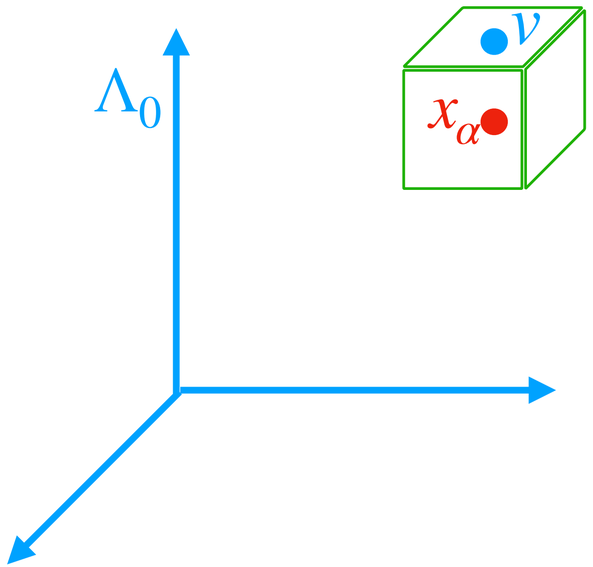
we want $v \in \Lambda_0$ such that $\|v - x_\alpha\|_\infty \leq M$

Define $v_0 := \lfloor A^{-1} x_\alpha \rceil$ (not a solution)

For each v “close” to v_0 :

Check if $Av - x_\alpha \in [-M, M]^k$

Enumeration



With A a basis of Λ_0, target vector x_α,

we want $v \in \Lambda_0$ such that $\|v - x_\alpha\|_\infty \leq M$

Define $v_0 := \lfloor A^{-1} x_\alpha \rceil$ (not a solution)

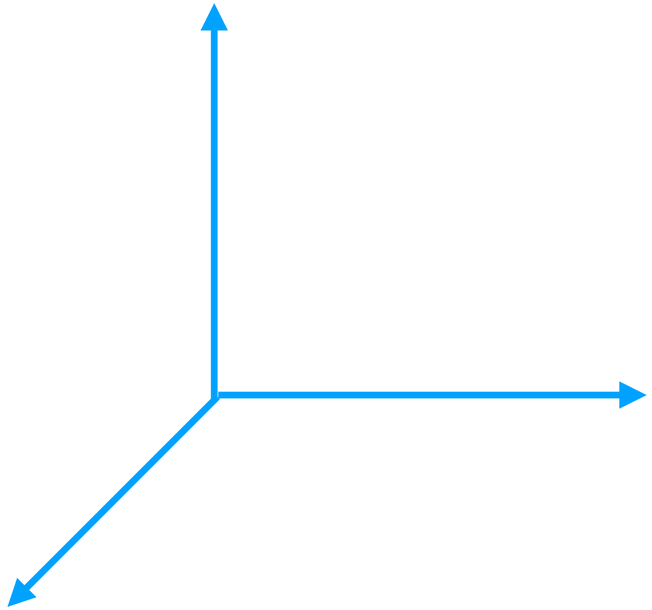
For each v “close” to v_0 :

