6.876 Advanced Topics in Cryptography: Lattices	September 30, 2015
Lecture 7	
Lecturer: Vinod Vaikuntanathan	*Scribe: Adam Sealfon*

Today and next time:

- Ajtai-Kumar-Sivakumar (AKS) Algorithm. $2^{O(n)}$ time, randomized, SVP (2000)
- Micciancio-Voulgaris (MV) algorithm. $2^{O(n)}$ time, deterministic, CVP (2010).

Two open problems:

Problem 1. *Can we do integer programming in $2^{O(n)}$ time? Using LLL we can solve integer programming in $2^{O(n^3)}$ time, and Kannan improved this to $n^{O(n)}$ time. AKS/MV/ADRS solve CVP/SVP in $2^{O(n)}$ time, but we don't know how to extend this to integer programming.*

Problem 2. *Can we solve SVP in $2^{O(n)}$ time but $2^{o(n)}$ space? But Kannan and LLL require* poly(n) *space, but the $2^{O(n)}$-time algorithms we know require 2^n space as well.*

1 Sieving

At a high level, the AKS algorithm for finding the shortest vector in a lattice works by taking a large sphere or box and choosing many (e.g. $\sim 2^{10n}$) random lattice points in the box by taking random combinations of the basis. The points are then refined through an iterative routine called sieving.

The sieving procedure $\mathsf{Sieve}(Y)$ will have the following properties. $\mathsf{Sieve}(Y) = C \subseteq Y$ such that:

- For all $y \in Y$, there exists $y' \in C$ such that $\|y - y'\| \leq R/2$.
- $|C| \leq 5^n$.

The idea is that we start with a bunch of points and keep reducing by sieving, eventually obtaining the shortest vector.

We implement $\mathsf{Sieve}(Y)$ as follows.

$\mathsf{Sieve}(Y)$:

1. $C = \emptyset$.
2. For every $y \in Y$,
   - If y is $R/2$-close to some $y' \in C$, then $\eta(y) = y'$.
   - Else $C = C \cup \{y\}$.

We use a packing argument to show that the set C is small. Any pair of points in C must be at least $R/2$ units apart, so the balls of radius $R/4$ centered at the points of C are all disjoint and are contained in the ball of radius $R + R/4$ centered at the origin. Consequently we have that

$$|C| < \frac{\mathsf{Vol}(B(0, 5R/4))}{\mathsf{Vol}(B(0, R/4))} = 5^n.$$

2 The AKS Algorithm

We now present the main AKS algorithm. We assume for now that $\lambda_1 \in [2, 3)$. We will remove this assumption by running the algorithm on a sequence of intervals of geometrically increasing size (i.e. the intervals $[3, 4.5)$, $[4.5, 6.75)$, and so on). Using LLL we can estimate λ_1 to within a factor of 2^n, so we will need to consider only $O(n)$ such intervals.

INITIALIZATION:

1. Let $R_0 = n \cdot \max \|b_i\|$ and $N = 2^{8n} \log R_0$.
2. Choose $x_1, \ldots, x_N \leftarrow B(0, 2)$.
3. Let $y_i = x_i \bmod \mathcal{P}(B)$ for each $i \in [N]$.
4. Let $R = R_0$, $X = \{x_i\}_{i \in [N]}$, $Y = \{y_i\}_{i \in [N]}$, $Z = \{(x_i, y_i)\}_{i \in [N]}$.

SIEVING:

While $R > 6$:

5. $C \leftarrow \mathsf{Sieve}(Y)$
6. For all $y_i \in Y$:
   - If $y_i \in C$, discard it.
   - Else, let $y_i \leftarrow y_i - (y_{\eta(i)} - x_{\eta(i)})$, where $\eta(i)$ denotes the index of the cluster center of i.
7. Set $R \leftarrow \frac{R}{2} + 2$ and repeat sieving step.

OUTPUT:

8. For all $(x_i, y_i), (x_j, y_j) \in Z$, output the shortest nonzero vector $(y_i - x_i) - (y_j - x_j)$.

We note two things about the sieving step. First, the sieving step only uses x_i values for cluster centers, which are points which are discarded. The values x_i for points which remain are not used. Second, the value $y_{\eta(i)} - x_{\eta(i)}$ which is subtracted from y_i in the sieving step is a lattice point. This preserves the invariant that $y_i - x_i$ is always a lattice point.

