Sieve, Enumerate, Slice, and Lift:
Hybrid Lattice Algorithms for SVP via CVPP

Emmanouil Doulgerakis, Thijs Laarhoven, and Benne de Weger

Technische Universiteit Eindhoven

July 2020

AfricaCrypt 2020, Cairo, Egypt

Outline

Outline

What is a lattice?

Definition

A lattice $\mathcal{L}$ is a discrete additive subgroup of $\mathbb{R}^n$.

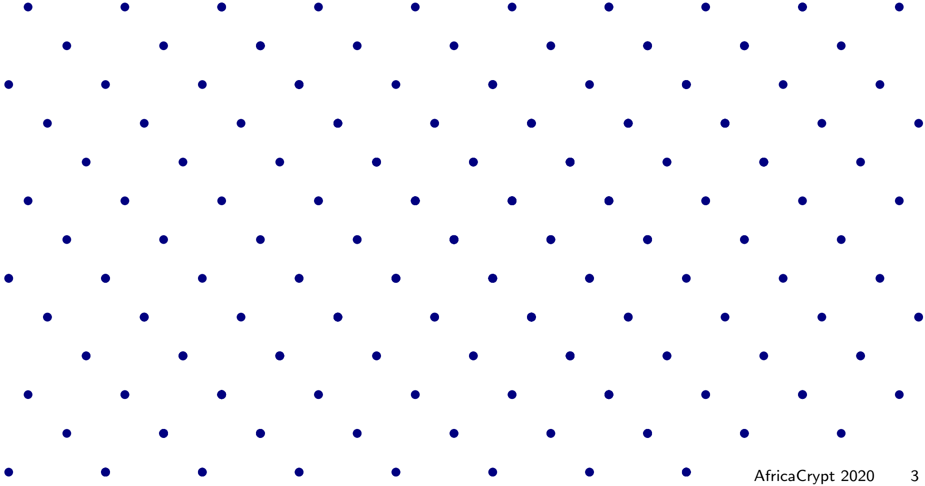
What is a lattice?

Definition

A lattice $\mathcal{L}$ is a discrete additive subgroup of $\mathbb{R}^n$.

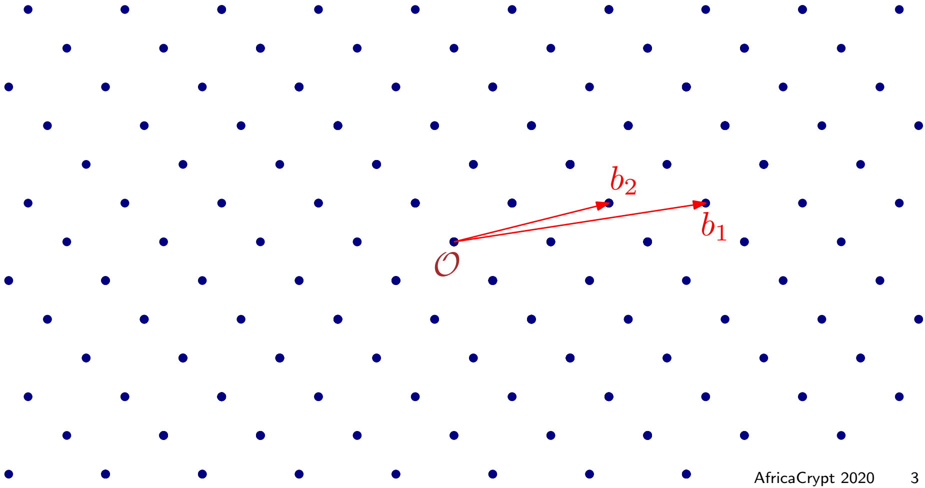
What is a lattice?

A lattice is an infinite grid of points in the n-dimensional space.



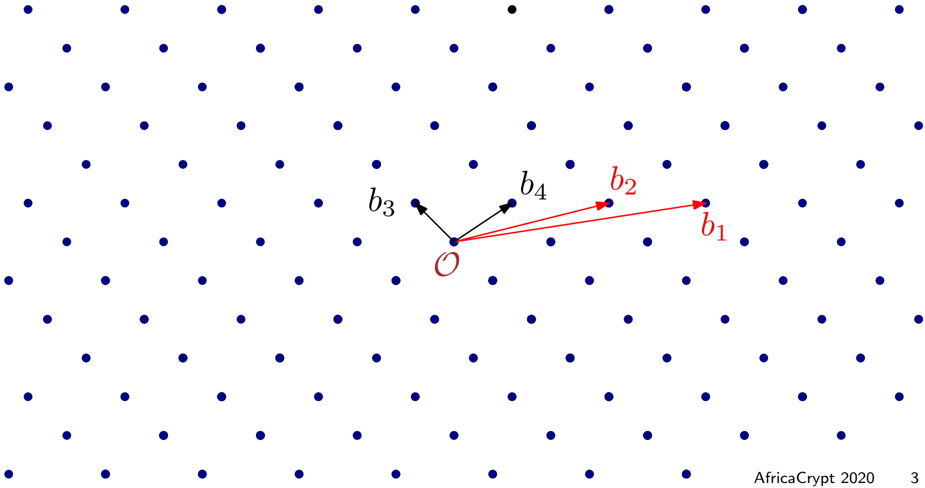
What is a lattice?

A lattice: The set of all integer linear combinations of some basis $\mathbf{B}$ where $\mathbf{B} = \{b_1, \ldots, b_n\} \subset \mathbb{R}^n$.



What is a lattice?

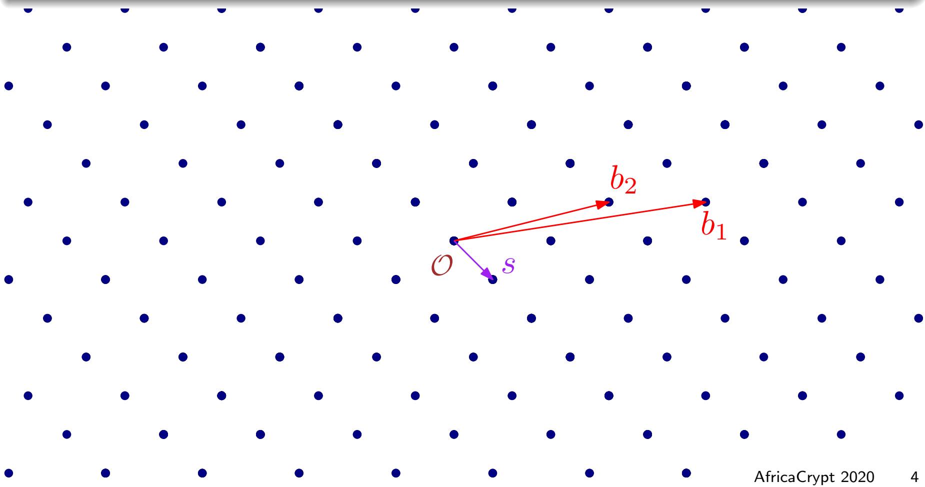
A lattice: The set of all integer linear combinations of some basis $\mathbf{B}$ where $\mathbf{B} = \{b_1, \ldots, b_n\} \subset \mathbb{R}^n$.
A lattice has many bases.



The Shortest Vector Problem (SVP)

Shortest Vector Problem (SVP)

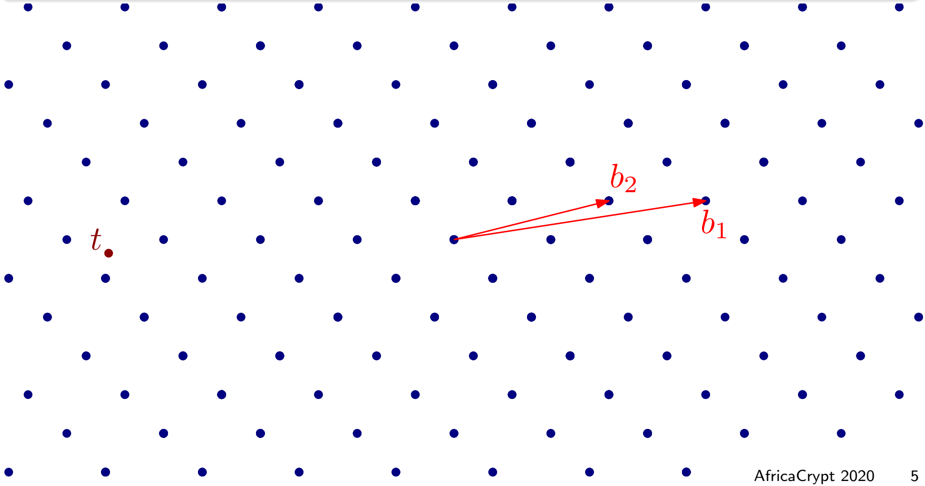
Given an arbitrary basis for $\mathcal{L}$, find a shortest non-zero vector s in $\mathcal{L}$ i.e. $\|s\| = \min_{v \in \mathcal{L} \setminus \{0\}} \|v\|$. We denote $\lambda_1(\mathcal{L}) = \min_{v \in \mathcal{L} \setminus \{0\}} \|v\|$.



The Closest Vector Problem (CVP)

Closest Vector Problem (CVP)

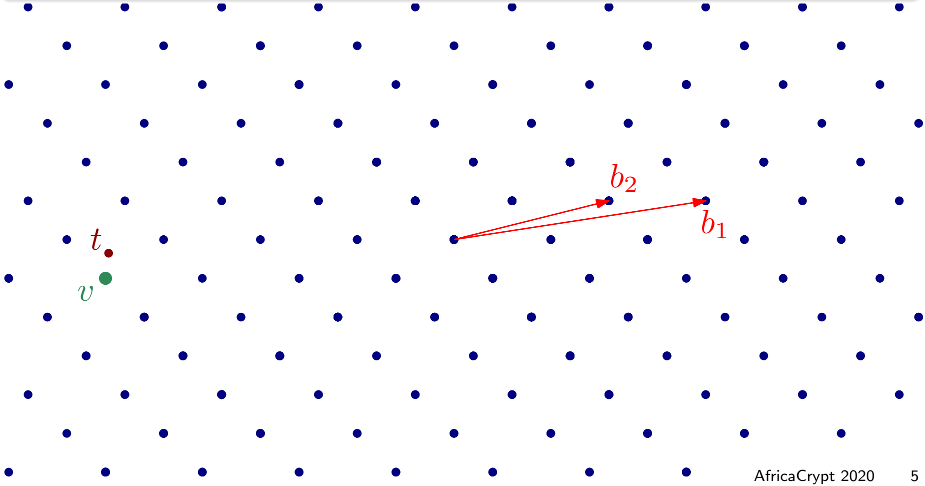
Given an arbitrary basis for $\mathcal{L}$ and a target vector t, find the closest lattice vector v in $\mathcal{L}$ such that $\|t - v\| = d(t, \mathcal{L})$.