Check if $Av - x_\alpha \in [-M, M]^k$

~ low memory, high gate count ~

Sieving

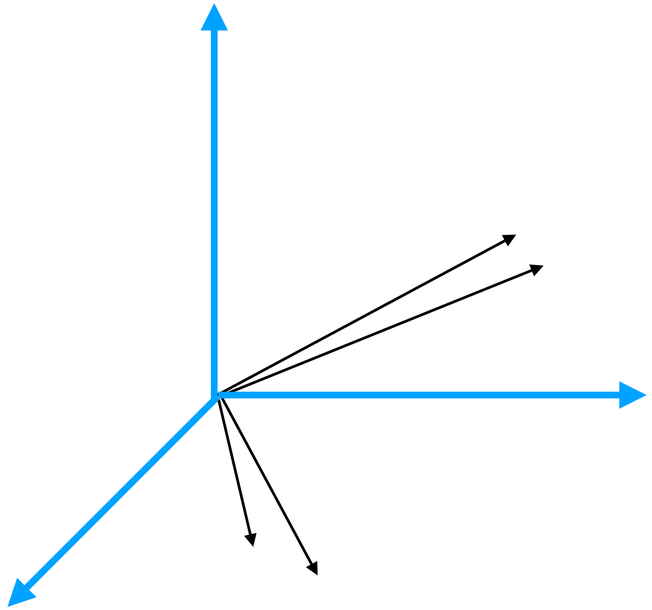
0. Reduce to an SVP in dimension $k+1$



Sieving

0. Reduce to an SVP in dimension $k + 1$

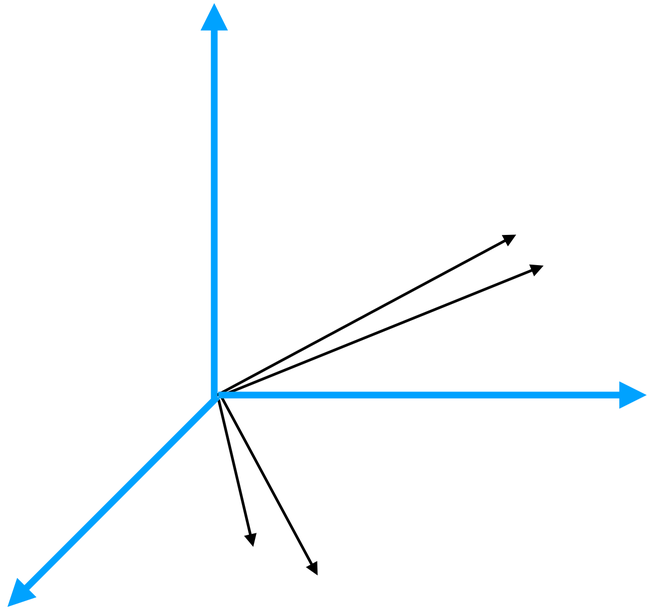
1. List *many* vectors



Sieving

0. Reduce to an SVP in dimension $k + 1$

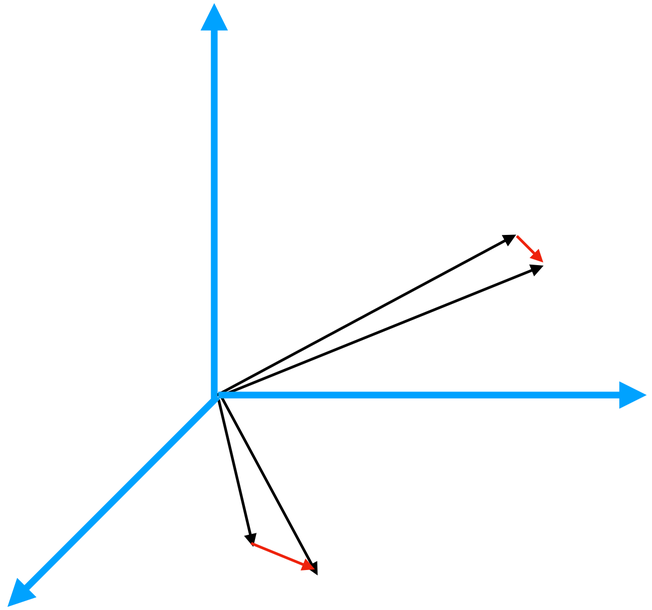
1. List *many* vectors
2. Subtract “close” vectors to obtain shorter vectors



Sieving

0. Reduce to an SVP in dimension $k + 1$

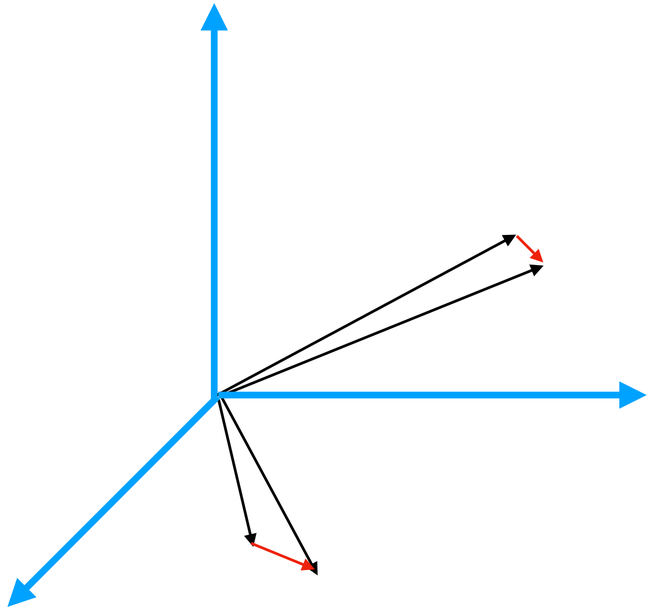
1. List *many* vectors
2. Subtract “close” vectors to obtain shorter vectors