3 Analysis

We first prove some basic invariants about the algorithm. Then, rather than analyzing the behavior of the algorithm's inner routine directly, we will show that any algorithm which satisfies the invariants and does not inspect the values of the x_is must produce the correct output with high probability. In effect, we treat the sieving subroutine as an adversary and show that any adversary which satisfies the invariants below is unable to mess up the algoirthm.

Claim 3. $y_i - x_i \in \mathcal{L}(B)$.

Proof. This is true after the initialization step, since $y_i = x_i \bmod \mathcal{P}(B)$. In the sieving step, y_i is replaced by $y_i - (y_{\eta(i)} - x_{\eta(i)})$ and x_i is unchanged, so the difference after an iteration of sieving is

$$\overline{y_i} - \overline{x_i} = y_i - (y_{\eta(i)} - x_{\eta(i)}) - x_i = (y_i - x_i) - (y_{\eta(i)} - x_{\eta(i)}) \in \mathcal{L}(B)$$

since it is the difference of two lattice vectors. □

Claim 4. *After an iteration of sieving,* $\|y_i\| \leq \frac{R}{2} + 2$.

Proof. By the Triangle Inequality,

$$\|\overline{y_i}\| = \|y_i - (y_{\eta(i)} - x_{\eta(i)})\| \leq \|y_i - y_{\eta(i)}\| + \|x_{\eta(i)}\| \leq \frac{R}{2} + 2.$$

□

Claim 5. *The number of iterations of sieving is at most* $2 \log R_0$.

Proof. If $R/2 > 6$ then after two iterations, R decreases to $((R/2)+2)/2+2 = R/4+3 < R/2$. Consequently after fewer than $2 \log R_0$ iterations, the sieving step will terminate. □

Claim 6. *At the conclusion of the sieving phase,* Z *consists of at least* 2^{7n} *pairs of points* (x_i, y_i) *such that* $\|y_i - x_i\| \leq 8$.

Proof. In each iteration of sieving, at most 5^n points are discarded. Consequently, after $2 \log R_0$ iterations, the number of points remaining is at least

$$|Z| \geq N - 2 \cdot 5^n \log R_0 = (2^{8n} - 2 \cdot 5^n) \log R_0 \geq 2^{7n} \log R_0.$$

□

3.1 A mental experiment

Consider the following mental experiment. Say there's an all-powerful individual A who knows a shortest vector v in $\mathcal{L}$. Then if the value $x_i \in B(0, 2)$ falls in a particular subregion, there are two possible values of x_i that produce the same y_i modulo the lattice. These two values differ by v. Consequently, even if the adversary is unbounded computationally, it cannot distinguish between these two ways of obtaining the vector y_i. Therefore, now considering the algorithm itself to be the adversary, if it does not look at the value x_i then the value $w_i = y_i - x_i$ produced at the end of the algorithm is equally likely to be two different vectors which differ by v. We will use this argument to prove the correctness of the algorithm.

Define regions C_1 and C_2 as follows, where v is a shortest vector in $\mathcal{L}(B)$. Let $C_1 = B(0, 2) \cap B(-v, 2)$ is the set of points which are distance at most 2 from both 0 and $-v$, and $C_2 = B(0, 2) \cap B(v, 2)$ is the set of points which are distance at most 2 from both 0 and $+v$.

Define the operation $\mathsf{Flip}(y)$ as follows.

$$\mathsf{Flip}(y) = \begin{cases} y & \text{if } y \notin C_1 \cup C_2 \\ y - v & \text{if } y \in C_2 \\ y + v & \text{if } y \in C_1 \end{cases}$$

The Flip operation swaps C_1 and C_2, keeping other points fixed. The idea is that if we flip half of the points x_i at the start of the algorithm, the distribution of the x_i's will be unchanged, so the adversary (and the algorithm) will be unchanged. Consequently for any y_i corresponding to some $x_i \in C_1 \cup C_2$, depending on whether $x_i \in C_1$ or x_i is the the corresponding vector in C_2, $y_i - x_i$ is equally likely to result in either of a pair of vectors which differ by v. Therefore, if we repeat enough times to guarantee that the same short lattice point is obtained many times, we should also find a pair of points whose difference is the shortest vector.

Lemma 7. $\Pr[x \in C_1 \cup C_2] \geq 2 \cdot 2^{-2n}$.