The Closest Vector Problem (CVP)

Closest Vector Problem (CVP)

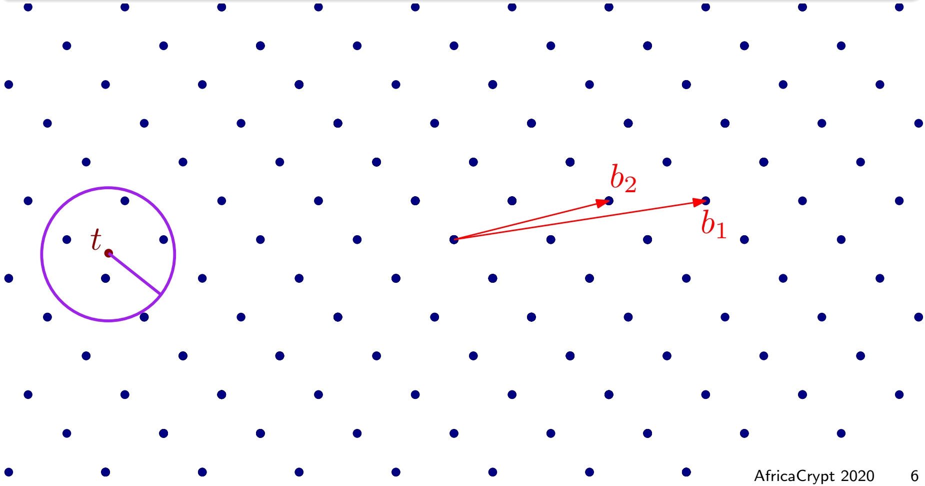
Given an arbitrary basis for $\mathcal{L}$ and a target vector t, find the closest lattice vector v in $\mathcal{L}$ such that $\|t - v\| = d(t, \mathcal{L})$.



The Approximate Closest Vector Problem (CVP_κ)

Approximate Closest Vector Problem (CVP_κ)

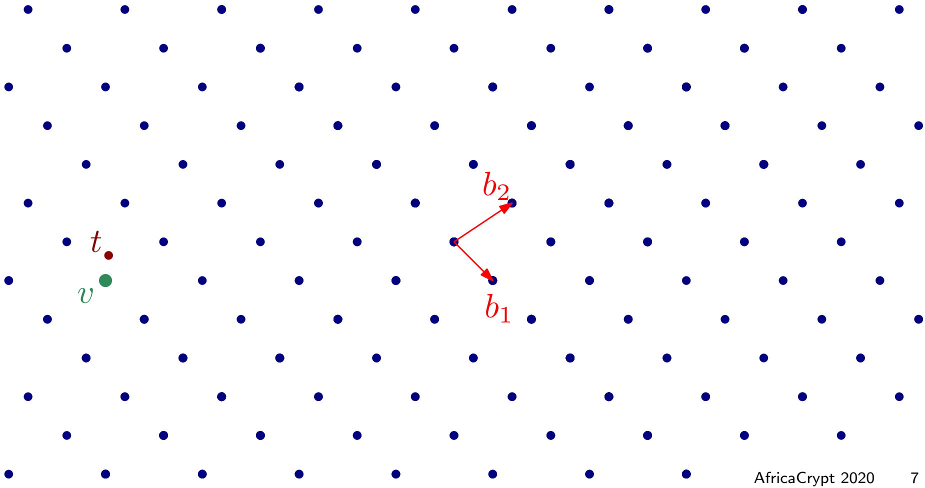
Given an arbitrary basis for $\mathcal{L}$, a target vector t and an approximation factor $\kappa \geq 1$, find a lattice vector v in $\mathcal{L}$ such that $\|t - v\| \leq \kappa d(t, \mathcal{L})$.



The Closest Vector Problem with Pre-processing (CVPP)

The CVPP variant

Given an arbitrary basis for $\mathcal{L}$, compute some pre-processing data such that when later given a target vector t, it will be "easy" to solve the CVP for t.



Outline

Solving SVP

- Let $\mathcal{L}$ be a lattice with basis $\mathbf{B} = \{\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n\} \subset \mathbb{R}^n$.
  Question: Find $\boldsymbol{s}$ in $\mathcal{L}$ with $\|\boldsymbol{s}\| = \lambda_1(\mathcal{L})$.

Solving SVP

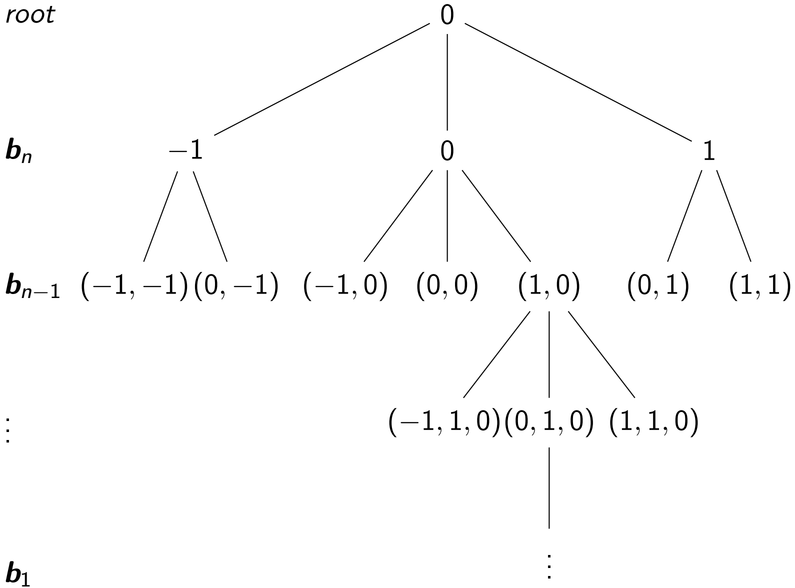
- Let $\mathcal{L}$ be a lattice with basis $\mathbf{B} = \{\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n\} \subset \mathbb{R}^n$.
  Question: Find $\boldsymbol{s}$ in $\mathcal{L}$ with $\|\boldsymbol{s}\| = \lambda_1(\mathcal{L})$.

- As $\boldsymbol{s} \in \mathcal{L}$ then $\exists x_1, \ldots, x_n \in \mathbb{Z}$ such that $\boldsymbol{s} = x_1 \boldsymbol{b}_1 + \cdots + x_n \boldsymbol{b}_n$.

Solving SVP

- Let $\mathcal{L}$ be a lattice with basis $\mathbf{B} = \{\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n\} \subset \mathbb{R}^n$.
  Question: Find $\boldsymbol{s}$ in $\mathcal{L}$ with $\|\boldsymbol{s}\| = \lambda_1(\mathcal{L})$.
- As $\boldsymbol{s} \in \mathcal{L}$ then $\exists x_1, \ldots, x_n \in \mathbb{Z}$ such that $\boldsymbol{s} = x_1\boldsymbol{b}_1 + \cdots + x_n\boldsymbol{b}_n$.
- We know that $\lambda_1(\mathcal{L}) \leq \|\boldsymbol{b}_1\|$.
- Enumeration explores all the choices of the x_i such that $\|x_1\boldsymbol{b}_1 + \cdots + x_n\boldsymbol{b}_n\| \leq \|\boldsymbol{b}_1\|$.

Enumeration tree (example)



Enumeration costs in small depth

Lemma (Costs of enumeration HS07)

Let $\mathbf{B}$ *be a strongly reduced basis of a lattice. Then the number of nodes* E_k *at depth* $k = o(n)$, $k = n^{1-o(1)}$, *satisfies:*

$$\mathrm{E}_k = n^{k/2+o(k)}.$$

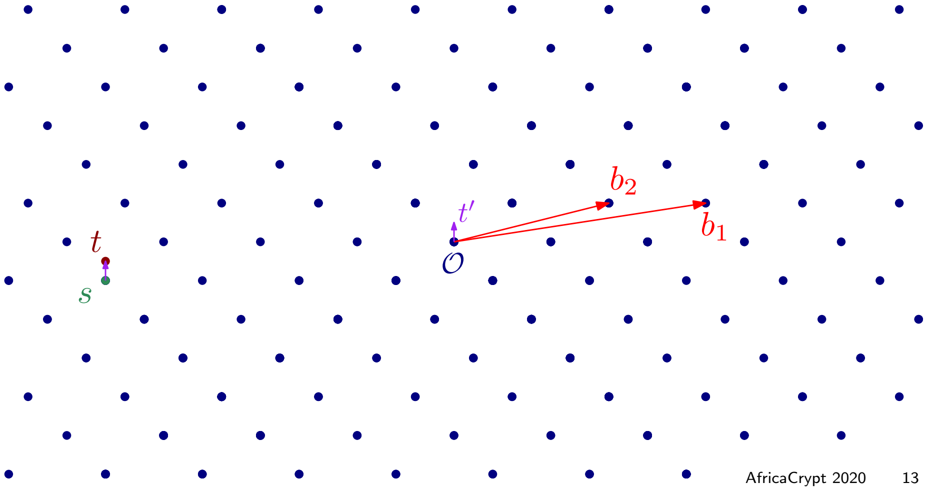
Enumerating all these nodes can be done in time $\mathrm{T}_{\mathrm{enum}}$ *and space* $\mathrm{S}_{\mathrm{enum}}$, *with:*

$$\mathrm{T}_{\mathrm{enum}} = \mathrm{E}_k \cdot n^{O(1)}, \qquad \mathrm{S}_{\mathrm{enum}} = n^{O(1)}.$$

Outline

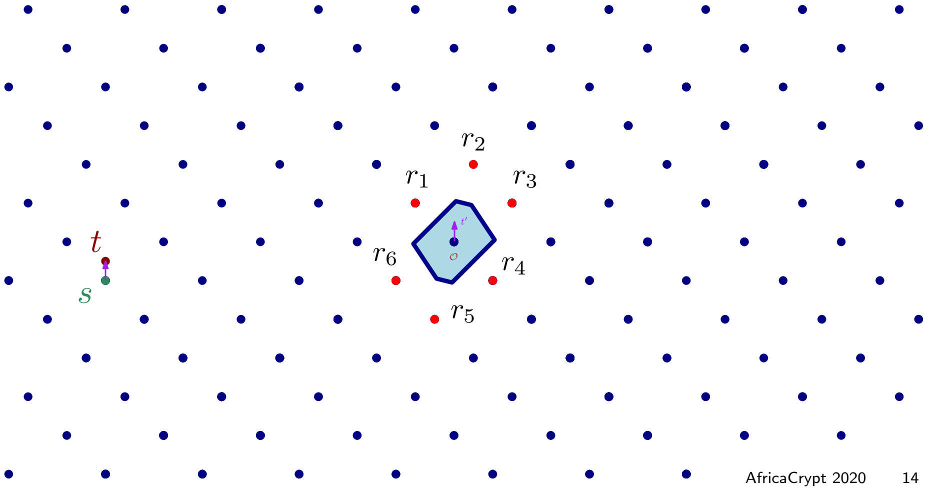
Solving CVP(P)

We have $t \in t + \mathcal{L}$ and $t' = t - s$ so $t' \in t + \mathcal{L}$ as well...
It suffices to find t'.



