

Криптография и решётки

Сергей Николенко

Криптография — CS Club, осень 2009

Outline

- 1 Решётки на службе криптоанализа
 - Общие сведения
 - Алгоритм LLL
 - Применения алгоритма LLL в криптоанализе
- 2 Решётки на службе криптографии
 - Идея
 - Конструкция хеш-функций Ajtai
 - Конструкция криптосистемы Ajtai-Dwork

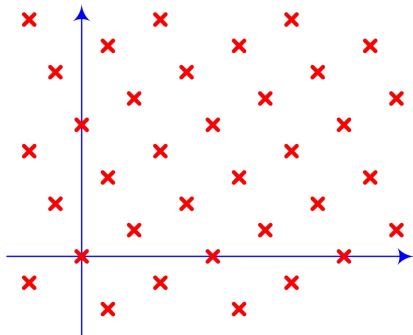
Решётки

- Будем обозначать скалярное произведение как $\langle x, y \rangle$.
- Пусть $B = \{b_1, \dots, b_m\}$ — набор линейно независимых векторов в \mathbb{R}^n . Тогда *решётка* размерности m — это набор линейных комбинаций b_i с целыми коэффициентами:

$$L = \mathbb{Z}b_1 + \mathbb{Z}b_2 + \dots + \mathbb{Z}b_m.$$

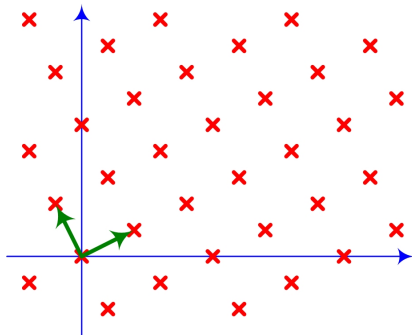
Решётки

- Вот типичная решётка.



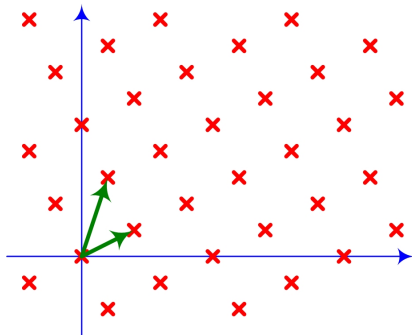
Решётки

- Вот типичная решётка.
- Базис решётки — это множество линейно независимых векторов, её порождающих.



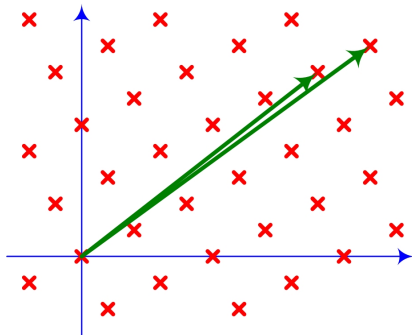
Решётки

- Вот типичная решётка.
- Базис решётки — это множество линейно независимых векторов, её порождающих.
- Вот другой базис той же решётки.



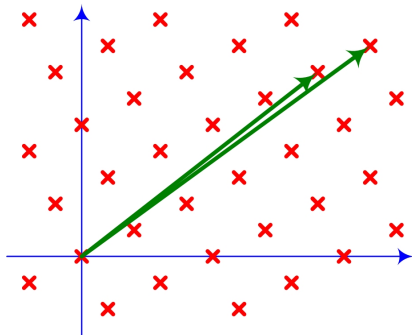
Решётки

- Вот типичная решётка.
- Базис решётки — это множество линейно независимых векторов, её порождающих.
- Вот другой базис той же решётки.
- А вот ещё один.



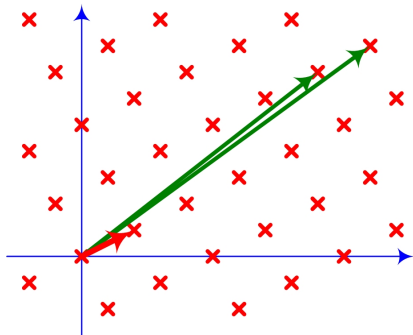
Решётки

- Базис может состоять из очень длинных векторов, даже если в решётке есть короткие.
- Найти короткий вектор решётки — сложная задача; её мы и будем рассматривать.