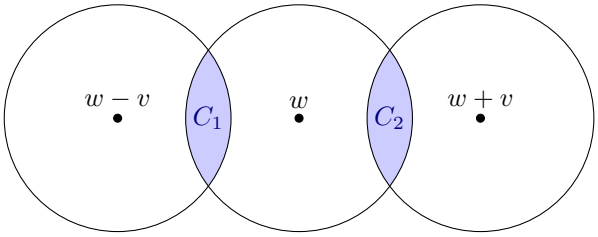

Figure 1: The regions C_1 and C_2 of $B(w, 2)$ used by the Flip operation, which is defined below. Flip swaps points in C_1 and C_2, leaving the remainder of $B(0, 2)$ fixed. $C_1 \subset B(0, 2)$ is the set of points which are distance at most 2 from both 0 and $-v$, and $C_2 \subset B(0, 2)$ is the set of points which are distance at most 2 from both 0 and $+v$.

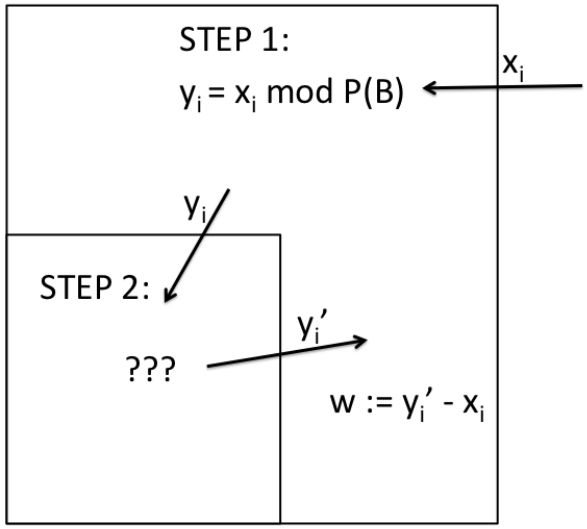

Figure 2: An outline of the structure of the AKS algorithm. The algorithm starts with uniformly random points $x_i \in B(0, 2)$ and produces $y_i = x_i \bmod \mathcal{L}(B)$. Without referring to the original value x_i, the algorithm produces some y_i' of length at most 6 such that $y_i' - x_i \in \mathcal{L}$. We will show that with noticeable probability the original point x_i lies in a special region $C_1 \cup C_2$ defined below. The sets C_1 and C_2 are translations of each other by the shortest vector v, so if x_i drawn from one of the regions, then $w = y_i' - x_i$ is equally likely to be either of a pair of vectors which differ by v.

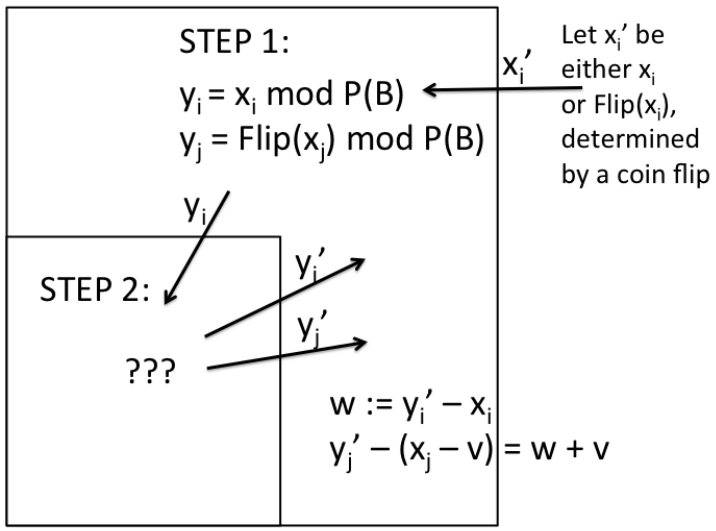

Figure 3: The Flip operation is useful for analyzing the AKS algorithm. The view of the adversary (and the algorithm) is the same whether or not x_i was flipped. Consequently the algorithm is equally likely to produce vectors w and $w + v$.