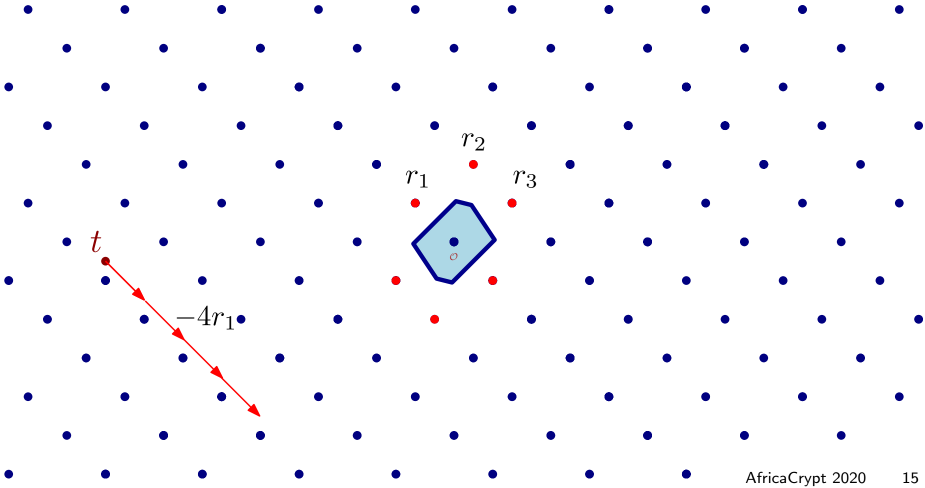
The iterative slicer (ideal case)

Create a list $L \subseteq \mathcal{L}$. Keep reducing t by the vectors r in the list L until the result cannot be reduced any more. Then we have found t'.



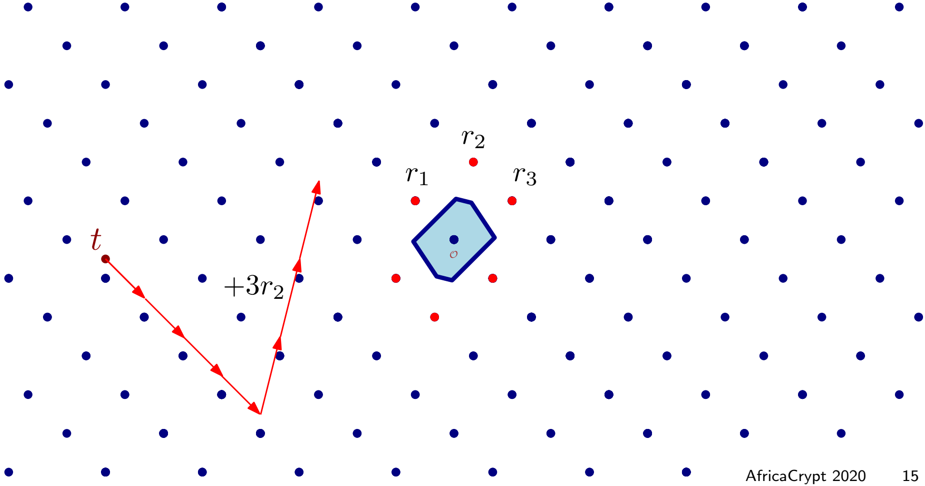
The iterative slicer (ideal case)

Create a list $L \subseteq \mathcal{L}$. Keep reducing t by the vectors r in the list L until the result cannot be reduced any more. Then we have found t'.



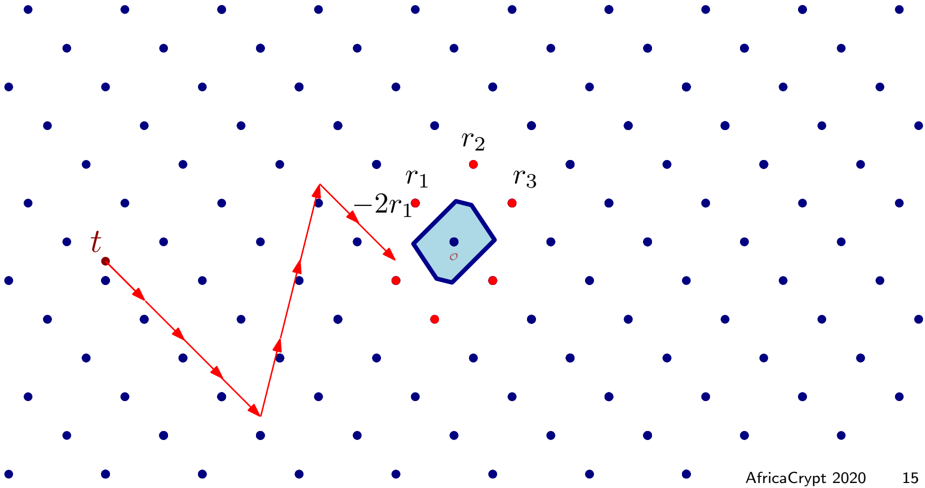
The iterative slicer (ideal case)

Create a list $L \subseteq \mathcal{L}$. Keep reducing t by the vectors r in the list L until the result cannot be reduced any more. Then we have found t'.



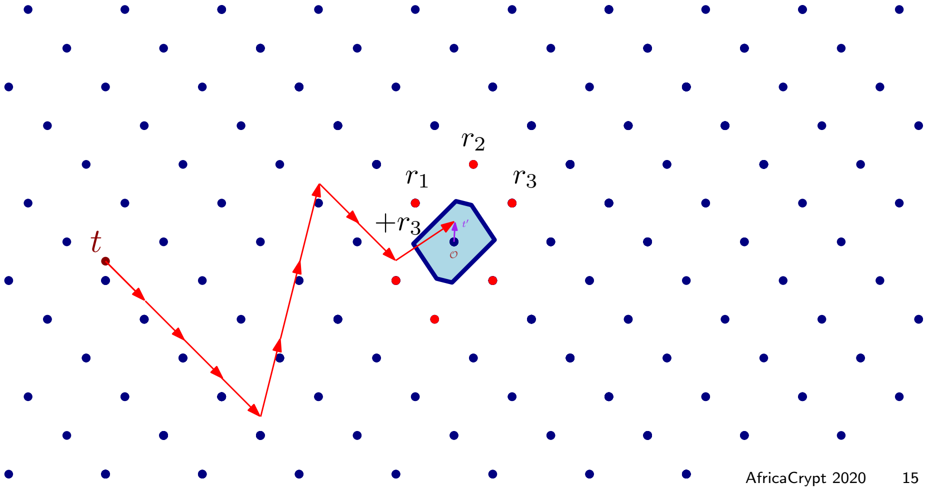
The iterative slicer (ideal case)

Create a list $L \subseteq \mathcal{L}$. Keep reducing t by the vectors r in the list L until the result cannot be reduced any more. Then we have found t'.



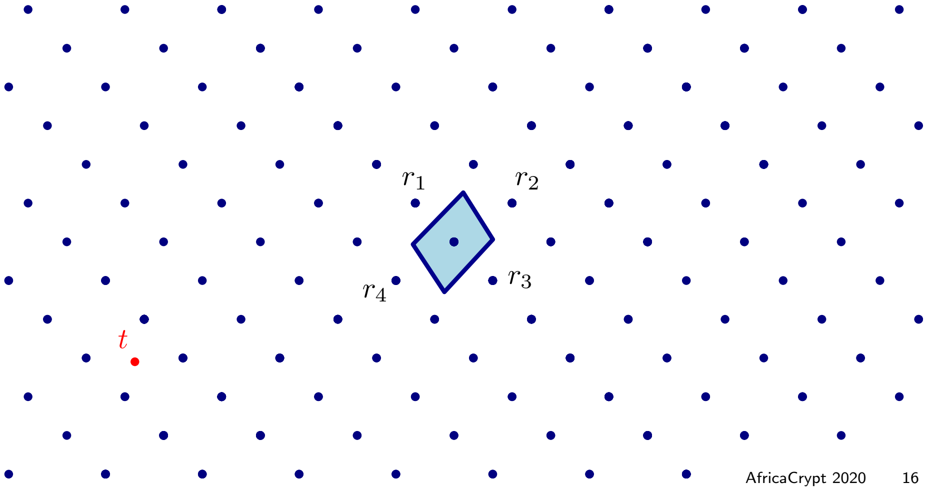
The iterative slicer (ideal case)

Create a list $L \subseteq \mathcal{L}$. Keep reducing t by the vectors r in the list L until the result cannot be reduced any more. Then we have found t'.



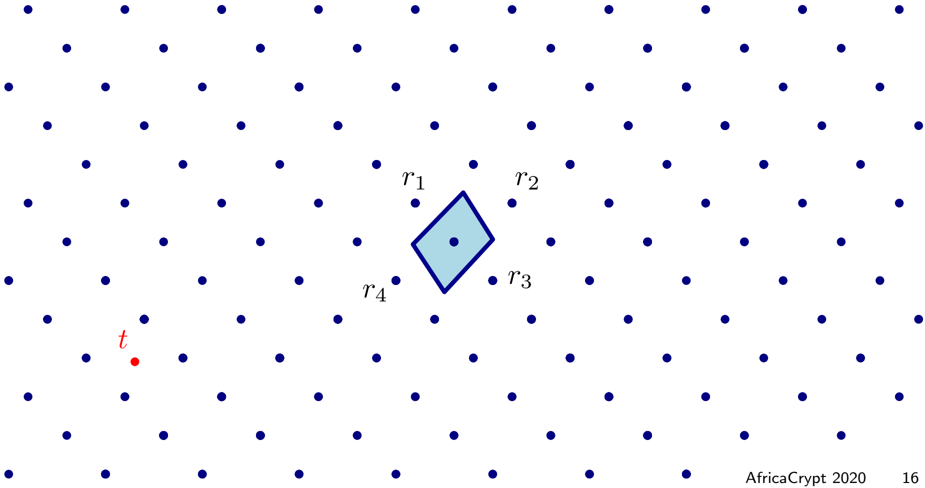
The iterative slicer (in practice)

- Computing t' correctly depends on the list L. Computing "the proper" list L is too costly. We can use approximations instead.