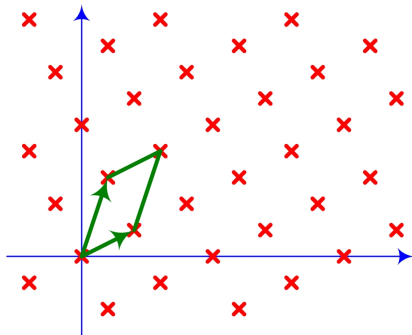
Кратчайший вектор

- Обозначим кратчайший вектор через $\lambda_1(L)$.
- Многомерное обобщение: $\lambda_k(L)$ — минимальное r , для которого размерность решётки внутри шара радиуса r больше либо равна k .



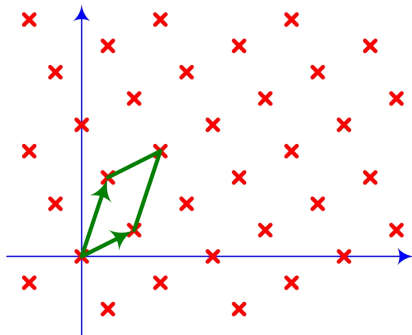
Определитель

- *Фундаментальный параллелепипед* образован базисом.
- Важный инвариант — его объём $\det L$, т.е. определитель матрицы, составленной из векторов базиса.



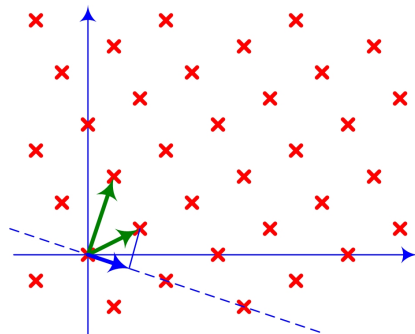
Определитель

- Он от базиса не зависит, т.к. разные базисы решётки эквивалентны, только если они преобразуются друг в друга не меняющими $\det L$ преобразованиями.



Ортогонализация

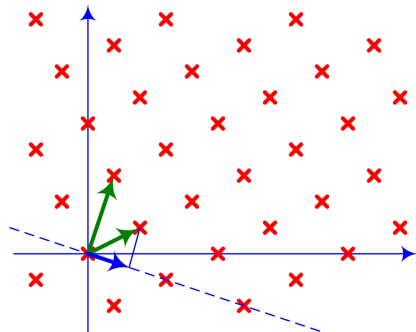
- Ортогонализация по методу Грама-Шмидта: из базиса (пространства) делает ортогональный базис.
- $b_1^* = b_1$, дальше каждый последующий ортогонален пространству предыдущих.
- Для этого вычитаем проекции предыдущих базисных векторов.



Ортогонализация

- В частности, при этом $\det L$ не поменяется, а получится ортогональный базис:

$$\det L = \|b_1^*\| \|b_2^*\| \dots \|b_n^*\|.$$



Решётки

- Рассмотрим решётку L с базисом $B = \{b_1, \dots, b_m\}$.
Определим, по методу ортогонализации Грама-Шмидта,

$$\mu_{ij} = \frac{\langle b_i, b_j^* \rangle}{\langle b_j^*, b_j^* \rangle}, \quad 1 \leq j < i \leq n,$$

$$b_i^* = b_i - \sum_{j=1}^{i-1} \mu_{ij} b_j^*, \quad 1 \leq i \leq m.$$

- Тогда $\langle b_i^*, b_j \rangle = 0$ для всех $j < i$, и $\langle b_i^*, b_i \rangle = \langle b_i^*, b_i^* \rangle$.