Proof. Since $\|v\| = \lambda_1 \leq 3$, we have that $B(v, 1/2) \subset C_2$. Consequently since C_1 and C_2 are of equal size and are disjoint, we have that

$$\Pr[x \in C_1 \cup C_2] = 2 \cdot \Pr[x \in C_2] = \frac{2 \cdot \text{Vol}(C_1)}{\text{Vol}(B(0,2))} \geq \frac{2 \cdot \text{Vol}(B(-v, 1/2))}{\text{Vol}(B(0,2))} = 2 \cdot 4^{-n}.$$

□

Call such a point $x \in C_1 \cup C_2$ "good." As a consequence of Lemma 7, roughly $2^{6n} \log R_0$ of the original points x_i are good, and roughly $2^{5n} \log R_0$ good points survive to the end of the sieving phase.

Lemma 8. *The number of lattice points in $B(0, 8)$ is at most 9^n.*

Proof. By assumption the shortest nonzero vector in the lattice has length at least 2. Consequently the balls of radius 1 around lattice points in $B(0, 8)$ are disjoint. But these balls are all contained in $B(0, 9)$, so the number of lattice points in $B(0, 8)$ is at most

$$\frac{\text{Vol}(B(0,9))}{\text{Vol}(B(0,1))} = 9^n.$$

□

Consequently roughly $2^{5n}/9^n \approx 2^{1.8n}$ surviving points y_i are close to the same lattice point and have a corresponding x_i which is good. These values x_i are equally likely to be in C_1 and C_2, so with high probability at least one x_i is in each of the two sets. But then there will be a pair of indices i, j for which $(y_i - x_i) - (y_j - x_j) = v$, and so the algorithm finds the shortest vector as desired.

4 A preview of next time: the MV algorithm for CVP

Next class we will discuss the algorithm of Micciancio and Voulgaris which solves the closest vector problem deterministically in $2^{O(n)}$ time. The main tool used in the MV algorithm is the Voronoi cell $V(\mathcal{L})$ of a lattice, which is the polytope consisting of the points which are closer to 0 than to any other lattice vector.

Definition 9. *The* V*oronoi cell* $V(\mathcal{L})$ *of a lattice* $\mathcal{L}$ *is the set*

$$\{x \in \mathbb{R}^n : \|x\| \leq \|x - v\| \forall v \in \mathcal{L}(B)\}.$$

The following theorem is due to Voronoi.

Theorem 10. *The number of* $(n-1)$*-dimensional facets of the Voronoi cell of an* n*-dimensional lattice is at most* $2(2^n - 1)$ *.*

The proof will follow from the following reductions:

- $\mathsf{Voronoi}_n \overset{2^n}{\leq} \mathsf{CVP}_n$, that is, we can solve $\mathsf{Voronoi}_n$ by solving at most 2^n instances of CVP_n.
- $\mathsf{CVP}_n \overset{2^n}{\leq} \mathsf{Voronoi}_n$.
- $\mathsf{CVP}_n \overset{2^n}{\leq} \mathsf{CVP}_{n-1}$.

Putting these together, we have:

$$\mathsf{Voronoi}_n \overset{2^n}{\leq} \mathsf{CVP}_n \overset{2^n}{\leq} \mathsf{CVP}_{n-1} \overset{2^n}{\leq} \mathsf{Voronoi}_{n-1}.$$

It looks like we haven't made any progress, since solving the n-dimensional problem requires solving $2^{O(n)}$ instances of the $n-1$-dimensional problem. However, the same Voronoi cell of dimension $(n-1)$ is used by each of the subproblems! Consequently we can compute this Voronoi cell once and use it each time it is needed, so the recurrence is $T(n) = 2^{O(n)} + T(n-1)$ rather than $T(n) = 2^{O(n)} \cdot T(n-1)$. This will result in a total running time $2^{O(n)}$.

References

[1] M. Ajtai, R. Kumar, and D. Sivakumar. A sieve algorithm for the shortest lattice vector problem. In *Proc. 33rd ACM Symp. on Theory of Computing*, pages 601–610, 2001.

[2] M. Khanevsky and O. Regev, Scribe notes for Oded Regev's class on Lattices in Computer Science, Lecture 8, `http://www.cims.nyu.edu/~regev/teaching/lattices_fall_2004/ln/svpalg.pdf`, 2004.

[3] D. Micciancio and P. Voulgaris. A deterministic single exponential time algorithm for most lattice problems based on Voronoi cell computations. In *Proc. 42nd ACM Symp. on Theory of Computing*, 2010.