Sieving

0. Reduce to an SVP in dimension $k + 1$

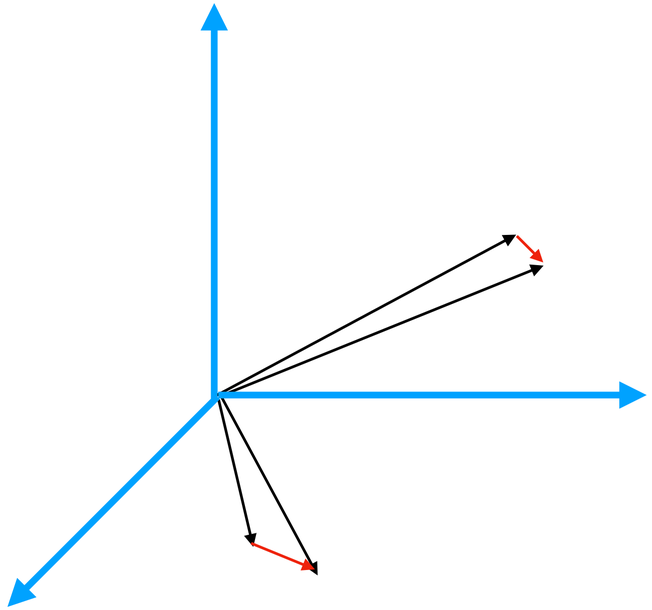
1. List *many* vectors
2. Subtract “close” vectors to obtain shorter vectors
3. Repeat



Sieving

0. Reduce to an SVP in dimension $k+1$

1. List *many* vectors
2. Subtract “close” vectors to obtain shorter vectors
3. Repeat

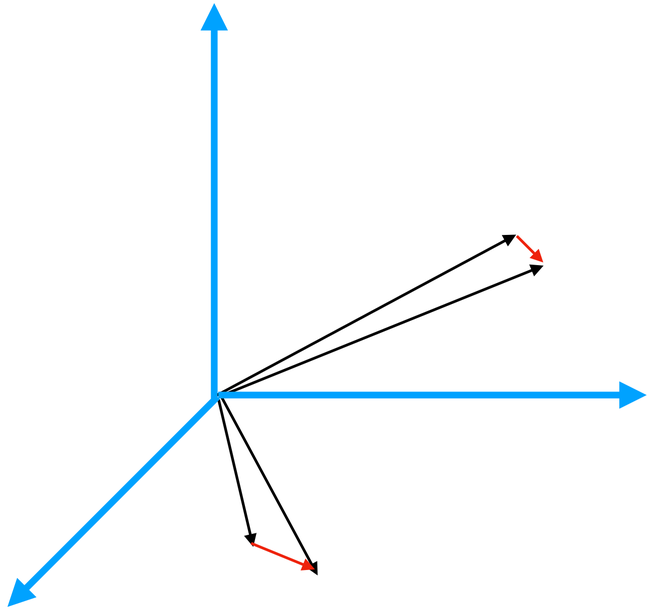


Long lists $\Longrightarrow$ large memory

Sieving

0. Reduce to an SVP in dimension $k+1$

1. List *many* vectors
2. Subtract “close” vectors to obtain shorter vectors
3. Repeat



Long lists $\Longrightarrow$ large memory

No QRAM necessary!! Only qubits

Application to CSI-SharK

Childs-van Dam

Group action cost estimate	Peikert [3]	M	CvD+sieving	CvD+enum
T-gates	T-gates		T-gates	T-gates
BLMP [1] $2^{43.8}$	$2^{59.8}$	2^{8}	$2^{51.7}$	$2^{73.9}$
		2^{12}	$2^{50.8}$	$2^{56.0}$
		2^{16}	$2^{50.5}$	$2^{52.7}$
BS [2] $2^{52.4}$	$2^{68.4}$	2^{8}	2^{60}	$2^{73.9}$
		2^{12}	$2^{59.4}$	$2^{56.7}$
		2^{16}	$2^{59.1}$	$2^{56.4}$

[1] Bernstein, Lange, Martindale, Panny. *Quantum circuits for the CSIDH: optimizing quantum evaluation of isogenies*. Eurocrypt 2019.

[2] Bonnetain, Schrottenloher. *Quantum security analysis of CSIDH.* Eurocrypt 2020.

[3] Peikert. *He gives c-sieves on the CSIDH.* Eurocrypt 2020.

In summary

Conclusion

(Childs)

— CSI-SharK is less secure than expected

— New cryptanalysis tool for multiple hidden shift problem :

Childs-van Dam (room for improvement)

ePrint 2025/376