Решётки

- Иначе говоря, если записать базис B в ортонормированном базисе, получаемом из B^* (i -й столбец матрицы — b_i в этом базисе):

$$\begin{pmatrix} \|b_1^*\| & \mu_{21}\|b_1^*\| & \mu_{31}\|b_1^*\| & \dots & \mu_{n1}\|b_1^*\| \\ 0 & \|b_2^*\| & \mu_{32}\|b_2^*\| & \dots & \mu_{n2}\|b_2^*\| \\ 0 & 0 & \|b_3^*\| & \dots & \mu_{n3}\|b_3^*\| \\ \vdots & \vdots & & & \vdots \\ 0 & 0 & 0 & \dots & \|b_n^*\| \end{pmatrix}$$

Решётки

- Например, для решётки с базисом $b_1 = (2 \ 1)$ и $b_2 = (1 \ 3)$ (на рисунках): $b_1^* = b_1$,

$$\mu_{21} = \frac{\langle b_2, b_1^* \rangle}{\langle b_1^*, b_1^* \rangle} = \frac{5}{5} = 1,$$

$$b_2^* = b_2 - \mu_{21} b_1^* = b_2 - b_1 = (-1 \ 2).$$

- А для решётки с базисом $b_1 = (9 \ 7)$ и $b_2 = (11 \ 8)$ (та же решётка): $b_1^* = b_1$,

$$\mu_{21} = \frac{\langle b_2, b_1^* \rangle}{\langle b_1^*, b_1^* \rangle} = \frac{155}{130} = \frac{31}{26},$$

$$b_2^* = b_2 - \mu_{21} b_1^* = b_2 - \frac{31}{26} b_1^* = \left(\frac{7}{26} \quad -\frac{9}{26} \right).$$

- Вектор b_2^* получился гораздо короче обоих исходных;
 $\|b_2^*\| \approx 0,4385$.

Решётки

- Докажем лемму, полезную для поиска кратчайшего вектора.
- Рассмотрим базис B и произвольный вектор в L ; он представляется как Bx для некоторого вектора $x \in \mathbb{Z}^n$.
- Пусть $j \in 1..n$ — наибольшее такое j , что $x_j \neq 0$. Тогда:

$$|\langle Bx, b_j^* \rangle| = \left| \left\langle \sum_{i=1}^j x_i b_i, b_j^* \right\rangle \right| = |x_j| \langle b_j^*, b_j^* \rangle = |x_j| \|b_j^*\|^2.$$

Решётки

- С другой стороны, $\left| \langle Bx, b_j^* \rangle \right| \leq \|Bx\| \cdot \|b_j^*\|$, и
$$\|Bx\| \geq |x_j| \|b_j^*\| \geq \|b_j^*\| \geq \min_j \|b_j^*\|.$$
- Мы доказали нижнюю оценку на размер кратчайшего вектора решётки: $\lambda_1(L) \geq \min_j \|b_j^*\|$.
- В нашем примере кратчайший вектор был $(2 \ 1)$ с длиной $\sqrt{5}$, что больше $\|b_2^*\|$.

δ -LLL-редуцированные базисы

- Базис B называется δ -LLL-редуцированным (δ -LLL reduced), если

$$|\mu_{ij}| \leq \frac{1}{2} \text{ для всех } 1 \leq j < i \leq n, \text{ и}$$

$$\delta \|b_i^*\|^2 \leq \|\mu_{i+1,i} b_i^* + b_{i+1}^*\|^2 \text{ для всех } 1 \leq i < n.$$

- Часто рассматривают $\delta = \frac{3}{4}$.

δ -LLL-редуцированные базисы

- Второе свойство:

$$\delta \|b_i^*\|^2 \leq \|\mu_{i+1,i} b_i^* + b_{i+1}^*\|^2 = \mu_{i+1,i}^2 \|b_i^*\|^2 + \|b_{i+1}^*\|^2, \text{ и}$$
$$\|b_{i+1}^*\|^2 \geq \left(\delta - \frac{1}{4}\right) \|b_i^*\|^2.$$

- Иначе говоря. Запишем базис B в ортонормированном базисе, получаемом из B^* (i -й столбец матрицы — b_i в этом базисе):

$$\begin{pmatrix} \|b_1^*\| & * & * & \dots & * \\ 0 & \|b_2^*\| & * & \dots & * \\ 0 & 0 & \|b_3^*\| & \dots & * \\ \vdots & \vdots & & & \vdots \\ 0 & 0 & 0 & \dots & \|b_n^*\| \end{pmatrix}$$