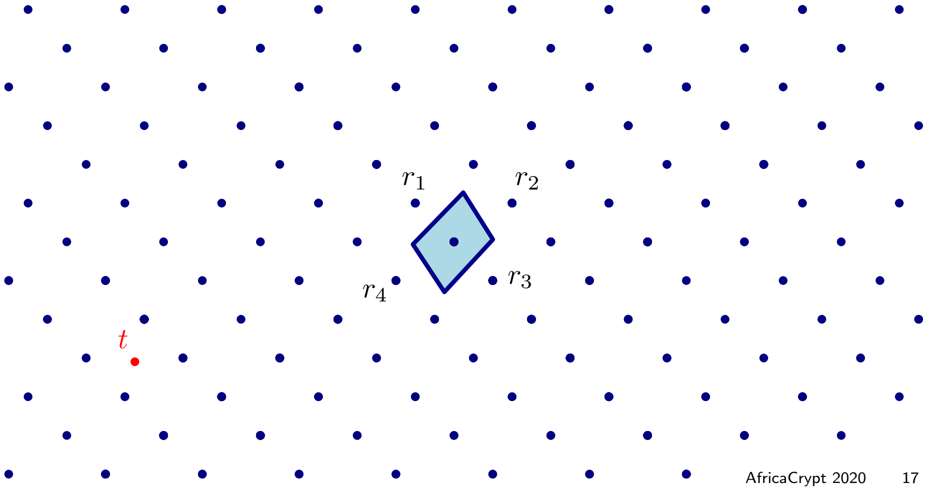
The iterative slicer (in practice)

- Computing t' correctly depends on the list L. Computing "the proper" list L is too costly. We can use approximations instead.
- Disadvantage: We might get a wrong t'.



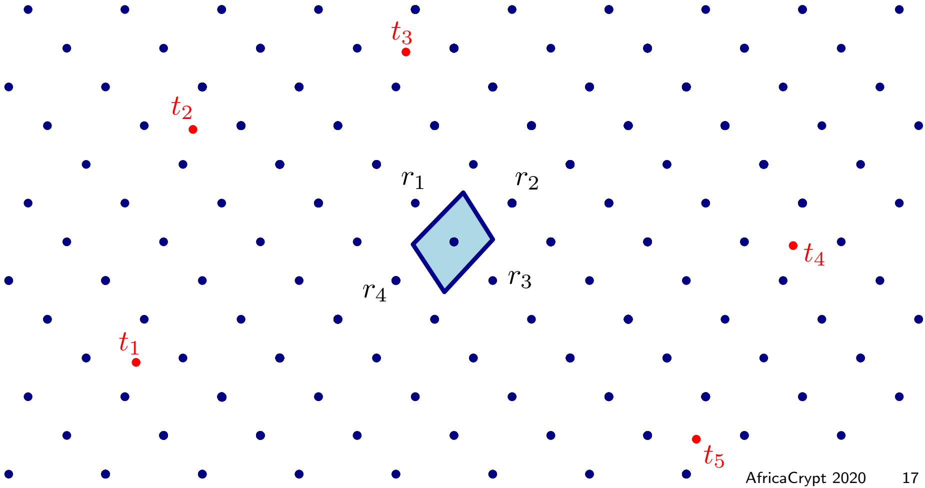
The randomized slicer

- Create a list L of lattice vectors (e.g. by running a sieving algorithm).



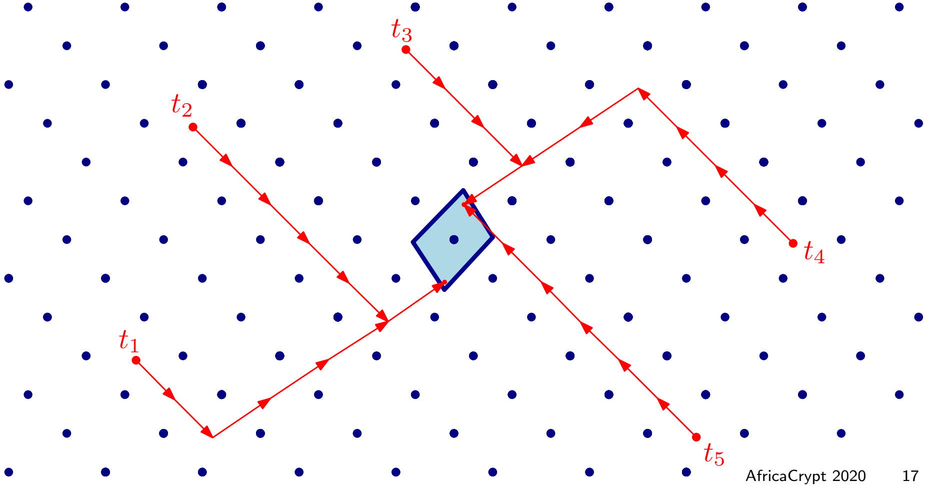
The randomized slicer

- Create a list L of lattice vectors (e.g. by running a sieving algorithm).
- Randomize t sufficiently many times (as t_i) and reduce it.



The randomized slicer

- Create a list L of lattice vectors (e.g. by running a sieving algorithm).
- Randomize t sufficiently many times (as t_i) and reduce it.
- Keep the shortest t_i' found as t'.



The randomized slicer algorithm

Algorithm 2 The randomized heuristic slicer for finding closest vectors

Require: A list $L \subset \mathcal{L}$ and a target $\boldsymbol{t} \in \mathbb{R}^d$
Ensure: The algorithm outputs a closest lattice vector $\boldsymbol{s} \in \mathcal{L}$ to $\boldsymbol{t}$

$\boldsymbol{s} \leftarrow \boldsymbol{0}$ ▷ Initial guess $\boldsymbol{s}$ for closest vector to $\boldsymbol{t}$
repeat
    Sample $\boldsymbol{t}' \sim D_{\boldsymbol{t}+\mathcal{L},s}$ ▷ Randomly shift $\boldsymbol{t}$ by a vector $\boldsymbol{v} \in \mathcal{L}$
    for each $\boldsymbol{r} \in L$ **do**
        if $\|\boldsymbol{t}' - \boldsymbol{r}\| < \|\boldsymbol{t}'\|$ **then** ▷ New shorter vector $\boldsymbol{t}' \in \boldsymbol{t} + \mathcal{L}$
            Replace $\boldsymbol{t}' \leftarrow \boldsymbol{t}' - \boldsymbol{r}$ and restart the **for**-loop
    if $\|\boldsymbol{t}'\| < \|\boldsymbol{t} - \boldsymbol{s}\|$ **then**
        $\boldsymbol{s} \leftarrow \boldsymbol{t} - \boldsymbol{t}'$ ▷ New lattice vector $\boldsymbol{s}$ closer to $\boldsymbol{t}$
until $\boldsymbol{s}$ is a closest lattice vector to $\boldsymbol{t}$
return $\boldsymbol{s}$

Costs of preprocessing

Lemma (Costs of lattice sieving BDGL16)

Given a basis $\mathbf{B}$ *of a lattice* $\mathcal{L}$*, the LDSieve heuristically returns a list* $L \subset \mathcal{L}$ *containing the* $(4/3)^{n/2+o(n)}$ *shortest lattice vectors, in time* $\mathrm{T}_{\mathrm{sieve}}$ *and space* $\mathrm{S}_{\mathrm{sieve}}$ *with:*

$$\mathrm{T}_{\mathrm{sieve}} = (3/2)^{n/2+o(n)}, \qquad \mathrm{S}_{\mathrm{sieve}} = (4/3)^{n/2+o(n)}.$$

With the LDSieve we can therefore solve SVP with the above complexities.

Costs of the randomized slicer

Lemma (single target DLW20)

Given a list of the $(4/3)^{n/2+o(n)}$ *shortest vectors of a lattice* $\mathcal{L}$ *and a target* $\boldsymbol{t} \in \mathbb{R}^n$*, the randomized slicer solves CVP for* $\boldsymbol{t}$ *in time* $\mathrm{T}_{\mathrm{slice}}$ *and space* $\mathrm{S}_{\mathrm{slice}}$*, with:*

$$\mathrm{T}_{\mathrm{slice}} = 2^{\zeta n + o(n)}, \qquad \mathrm{S}_{\mathrm{slice}} = (4/3)^{n/2+o(n)}.$$

In our case $\zeta = 0.2639\ldots$

Costs of the randomized slicer

Lemma (many targets DLW20)

Given a list of the $(4/3)^{n/2+o(n)}$ shortest vectors of a lattice $\mathcal{L}$ and a batch of $N \geq (13/12)^{n/2+o(n)}$ target vectors $\boldsymbol{t}_1, \ldots, \boldsymbol{t}_N \in \mathbb{R}^n$, the batched randomized slicer solves CVP for all targets $\boldsymbol{t}_i$ in total time $\mathrm{T}_{\mathrm{slice}}$ and space $\mathrm{S}_{\mathrm{slice}}$, with:

$$\mathrm{T}_{\mathrm{slice}} = N \cdot (18/13)^{n/2+o(n)}, \qquad \mathrm{S}_{\mathrm{slice}} = (4/3)^{n/2+o(n)}.$$

Outline

Solving SVP via CVPP (Part 1)

- Let $\mathcal{L}$ be a lattice with basis $\mathbf{B} = \{\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n\} \subset \mathbb{R}^n$.
  Question: Find $\boldsymbol{s}$ in $\mathcal{L}$ with $\|\boldsymbol{s}\| = \lambda_1(\mathcal{L})$.

Solving SVP via CVPP (Part 1)

- Let $\mathcal{L}$ be a lattice with basis $\mathbf{B} = \{\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n\} \subset \mathbb{R}^n$.
  Question: Find $\boldsymbol{s}$ in $\mathcal{L}$ with $\|\boldsymbol{s}\| = \lambda_1(\mathcal{L})$.
- Choose $0 \leq k \leq n$ and split $\mathbf{B}$ as $\mathbf{B} = \mathbf{B}_{\mathrm{bot}} \cup \mathbf{B}_{\mathrm{top}}$ where $\mathbf{B}_{\mathrm{bot}} := \{\boldsymbol{b}_1, \ldots, \boldsymbol{b}_{n-k}\}$ and $\mathbf{B}_{\mathrm{top}} := \{\boldsymbol{b}_{n-k+1}, \ldots, \boldsymbol{b}_n\}$.