δ -LLL-редуцированные базисы

- Первое условие ($|\mu_{ij}| \leq \frac{1}{2}$) говорит, что

$$\begin{pmatrix} \|b_1^*\| & \leq \frac{1}{2}\|b_1^*\| & \leq \frac{1}{2}\|b_1^*\| & \dots & \leq \frac{1}{2}\|b_1^*\| \\ 0 & \|b_2^*\| & \leq \frac{1}{2}\|b_2^*\| & \dots & \leq \frac{1}{2}\|b_2^*\| \\ 0 & 0 & \|b_3^*\| & \dots & \leq \frac{1}{2}\|b_3^*\| \\ \vdots & \vdots & & & \vdots \\ 0 & 0 & 0 & \dots & \leq \frac{1}{2}\|b_{n-1}^*\| \\ 0 & 0 & 0 & \dots & \|b_n^*\| \end{pmatrix}$$

δ -LLL-редуцированные базисы

- Второе условие: в подматрице

$$\begin{pmatrix} \|b_i^*\| & \mu_{i+1,i} \|b_i^*\| \\ 0 & \|b_{i+1}^*\| \end{pmatrix}$$

второй столбец почти такой же длины, как первый столбец:

$$\delta \|b_i^*\|^2 \leq \mu_{i+1,i}^2 \|b_i^*\|^2 + \|b_{i+1}^*\|^2.$$

- Есть обобщение, которое обобщает это требование на подматрицы $k \times k$, но нам бы сейчас хотя бы с базовым LLL разобраться.

δ -LLL-редуцированные базисы

- Если мы нашли δ -LLL-редуцированный базис, то:

$$\begin{aligned}\|b_n^*\|^2 &\geq \left(\delta - \frac{1}{4}\right) \|b_{n-1}^*\|^2 \geq \dots \geq \\ &\geq \left(\delta - \frac{1}{4}\right)^{n-1} \|b_1^*\|^2 = \left(\delta - \frac{1}{4}\right)^{n-1} \|b_1\|^2.\end{aligned}$$

- Значит, для любого i

$$\begin{aligned}\|b_1\| &\leq \left(\delta - \frac{1}{4}\right)^{-\frac{i-1}{2}} \|b_i^*\| \leq \left(\delta - \frac{1}{4}\right)^{-\frac{n-1}{2}} \|b_i^*\|, \text{ и} \\ \|b_1\| &\leq \left(\delta - \frac{1}{4}\right)^{-\frac{n-1}{2}} \min_i \|b_i^*\| \leq \left(\delta - \frac{1}{4}\right)^{-\frac{n-1}{2}} \lambda_1(L).\end{aligned}$$

δ -LLL-редуцированные базисы

- Получили:

$$\|b_1\| \leq \left(\delta - \frac{1}{4}\right)^{-\frac{n-1}{2}} \lambda_1(L).$$

- Например, для $\delta = \frac{3}{4}$

$$\|b_1\| \leq 2^{(n-1)/2} \|x\|.$$

- Иначе говоря, если мы получим LLL-редуцированный базис, мы сразу решим (с точностью $(\delta - \frac{1}{4})^{-\frac{n-1}{2}}$) задачу поиска кратчайшего вектора.

Алгоритм L^3

- Алгоритм L^3 или LLL: Lenstra, Lenstra, Lovasz (1982).
- Задача — вычислить LLL-редуцированный базис.
 - 1 Вычислить b_1^*, \dots, b_n^* .
 - 2 Редукция.
 - 1 Для всех $2 \leq i \leq n$ и всех $i-1 \geq j \geq 1$

$$b_i := b_i - c_{ij} b_j, \quad \text{где } c_{ij} = \left\lfloor \frac{\langle b_i, b_j^* \rangle}{\langle b_j^*, b_j^* \rangle} \right\rfloor.$$

- 3 Обмен значениями.
 - 1 Если для какого-то i

$$\delta \|b_i^*\|^2 > \|\mu_{i+1,i} b_i^* + b_{i+1}^*\|^2,$$

то поменять местами b_i с b_{i+1} и вернуться к шагу 1.

- 4 Выдать b_1, \dots, b_n .

Алгоритм L^3

- Очевидно, что если обмена не произошло, то второе требование выполняется.
- А редукция обеспечивает первое условие: она вычитает столбец за столбцом так, чтобы оставалось $\leq \frac{1}{2}\|b_i^*\|$.
- Перед началом редукции:

$$\begin{pmatrix} \|b_1^*\| & * & * & \dots & * \\ 0 & \|b_2^*\| & * & \dots & * \\ 0 & 0 & \|b_3^*\| & \dots & * \\ \vdots & \vdots & & & \vdots \\ 0 & 0 & 0 & \dots & * \\ 0 & 0 & 0 & \dots & \|b_n^*\| \end{pmatrix}$$

Алгоритм L^3

- Очевидно, что если обмена не произошло, то второе требование выполняется.
- А редукция обеспечивает первое условие: она вычитает столбец за столбцом так, чтобы оставалось $\leq \frac{1}{2}\|b_i^*\|$.
- Первый шаг: $i = 2, j = 1$:

$$\begin{pmatrix} \|b_1^*\| & \leq \frac{1}{2}\|b_1^*\| & * & \dots & * \\ 0 & \|b_2^*\| & * & \dots & * \\ 0 & 0 & \|b_3^*\| & \dots & * \\ \vdots & \vdots & & & \vdots \\ 0 & 0 & 0 & \dots & * \\ 0 & 0 & 0 & \dots & \|b_n^*\| \end{pmatrix}$$

Алгоритм L^3

- Очевидно, что если обмена не произошло, то второе требование выполняется.
- А редукция обеспечивает первое условие: она вычитает столбец за столбцом так, чтобы оставалось $\leq \frac{1}{2} \|b_i^*\|$.
- Второй шаг: $i = 3, j = 2$:

$$\begin{pmatrix} \|b_1^*\| & \leq \frac{1}{2} \|b_1^*\| & * & \dots & * \\ 0 & \|b_2^*\| & \leq \frac{1}{2} \|b_2^*\| & \dots & * \\ 0 & 0 & \|b_3^*\| & \dots & * \\ \vdots & \vdots & & & \vdots \\ 0 & 0 & 0 & \dots & * \\ 0 & 0 & 0 & \dots & \|b_n^*\| \end{pmatrix}$$

Алгоритм L^3

- Очевидно, что если обмена не произошло, то второе требование выполняется.
- А редукция обеспечивает первое условие: она вычитает столбец за столбцом так, чтобы оставалось $\leq \frac{1}{2} \|b_i^*\|$.
- Третий шаг: $i = 3, j = 1$:

$$\begin{pmatrix} \|b_1^*\| & \leq \frac{1}{2} \|b_1^*\| & \leq \frac{1}{2} \|b_1^*\| & \dots & * \\ 0 & \|b_2^*\| & \leq \frac{1}{2} \|b_2^*\| & \dots & * \\ 0 & 0 & \|b_3^*\| & \dots & * \\ \vdots & \vdots & & & \vdots \\ 0 & 0 & 0 & \dots & * \\ 0 & 0 & 0 & \dots & \|b_n^*\| \end{pmatrix}$$

Алгоритм L^3

- Очевидно, что если обмена не произошло, то второе требование выполняется.
- А редукция обеспечивает первое условие: она вычитает столбец за столбцом так, чтобы оставалось $\leq \frac{1}{2} \|b_i^*\|$.
- И так далее; после всех шагов:

$$\begin{pmatrix} \|b_1^*\| & \leq \frac{1}{2} \|b_1^*\| & \leq \frac{1}{2} \|b_1^*\| & \dots & \leq \frac{1}{2} \|b_1^*\| \\ 0 & \|b_2^*\| & \leq \frac{1}{2} \|b_2^*\| & \dots & \leq \frac{1}{2} \|b_2^*\| \\ 0 & 0 & \|b_3^*\| & \dots & \leq \frac{1}{2} \|b_3^*\| \\ \vdots & \vdots & & & \vdots \\ 0 & 0 & 0 & \dots & \leq \frac{1}{2} \|b_{n-1}^*\| \\ 0 & 0 & 0 & \dots & \|b_n^*\| \end{pmatrix}$$