Solving SVP via CVPP (Part 1)

- Let $\mathcal{L}$ be a lattice with basis $\mathbf{B} = \{\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n\} \subset \mathbb{R}^n$.
  Question: Find $\boldsymbol{s}$ in $\mathcal{L}$ with $\|\boldsymbol{s}\| = \lambda_1(\mathcal{L})$.
- Choose $0 \leq k \leq n$ and split $\mathbf{B}$ as $\mathbf{B} = \mathbf{B}_{\mathrm{bot}} \cup \mathbf{B}_{\mathrm{top}}$ where $\mathbf{B}_{\mathrm{bot}} := \{\boldsymbol{b}_1, \ldots, \boldsymbol{b}_{n-k}\}$ and $\mathbf{B}_{\mathrm{top}} := \{\boldsymbol{b}_{n-k+1}, \ldots, \boldsymbol{b}_n\}$.
- This partitions the lattice as $\mathcal{L} = \mathcal{L}_{\mathrm{bot}} \oplus \mathcal{L}_{\mathrm{top}}$ where $\mathcal{L}_{\mathrm{bot}} := \mathcal{L}(\mathbf{B}_{\mathrm{bot}})$ and $\mathcal{L}_{\mathrm{top}} := \mathcal{L}(\mathbf{B}_{\mathrm{top}})$.

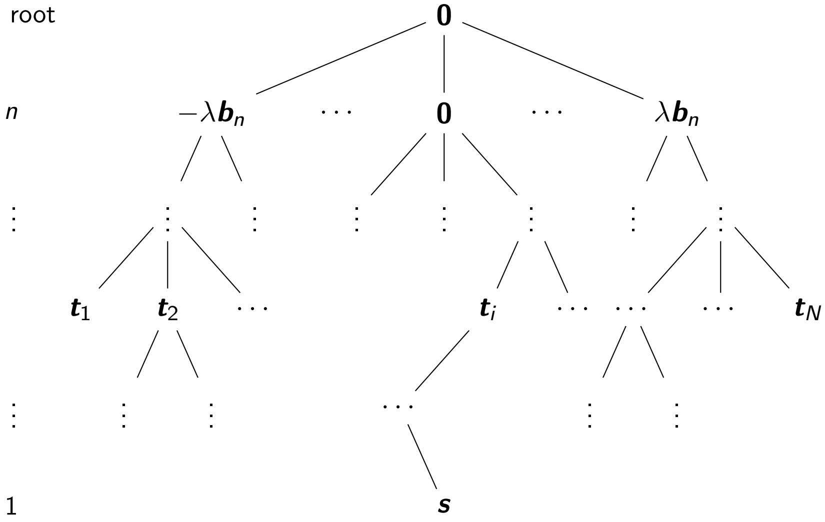
Solving SVP via CVPP (Part 1)

- Let $\mathcal{L}$ be a lattice with basis $\mathbf{B} = \{\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n\} \subset \mathbb{R}^n$.
  Question: Find $\boldsymbol{s}$ in $\mathcal{L}$ with $\|\boldsymbol{s}\| = \lambda_1(\mathcal{L})$.
- Choose $0 \leq k \leq n$ and split $\mathbf{B}$ as $\mathbf{B} = \mathbf{B}_{\mathrm{bot}} \cup \mathbf{B}_{\mathrm{top}}$ where $\mathbf{B}_{\mathrm{bot}} := \{\boldsymbol{b}_1, \ldots, \boldsymbol{b}_{n-k}\}$ and $\mathbf{B}_{\mathrm{top}} := \{\boldsymbol{b}_{n-k+1}, \ldots, \boldsymbol{b}_n\}$.
- This partitions the lattice as $\mathcal{L} = \mathcal{L}_{\mathrm{bot}} \oplus \mathcal{L}_{\mathrm{top}}$ where $\mathcal{L}_{\mathrm{bot}} := \mathcal{L}(\mathbf{B}_{\mathrm{bot}})$ and $\mathcal{L}_{\mathrm{top}} := \mathcal{L}(\mathbf{B}_{\mathrm{top}})$.
- As $\boldsymbol{s} \in \mathcal{L}$ then $\exists x_1, \ldots, x_n \in \mathbb{Z}$ such that $\boldsymbol{s} = x_1 \boldsymbol{b}_1 + \cdots + x_n \boldsymbol{b}_n$.

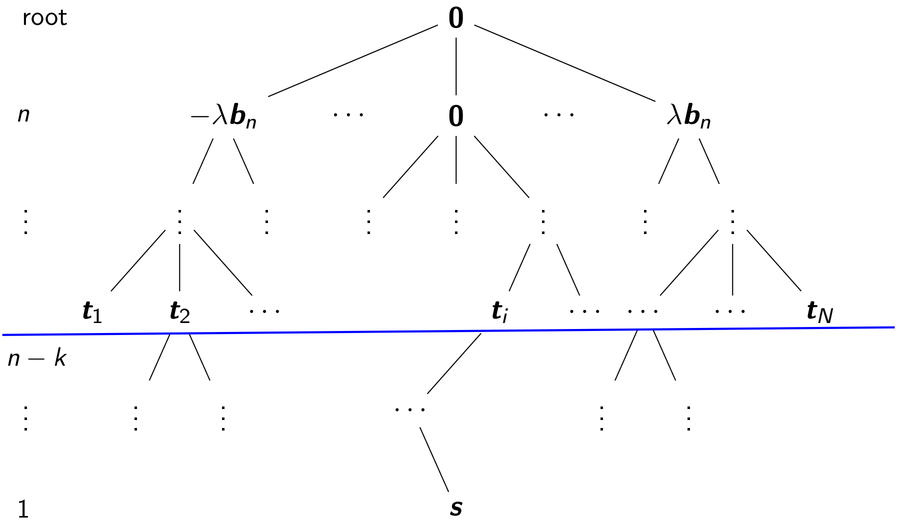
Solving SVP via CVPP (Part 1)

- Let $\mathcal{L}$ be a lattice with basis $\mathbf{B} = \{\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n\} \subset \mathbb{R}^n$.
  Question: Find $\boldsymbol{s}$ in $\mathcal{L}$ with $\|\boldsymbol{s}\| = \lambda_1(\mathcal{L})$.
- Choose $0 \leq k \leq n$ and split $\mathbf{B}$ as $\mathbf{B} = \mathbf{B}_{\mathrm{bot}} \cup \mathbf{B}_{\mathrm{top}}$ where $\mathbf{B}_{\mathrm{bot}} := \{\boldsymbol{b}_1, \ldots, \boldsymbol{b}_{n-k}\}$ and $\mathbf{B}_{\mathrm{top}} := \{\boldsymbol{b}_{n-k+1}, \ldots, \boldsymbol{b}_n\}$.
- This partitions the lattice as $\mathcal{L} = \mathcal{L}_{\mathrm{bot}} \oplus \mathcal{L}_{\mathrm{top}}$ where $\mathcal{L}_{\mathrm{bot}} := \mathcal{L}(\mathbf{B}_{\mathrm{bot}})$ and $\mathcal{L}_{\mathrm{top}} := \mathcal{L}(\mathbf{B}_{\mathrm{top}})$.
- As $\boldsymbol{s} \in \mathcal{L}$ then $\exists x_1, \ldots, x_n \in \mathbb{Z}$ such that $\boldsymbol{s} = x_1 \boldsymbol{b}_1 + \cdots + x_n \boldsymbol{b}_n$.
- We can also split $\boldsymbol{s}$ as $\boldsymbol{s} = \boldsymbol{s}_{\mathrm{bot}} + \boldsymbol{s}_{\mathrm{top}}$ where
  $\boldsymbol{s}_{\mathrm{bot}} = x_1 \boldsymbol{b}_1 + \cdots + x_{n-k} \boldsymbol{b}_{n-k} \in \mathcal{L}_{\mathrm{bot}}$ and
  $\boldsymbol{s}_{\mathrm{top}} = x_{n-k+1} \boldsymbol{b}_{n-k+1} + \cdots + x_n \boldsymbol{b}_n \in \mathcal{L}_{\mathrm{top}}$.

Solving SVP via CVPP (Part 1)



Solving SVP via CVPP (Part 1)



Solving SVP via CVPP (Part 2)

- We split $\boldsymbol{s}$ as $\boldsymbol{s} = \boldsymbol{s}_{\mathrm{bot}} + \boldsymbol{s}_{\mathrm{top}}$ where $\boldsymbol{s}_{\mathrm{bot}} = x_1 \boldsymbol{b}_1 + \cdots + x_{n-k} \boldsymbol{b}_{n-k} \in \mathcal{L}_{\mathrm{bot}}$ and $\boldsymbol{s}_{\mathrm{top}} = x_{n-k+1} \boldsymbol{b}_{n-k+1} + \cdots + x_n \boldsymbol{b}_n \in \mathcal{L}_{\mathrm{top}}$.

Solving SVP via CVPP (Part 2)

- We split $\boldsymbol{s}$ as $\boldsymbol{s} = \boldsymbol{s}_{\mathrm{bot}} + \boldsymbol{s}_{\mathrm{top}}$ where $\boldsymbol{s}_{\mathrm{bot}} = x_1\boldsymbol{b}_1 + \cdots + x_{n-k}\boldsymbol{b}_{n-k} \in \mathcal{L}_{\mathrm{bot}}$ and $\boldsymbol{s}_{\mathrm{top}} = x_{n-k+1}\boldsymbol{b}_{n-k+1} + \cdots + x_n\boldsymbol{b}_n \in \mathcal{L}_{\mathrm{top}}$.
- Two cases:
  - If $\boldsymbol{s}_{\mathrm{top}} = \mathbf{0}$ then $\boldsymbol{s} = \mathrm{SVP}(\mathcal{L}_{\mathrm{bot}})$.
  - If $\boldsymbol{s}_{\mathrm{top}} \neq \mathbf{0}$ then $\boldsymbol{s} = \boldsymbol{s}_{\mathrm{top}} - \mathrm{CVP}(\mathcal{L}_{\mathrm{bot}}, \boldsymbol{s}_{\mathrm{top}})$.