Алгоритм L^3

- Очевидно, что если обмена не произошло, то второе требование выполняется.
- А редукция обеспечивает первое условие: она вычитает столбец за столбцом так, чтобы оставалось $\leq \frac{1}{2} \|b_i^*\|$.
- Формально: после редукции

$$\begin{aligned} |\mu_{ij}| &= \left| \frac{\langle b_i, b_j^* \rangle}{\langle b_j^*, b_j^* \rangle} \right| = \left| \frac{\langle b_i - c_{ij} b_j, b_j^* \rangle}{\langle b_j^*, b_j^* \rangle} \right| = \\ &= \left| \frac{\langle b_i, b_j^* \rangle}{\langle b_j^*, b_j^* \rangle} - \left[\frac{\langle b_i, b_j^* \rangle}{\langle b_j^*, b_j^* \rangle} \right] \frac{\langle b_j, b_j^* \rangle}{\langle b_j^*, b_j^* \rangle} \right| = \left| \frac{\langle b_i, b_j^* \rangle}{\langle b_j^*, b_j^* \rangle} - \left[\frac{\langle b_i, b_j^* \rangle}{\langle b_j^*, b_j^* \rangle} \right] \right| \leq \frac{1}{2}, \end{aligned}$$

$$\text{т.к. } \langle b_j, b_j^* \rangle = \langle b_j^*, b_j^* \rangle.$$

Алгоритм L^3

- Осталось доказать, что LLL — полиномиальный алгоритм.
- Две части: количество итераций и каждая итерация. Про каждую итерацию не будем.
- Чтобы ограничить количество итераций, можно рассмотреть *потенциал* базиса решётки: для базиса $B = \{b_1, \dots, b_m\}$ потенциал — это

$$\mathcal{D}_B = \prod_{i=1}^n \mathcal{D}_{B,i} = \prod_{i=1}^n \det L_i = \prod_{i=1}^n \|b_1^*\| \|b_2^*\| \dots \|b_i^*\|.$$

- Здесь L_i — решётка, порождаемая b_1, \dots, b_i .

Алгоритм L^3

- До работы алгоритма, т.к. $\|b_i^*\| \leq \|b_i\|$, потенциал можно оценить как

$$\mathcal{D}_B \leq \left(\max_i \|b_i\| \right)^{\frac{n(n-1)}{2}}.$$

- Заметим, что $\log \mathcal{D}_B$ полиномиален от длины входа $(\max\{n, \log \|b_i\|\})$.

Алгоритм L^3

- От редукций \mathcal{D}_B не убывает, т.к. b_i^* не меняются.
- Когда мы меняем местами b_i и b_{i+1} , из всех $\det L_k$ меняется только $\det L_i$, и

$$\begin{aligned}\frac{\mathcal{D}'_B}{\mathcal{D}_B} &= \frac{\det L'_i}{\det L_i} = \frac{\det L(b_1, \dots, b_{i-1}, b_{i+1})}{\det L(b_1, \dots, b_i)} = \\ &= \frac{\prod_{j=1}^{i-1} \|b_j^*\| \cdot \|\mu_{i+1,i} b_i^* + b_{i+1}^*\|}{\prod_{j=1}^{i-1} \|b_j^*\| \cdot \|b_i^*\|} = \frac{\|\mu_{i+1,i} b_i^* + b_{i+1}^*\|}{\|b_i^*\|} \leq \sqrt{\delta}.\end{aligned}$$

- Значит, итераций может быть не больше $\log_{\frac{1}{\sqrt{\delta}}} \mathcal{D}_B \leq \frac{1}{\log \frac{1}{\sqrt{\delta}}} \frac{n(n-1)}{2} \log(\max_j \|b_j\|)$.

Алгоритм L^3 : итоги

- Краткий итог: мы получили алгоритм с константным параметром δ , который умеет решать задачу поиска кратчайшего вектора с точностью $(\delta - \frac{1}{4})^{-\frac{n-1}{2}}$.
- Можно рассмотреть $\delta = \frac{1}{4} + (\frac{3}{4})^{\frac{n}{n-1}}$ и получить точность $(\frac{2}{\sqrt{3}})^n$.
- Есть ещё алгоритм Шнорра, который достигает субэкспоненциальной точности: $2^{O(\frac{n(\log \log n)^2}{\log n})}$.
- Теперь поговорим о том, зачем всё это нужно.

Сведение subset sum к решётке

- Начнём с того, что обещали: применим LLL к задаче subset sum.
- Для этого нужно свести задачу subset sum к задаче поиска кратчайшего вектора в некоторой решётке.
- Рассмотрим задачу subset sum: даны числа $\{a_1, \dots, a_n\}$ и число s .