Solving SVP via CVPP (Part 2)

- We split $\boldsymbol{s}$ as $\boldsymbol{s} = \boldsymbol{s}_{\mathrm{bot}} + \boldsymbol{s}_{\mathrm{top}}$ where $\boldsymbol{s}_{\mathrm{bot}} = x_1 \boldsymbol{b}_1 + \cdots + x_{n-k} \boldsymbol{b}_{n-k} \in \mathcal{L}_{\mathrm{bot}}$ and $\boldsymbol{s}_{\mathrm{top}} = x_{n-k+1} \boldsymbol{b}_{n-k+1} + \cdots + x_n \boldsymbol{b}_n \in \mathcal{L}_{\mathrm{top}}$.
- Two cases:
  - If $\boldsymbol{s}_{\mathrm{top}} = \boldsymbol{0}$ then $\boldsymbol{s} = \mathrm{SVP}(\mathcal{L}_{\mathrm{bot}})$.
  - If $\boldsymbol{s}_{\mathrm{top}} \neq \boldsymbol{0}$ then $\boldsymbol{s} = \boldsymbol{s}_{\mathrm{top}} - \mathrm{CVP}(\mathcal{L}_{\mathrm{bot}}, \boldsymbol{s}_{\mathrm{top}})$.
- The vector $\boldsymbol{s}_{\mathrm{top}}$ will be one of the vectors $\boldsymbol{t}_i$ in the enumeration tree. We do not know in advance which one.

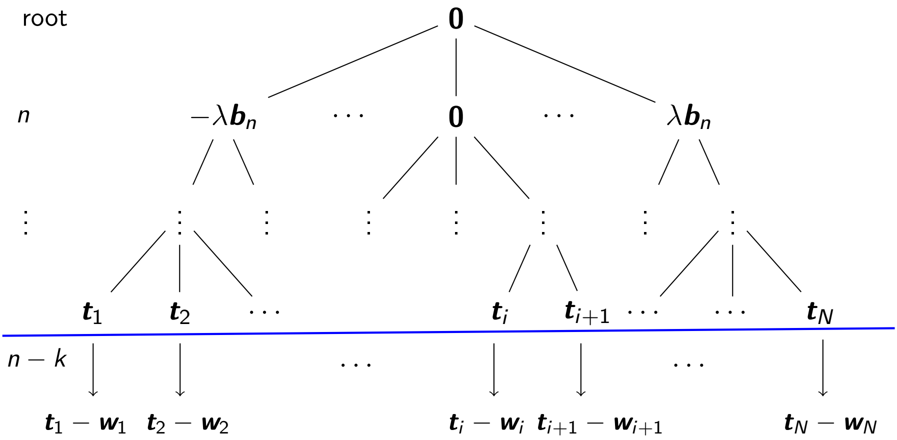
Solving SVP via CVPP (Part 2)

- We split $\boldsymbol{s}$ as $\boldsymbol{s} = \boldsymbol{s}_{\text{bot}} + \boldsymbol{s}_{\text{top}}$ where $\boldsymbol{s}_{\text{bot}} = x_1 \boldsymbol{b}_1 + \cdots + x_{n-k} \boldsymbol{b}_{n-k} \in \mathcal{L}_{\text{bot}}$ and $\boldsymbol{s}_{\text{top}} = x_{n-k+1} \boldsymbol{b}_{n-k+1} + \cdots + x_n \boldsymbol{b}_n \in \mathcal{L}_{\text{top}}$.
- Two cases:
  - If $\boldsymbol{s}_{\text{top}} = \boldsymbol{0}$ then $\boldsymbol{s} = \mathrm{SVP}(\mathcal{L}_{\text{bot}})$.
  - If $\boldsymbol{s}_{\text{top}} \neq \boldsymbol{0}$ then $\boldsymbol{s} = \boldsymbol{s}_{\text{top}} - \mathrm{CVP}(\mathcal{L}_{\text{bot}}, \boldsymbol{s}_{\text{top}})$.
- The vector $\boldsymbol{s}_{\text{top}}$ will be one of the vectors $\boldsymbol{t}_i$ in the enumeration tree. We do not know in advance which one.
- Solve $\mathrm{CVP}(\mathcal{L}_{\text{bot}}, \boldsymbol{t}_i)$ for all $\boldsymbol{t}_i \quad \Rightarrow \quad \mathrm{CVPP}$.

Solving SVP via CVPP (Part 2)

- We split $\boldsymbol{s}$ as $\boldsymbol{s} = \boldsymbol{s}_{\mathrm{bot}} + \boldsymbol{s}_{\mathrm{top}}$ where
  $\boldsymbol{s}_{\mathrm{bot}} = x_1 \boldsymbol{b}_1 + \cdots + x_{n-k} \boldsymbol{b}_{n-k} \in \mathcal{L}_{\mathrm{bot}}$ and
  $\boldsymbol{s}_{\mathrm{top}} = x_{n-k+1} \boldsymbol{b}_{n-k+1} + \cdots + x_n \boldsymbol{b}_n \in \mathcal{L}_{\mathrm{top}}$.
- Two cases:
  - If $\boldsymbol{s}_{\mathrm{top}} = \mathbf{0}$ then $\boldsymbol{s} = \mathrm{SVP}(\mathcal{L}_{\mathrm{bot}})$.
  - If $\boldsymbol{s}_{\mathrm{top}} \neq \mathbf{0}$ then $\boldsymbol{s} = \boldsymbol{s}_{\mathrm{top}} - \mathrm{CVP}(\mathcal{L}_{\mathrm{bot}}, \boldsymbol{s}_{\mathrm{top}})$.
- The vector $\boldsymbol{s}_{\mathrm{top}}$ will be one of the vectors $\boldsymbol{t}_i$ in the enumeration tree. We do not know in advance which one.
- Solve $\mathrm{CVP}(\mathcal{L}_{\mathrm{bot}}, \boldsymbol{t}_i)$ for all $\boldsymbol{t}_i \quad \Rightarrow \quad \mathrm{CVPP}$.
- Keep the shortest $\boldsymbol{t}_i - \mathrm{CVP}(\mathcal{L}_{\mathrm{bot}}, \boldsymbol{t}_i)$ as $\boldsymbol{s}$.

Solving SVP via CVPP (Part 2)



where $\boldsymbol{w}_i = \mathrm{CVP}(\mathcal{L}_{\mathrm{bot}}, \boldsymbol{t}_i)$

Hybrid 1 (sieve, enumerate–and–slice)

- Step 1: Generate a list $L \subset \mathcal{L}_{\mathrm{bot}}$ (running a lattice sieve on $\mathcal{L}_{\mathrm{bot}}$).
- Step 2: Run enumeration in $\mathcal{L}_{\mathrm{top}}$, where for each leaf $\boldsymbol{t}_i \in \mathcal{L}_{\mathrm{top}}$ run the randomized slicer to find the closest vector $\mathrm{CVP}(\boldsymbol{t}_i) \in \mathcal{L}_{\mathrm{bot}}$.
- Output the shortest vector $\boldsymbol{t}_i - \mathrm{CVP}(\boldsymbol{t}_i)$ found.

Hybrid 1 (sieve, enumerate–and–slice)

- Step 1: Generate a list $L \subset \mathcal{L}_{\text{bot}}$ (running a lattice sieve on $\mathcal{L}_{\text{bot}}$).
- Step 2: Run enumeration in $\mathcal{L}_{\text{top}}$, where for each leaf $\boldsymbol{t}_i \in \mathcal{L}_{\text{top}}$ run the randomized slicer to find the closest vector $\text{CVP}(\boldsymbol{t}_i) \in \mathcal{L}_{\text{bot}}$.
- Output the shortest vector $\boldsymbol{t}_i - \text{CVP}(\boldsymbol{t}_i)$ found.

Balancing and minimizing the costs between the two steps leads to a choice of $k = \alpha n / \log_2 d$ where $\alpha < 0.0570$.

Hybrid 1 (sieve, enumerate–and–slice)

- Step 1: Generate a list $L \subset \mathcal{L}_{\mathrm{bot}}$ (running a lattice sieve on $\mathcal{L}_{\mathrm{bot}}$).
- Step 2: Run enumeration in $\mathcal{L}_{\mathrm{top}}$, where for each leaf $\boldsymbol{t}_i \in \mathcal{L}_{\mathrm{top}}$ run the randomized slicer to find the closest vector $\mathrm{CVP}(\boldsymbol{t}_i) \in \mathcal{L}_{\mathrm{bot}}$.
- Output the shortest vector $\boldsymbol{t}_i - \mathrm{CVP}(\boldsymbol{t}_i)$ found.

Balancing and minimizing the costs between the two steps leads to a choice of $k = \alpha n / \log_2 d$ where $\alpha < 0.0570$.

Proposition (Heuristic result 1)

Let be k as above and let $\mathrm{T}_1^{(n)}$ and $\mathrm{S}_1^{(n)}$ denote the overall time and space complexities of the sieve, enumerate–and–slice hybrid algorithm in dimension n. Then:

$$\mathrm{T}_1^{(n)} = \mathrm{T}_{\mathrm{sieve}}^{(n-k)} \cdot (1 + o(1)), \qquad \mathrm{S}_1^{(n)} = \mathrm{S}_{\mathrm{sieve}}^{(n-k)} \cdot (1 + o(1)).$$

Hybrid 2 (sieve, enumerate, slice)

- Step 1: Generate a list $L \subset \mathcal{L}_{\text{bot}}$ (running a lattice sieve on $\mathcal{L}_{\text{bot}}$).
- Step 2: Enumerate all nodes $\boldsymbol{t}_i \in \mathcal{L}_{\text{top}}$ at depth k and store them in a list of targets $T \subset \mathcal{L}_{\text{top}}$.
- Step 3: Run the batched randomized slicer to solve CVP on $\mathcal{L}_{\text{bot}}$ for all targets $\boldsymbol{t}_i \in T$.
- Output the shortest vector $\boldsymbol{t}_i - \text{CVP}(\boldsymbol{t}_i)$ found.

Hybrid 2 (sieve, enumerate, slice)

- Step 1: Generate a list $L \subset \mathcal{L}_{\mathrm{bot}}$ (running a lattice sieve on $\mathcal{L}_{\mathrm{bot}}$).
- Step 2: Enumerate all nodes $\boldsymbol{t}_i \in \mathcal{L}_{\mathrm{top}}$ at depth k and store them in a list of targets $T \subset \mathcal{L}_{\mathrm{top}}$.
- Step 3: Run the batched randomized slicer to solve CVP on $\mathcal{L}_{\mathrm{bot}}$ for all targets $\boldsymbol{t}_i \in T$.
- Output the shortest vector $\boldsymbol{t}_i - \mathrm{CVP}(\boldsymbol{t}_i)$ found.