Сведение subset sum к решётке

- Положим $m = \lceil \frac{1}{2}\sqrt{n} \rceil$ и запишем матрицу

$$\begin{pmatrix} 1 & 0 & 0 & \dots & 0 & ma_1 \\ 0 & 1 & 0 & \dots & 0 & ma_2 \\ 0 & 0 & 1 & \dots & 0 & ma_3 \\ \vdots & \vdots & & & \vdots & \\ 0 & 0 & 0 & \dots & 1 & ma_n \\ \frac{1}{2} & \frac{1}{2} & \frac{1}{2} & \dots & \frac{1}{2} & ms \end{pmatrix}$$

- Рассмотрим решётку L , порождённую строками этой матрицы b_1, b_2, \dots, b_{n+1} .

Сведение subset sum к решётке

- Если (x_1, \dots, x_n) — решение subset sum, т.е. $\sum_{i=1}^n a_i = s$, то вектор $y = \sum_{i=1}^n x_i b_i - b_{n+1}$ лежит в L .
- Но у него $y_i = \pm \frac{1}{2}$ для $1 \leq i \leq n$, и $y_{n+1} = 0$. Иначе говоря, он короткий, и LLL может его найти.

Сведение subset sum к решётке

- LLL не обязан его находить, но для задач низкой *плотности* он его с большой вероятностью находит.
- Плотность задачи subset sum (и вообще задачи о рюкзаке):

$$d = \frac{n}{\max_j \log_2 a_j}.$$

- Известно, что если $d < 0,9408$, у LLL хорошие шансы.
- А в криптосистеме Меркле-Хеллмана плотность должна быть меньше $\frac{n}{n - \log n}$, потому что в противном случае будут наборы чисел с одинаковыми суммами, и не получится однозначно декодировать. Почему так, кстати?
- Более того, запутывание умножением уменьшает плотность.

Поиск корней многочленов

- Другое применение LLL-алгоритма — поиск корней многочленов.
- Рассмотрим целое число N и унитарный многочлен $f \in \mathbb{Z}_N[x]$ степени d .
- Тогда при помощи LLL мы можем найти все корни f в \mathbb{Z}_N , по модулю не превосходящие $B = N^{\frac{1}{d}}$.
- Важно: мы *не знаем* разложения N на множители; если бы знали, могли бы проще найти корни.
- Мы сейчас докажем упрощённую версию, для $B = cN^{\frac{2}{d(d+1)}}$.

Поиск корней многочленов

- Чтобы это сделать, рассмотрим многочлен

$$f(x) = x^d + a_{d-1}x^{d-1} + \dots + a_1x + a_0.$$

- Заметим, что если вдруг все коэффициенты f сильно ограничены:

$$|a_i B^i| < \frac{N}{d+1}, \quad 0 \leq i \leq d-1,$$

то мы можем найти все интересующие нас корни. Почему?

Поиск корней многочленов

- Чтобы это сделать, рассмотрим многочлен

$$f(x) = x^d + a_{d-1}x^{d-1} + \dots + a_1x + a_0.$$

- Заметим, что если вдруг все коэффициенты f сильно ограничены:

$$|a_i B^i| < \frac{N}{d+1}, \quad 0 \leq i \leq d-1,$$

то мы можем найти все интересующие нас корни. Почему?

- Потому что тогда для $|x| \leq B$

$$|f(x)| \leq \sum_{i=0}^d |a_i B^i| < N,$$

и можно искать просто обычные корни многочлена, что легко.

Поиск корней многочленов

- Конечно, нам так не повезёт. Но мы можем попробовать сделать так, чтобы нам повезло.
- Мы хотим найти такой многочлен g , что он имеет те же корни, что и f , но при этом маленькие коэффициенты.
- Маленькие коэффициенты — то есть *короткий вектор* коэффициентов... осталось придумать решётку.