Proposition (Heuristic result 2)

Let $k = \alpha n / \log_2 n$ *with* $\alpha < \log_2(\frac{13}{12}) = 0.1154\dots$.
Let $\mathrm{T}_2^{(n)}$ *and* $\mathrm{S}_2^{(n)}$ *denote the overall time and space complexities of the batched sieve, enumerate, slice hybrid algorithm in dimension n. Then:*

$$\mathrm{T}_2^{(n)} = \mathrm{T}_{\mathrm{sieve}}^{(n-k)} \cdot (1 + o(1)), \qquad \mathrm{S}_2^{(n)} = \mathrm{S}_{\mathrm{sieve}}^{(n-k)} \cdot (1 + o(1)).$$

Further Hybrids

A basis $\mathbf{B}$ could be partitioned as $\mathbf{B} = \mathbf{B}_{\mathrm{bot}} \cup \mathbf{B}_{\mathrm{mid}} \cup \mathbf{B}_{\mathrm{top}}$. The three bases $\mathbf{B}_{\mathrm{bot}}$, $\mathbf{B}_{\mathrm{mid}}$, and $\mathbf{B}_{\mathrm{top}}$ generate lattices $\mathcal{L}_{\mathrm{bot}}, \mathcal{L}_{\mathrm{mid}}, \mathcal{L}_{\mathrm{top}}$ such that $\mathcal{L} = \mathcal{L}_{\mathrm{bot}} \oplus \mathcal{L}_{\mathrm{mid}} \oplus \mathcal{L}_{\mathrm{top}}$.

Further Hybrids

A basis $\mathbf{B}$ could be partitioned as $\mathbf{B} = \mathbf{B}_{\mathrm{bot}} \cup \mathbf{B}_{\mathrm{mid}} \cup \mathbf{B}_{\mathrm{top}}$. The three bases $\mathbf{B}_{\mathrm{bot}}$, $\mathbf{B}_{\mathrm{mid}}$, and $\mathbf{B}_{\mathrm{top}}$ generate lattices $\mathcal{L}_{\mathrm{bot}}, \mathcal{L}_{\mathrm{mid}}, \mathcal{L}_{\mathrm{top}}$ such that $\mathcal{L} = \mathcal{L}_{\mathrm{bot}} \oplus \mathcal{L}_{\mathrm{mid}} \oplus \mathcal{L}_{\mathrm{top}}$.

- Step 1: Generate a list $L \subset \mathcal{L}_{\mathrm{mid}}$ (running a lattice sieve on $\mathcal{L}_{\mathrm{mid}}$).

Further Hybrids

A basis $\mathbf{B}$ could be partitioned as $\mathbf{B} = \mathbf{B}_{\mathrm{bot}} \cup \mathbf{B}_{\mathrm{mid}} \cup \mathbf{B}_{\mathrm{top}}$. The three bases $\mathbf{B}_{\mathrm{bot}}$, $\mathbf{B}_{\mathrm{mid}}$, and $\mathbf{B}_{\mathrm{top}}$ generate lattices $\mathcal{L}_{\mathrm{bot}}, \mathcal{L}_{\mathrm{mid}}, \mathcal{L}_{\mathrm{top}}$ such that $\mathcal{L} = \mathcal{L}_{\mathrm{bot}} \oplus \mathcal{L}_{\mathrm{mid}} \oplus \mathcal{L}_{\mathrm{top}}$.

- Step 1: Generate a list $L \subset \mathcal{L}_{\mathrm{mid}}$ (running a lattice sieve on $\mathcal{L}_{\mathrm{mid}}$).
- Step 2:
  - Enumerate all nodes $\boldsymbol{t} \in \mathcal{L}_{\mathrm{top}}$.
  - For each $\boldsymbol{t}$ run the slicer with the list L to find close vectors $\boldsymbol{v} \in \mathcal{L}_{\mathrm{mid}}$.
  - For each pair $\boldsymbol{t}, \boldsymbol{v}$ add the vector $\boldsymbol{t} - \boldsymbol{v}$ to an output list S.

Further Hybrids

A basis $\mathbf{B}$ could be partitioned as $\mathbf{B} = \mathbf{B}_{\mathrm{bot}} \cup \mathbf{B}_{\mathrm{mid}} \cup \mathbf{B}_{\mathrm{top}}$. The three bases $\mathbf{B}_{\mathrm{bot}}$, $\mathbf{B}_{\mathrm{mid}}$, and $\mathbf{B}_{\mathrm{top}}$ generate lattices $\mathcal{L}_{\mathrm{bot}}, \mathcal{L}_{\mathrm{mid}}, \mathcal{L}_{\mathrm{top}}$ such that $\mathcal{L} = \mathcal{L}_{\mathrm{bot}} \oplus \mathcal{L}_{\mathrm{mid}} \oplus \mathcal{L}_{\mathrm{top}}$.

- Step 1: Generate a list $L \subset \mathcal{L}_{\mathrm{mid}}$ (running a lattice sieve on $\mathcal{L}_{\mathrm{mid}}$).
- Step 2:
  - Enumerate all nodes $\boldsymbol{t} \in \mathcal{L}_{\mathrm{top}}$.
  - For each $\boldsymbol{t}$ run the slicer with the list L to find close vectors $\boldsymbol{v} \in \mathcal{L}_{\mathrm{mid}}$.
  - For each pair $\boldsymbol{t}, \boldsymbol{v}$ add the vector $\boldsymbol{t} - \boldsymbol{v}$ to an output list S.
- Step 3: Extend each vector $\boldsymbol{s}' \in S$ to a candidate solution $\boldsymbol{s} \in \mathcal{L}$ by running Babai's nearest plane algorithm.
- Output the shortest lifted vector.

Further Hybrids

This hybrid depends on

Assumption (Hybrid assumption)

The list S, output by the slicer, contains the $2^{(\alpha+\log_2(16/13))\cdot n/2+o(n)}$ *shortest lattice vectors of* $\mathcal{L}_{\mathrm{mid}} \oplus \mathcal{L}_{\mathrm{top}}$.

Further Hybrids

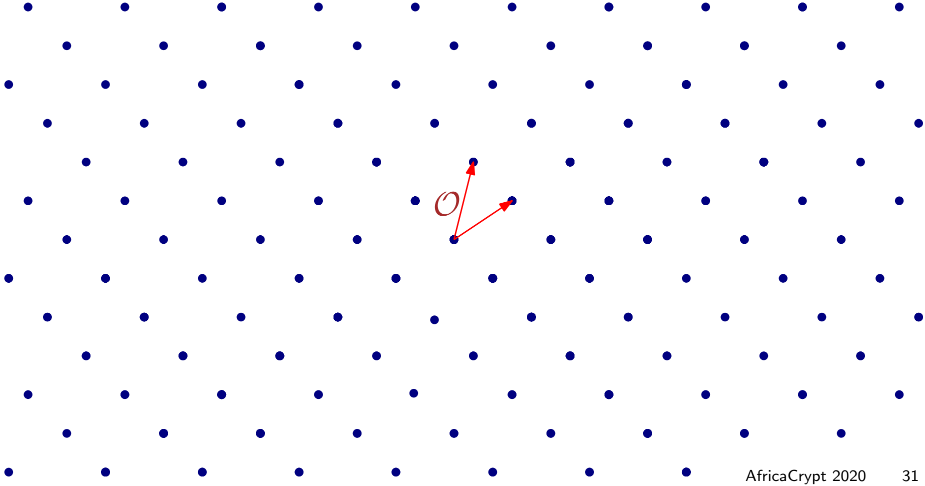
This hybrid depends on

Assumption (Hybrid assumption)

The list S, output by the slicer, contains the $2^{(\alpha+\log_2(16/13))\cdot n/2+o(n)}$ shortest lattice vectors of $\mathcal{L}_{\mathrm{mid}} \oplus \mathcal{L}_{\mathrm{top}}$.

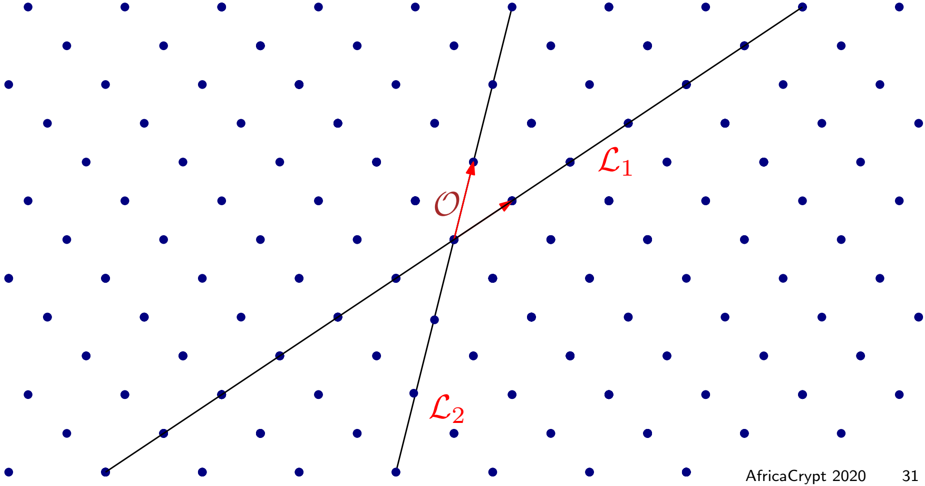
Léo Ducas and Wessel van Woerden later informed us that counterexamples can be found where S might only contain at most an exponentially small fraction of the shortest vectors of $\mathcal{L}_{\mathrm{mid}} \oplus \mathcal{L}_{\mathrm{top}}$.

Visualisation of the assumption



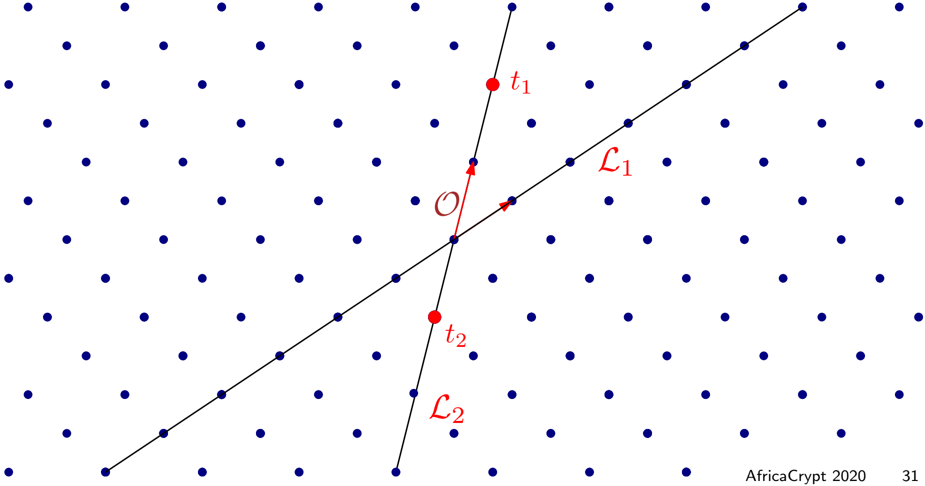
Visualisation of the assumption

- Split $\mathcal{L}$ as $\mathcal{L} = \mathcal{L}_1 \oplus \mathcal{L}2$.



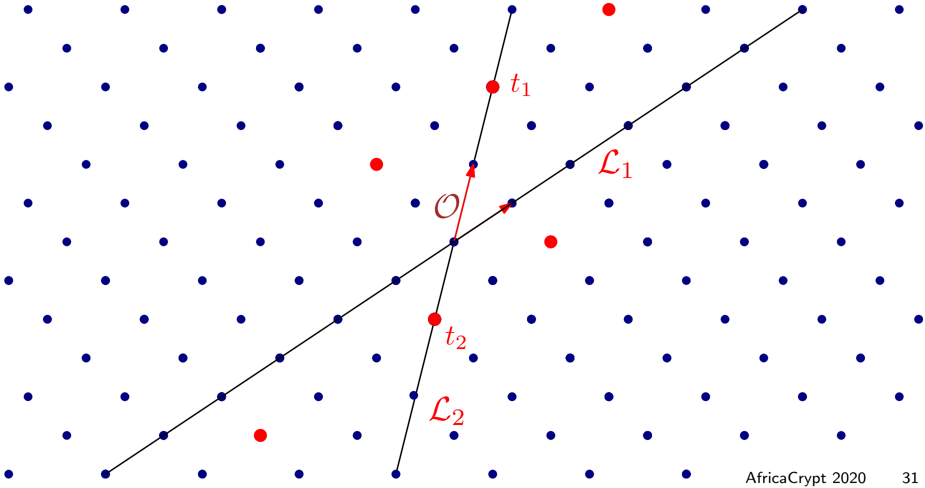
Visualisation of the assumption

- Split $\mathcal{L}$ as $\mathcal{L} = \mathcal{L}_1 \oplus \mathcal{L}2$.
- Enumerate targets in $\mathcal{L}2$.



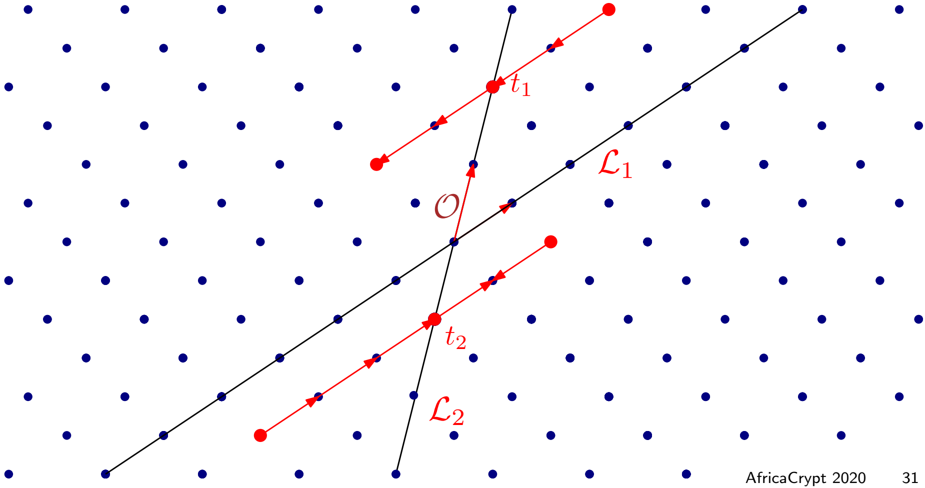
Visualisation of the assumption

- Split $\mathcal{L}$ as $\mathcal{L} = \mathcal{L}_1 \oplus \mathcal{L}2$.
- Enumerate targets $\boldsymbol{t}_i$ in $\mathcal{L}2$.
- Randomise the $\boldsymbol{t}_i$ using vectors in $\mathcal{L}_1$.



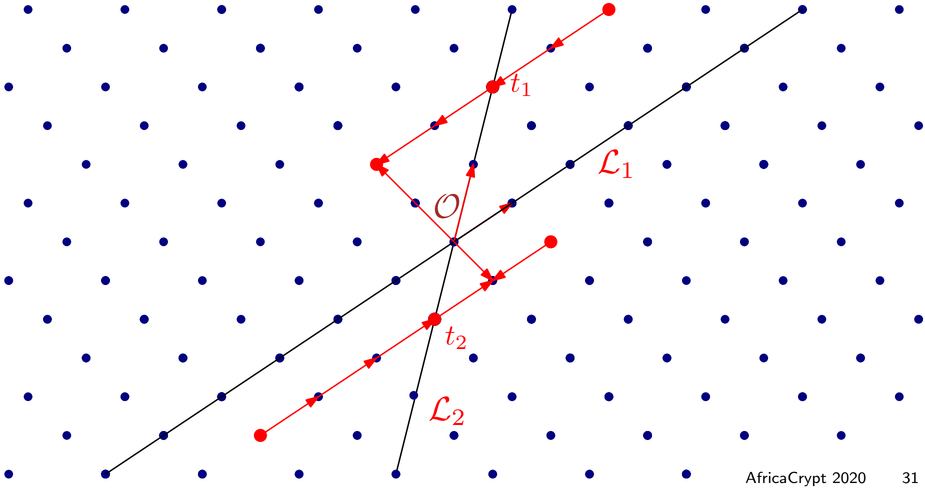
Visualisation of the assumption

- Reduce all the randomised vectors by short vectors in $\mathcal{L}_1$.



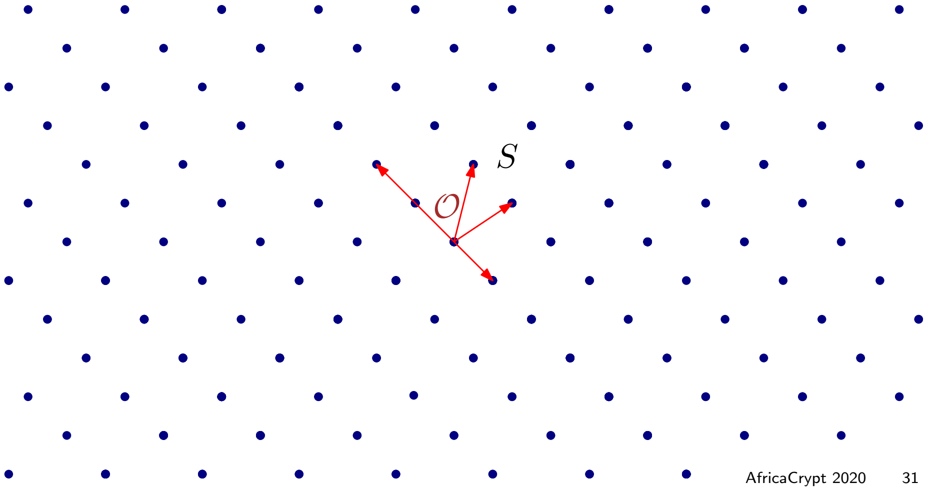
Visualisation of the assumption

- Reduce all the randomised vectors by short vectors in $\mathcal{L}_1$.



Visualisation of the assumption

- Reduce all the randomised vectors by short vectors in $\mathcal{L}_1$.
- Keep the resulting vectors as the set S.



Experimental results

Parameters		BKZ	— Sieve —		— Enum —		— Slice —			Total
d	k	$\mathrm{T}_{\mathrm{BKZ}}^{(d-10)}$	$\lvert L\rvert$	$\mathrm{T}_{\mathrm{sieve}}^{(d-k)}$	$\lvert T\rvert$	$\mathrm{T}_{\mathrm{enum}}^{(k)}$	$\mathrm{T}_{\mathrm{iter}}^{(d-k)}$	$\mathrm{p}_{\mathrm{iter}}^{-1}$	$\mathrm{T}_{\mathrm{slice}}^{(d-k)}$	$\mathrm{T}_{\mathrm{hyb}}^{(d)}$
60	0	4s	18k	19s	-	-	-	-	-	23s
	1	4s	16k	16s	5	0s	3.2ms	830	13s	33s
	2	4s	13k	12s	30	0s	2.7ms	530	43s	59s
	3	4s	12k	9s	155	0s	2.4ms	760	280s	293s
	1+1	4s	13k	12s	4	0s	3.0ms	500	6s	51s
			(16k)	(0s)	5	0s	3.2ms	1820	29s	
65	0	8s	37k	78s	-	-	-	-	-	1m
	1	8s	32k	57s	5	0s	6.8ms	12.5k	7m	8m
	2	8s	28k	44s	37	0s	6.6ms	2.9k	12m	13m
	3	8s	24k	36s	215	0s	5.6ms	2.9k	58m	59m
	1+1	8s	28k	44s	4	0s	6.6ms	1.1k	0.5m	6m
			(32k)	(0s)	5	0s	6.8ms	6.7k	4m	
70	0	1m	76k	5m	-	-	-	-	-	6m
	1	1m	65k	4m	6	0m	20ms	17k	35m	40m
	2	1m	57k	3m	46	0m	16ms	1k	12m	16m
	3	1m	49k	2m	293	0m	13ms	6k	381m	384m
	1+1	1m	57k	3m	5	0m	15ms	2k	2m	43m
			(65k)	(0m)	5	0m	18ms	25k	37m	
75	0	2m	155k	22m	-	-	-	-	-	0.4h
	1	2m	134k	16m	6	0m	40ms	25k	2h	2h
	2	2m	116k	11m	50	0m	48ms	20k	13h	14h
	3	2m	101k	8m	366	0m	30ms	12k	37h	37h
	1+1	2m	116k	11m	5	0m	35ms	4k	0.2h	>8h
			(134k)	(0m)	6	0m	41ms	>100k	>7h	
80	0	14m	320k	74m	-	-	-	-	-	1.5h
	1	14m	275k	58m	7	0m	95ms	>100k	>18h	>20h
	2	14m	240k	45m	64	0m	74ms	>50k	>66h	>67h
	3	14m	205k	36m	506	0m	66ms	>50k	>19d	>19d

Thank you!