Поиск корней многочленов

- Рассмотрим множество многочленов

$$\mathcal{Z} = \{N, Nx, Nx^2, \dots, Nx^{d-1}, f(x)\}.$$

- Все корни f также являются корнями любой их линейной комбинации (мы корни ищем по модулю N).
- Рассмотрим решётку из столбцов матрицы

$$L = \begin{pmatrix} N & 0 & 0 & \dots & 0 & a_0 \\ 0 & BN & 0 & \dots & 0 & Ba_1 \\ 0 & 0 & B^2N & \dots & 0 & B^2a_2 \\ \vdots & \vdots & & & \vdots & \\ 0 & 0 & 0 & \dots & B^{d-1}N & B^{d-1}a_{d-1} \\ 0 & 0 & 0 & \dots & 0 & B^d \end{pmatrix}$$

Поиск корней многочленов

- Мы ищем короткий вектор в этой решётке при помощи LLL-алгоритма.
- Определитель решётки:

$$\det L = N \cdot BN \cdot \dots \cdot B^{d-1}N \cdot B^d = N^d B^{d(d+1)/2}.$$

- После применения LLL мы найдём такой v , что

$$\|v\| \leq O(\lambda_1(L)) \leq O\left((\det L)^{\frac{1}{d+1}}\right) = O\left(N \frac{B^{d/2}}{N^{\frac{1}{d+1}}}\right),$$

где константы под O зависят только от d (размерности решётки).

- Это для $B < c_1(d) N^{\frac{2}{d(d+1)}}$ уже получится меньше, чем $\frac{N}{d+1}$.

Поиск корней многочленов

- Иначе говоря, мы нашли такой многочлен

$$g(x) = b_d x^d + \dots + b_1 x + b_0,$$

что у g те же корни, что у f , но $|b_i B^i| < \frac{N}{d+1}$.

- Теперь мы его просто решим, и всё.
- Чтобы улучшить оценку, нужно улучшить множество многочленов; самое общее — для некоторого h рассмотреть

$$\mathcal{Z} = \left\{ N^{h-i-1} f(x)^i x^j \mid 0 \leq j < d, 0 \leq i < h \right\}.$$

- Это для медленно растущей функции h и даст $B = N^{1/d}$.

Атаки на RSA с малой экспонентой

- Теперь применим всё это к криптографии.
- Вспомним RSA. Мы говорили, что если Алиса разошлёт одно и то же m трём Бобам с разными N_i и экспонентой e^i , то получится система уравнений $m^3 \equiv c_i \pmod{N_i}$, из неё по китайской теореме об остатках $m^3 \equiv c \pmod{N_1 N_2 N_3}$, и так как $m < N_i$, то последний корень можно просто извлечь.

Атаки на RSA с малой экспонентой

- Алиса может попробовать избежать этой участи, дописав в сообщение ID Боба, например, в начале. Тогда Бобу; она пошлёт $(m + 2^l ID_i)^3$, где l — длина сообщения.
- Рассмотрим даже чуть обобщённую ситуацию — у Боба есть многочлен g , являющийся частью публичного ключа (N, g) , и Алиса посылает ему $g(m) \pmod{N}$; в нашей ситуации $g(m) = (m + 2^l ID_i)^3$.

Атаки на RSA с малой экспонентой

- Пусть Бобов несколько: есть несколько N_1, \dots, N_k (взаимно простых) и несколько g_1, \dots, g_k .
- Пусть есть одно сообщение $m < N_{\min} = \min_i N_i$, и Чарли получил $g_i(m) \equiv c_i \pmod{N_i}$ для всех $1 \leq i \leq k$.
- Тогда, если $k \geq \deg(g_i)$, Чарли сможет найти m .

Атаки на RSA с малой экспонентой

- Для этого определим $h_i = g_i - c_i$; нам нужно найти общий корень m : $h_i(m) \equiv 0 \pmod{N_i}$.
- По китайской теореме об остатках, совместим h_i в общий h :

$$h(x) = \sum_{i=1}^k T_i h_i(x) \pmod{N_1 N_2 \dots N_k}.$$

- Но так как

$$M < N_{\min} \leq (N_1 N_2 \dots N_k)^{\frac{1}{k}} \leq (N_1 N_2 \dots N_k)^{\frac{1}{d}},$$

мы можем применить LLL-алгоритм и найти корень.

Outline

- 1 Решётки на службе криптоанализа
 - Общие сведения
 - Алгоритм LLL
 - Применения алгоритма LLL в криптоанализе
- 2 Решётки на службе криптографии
 - Идея
 - Конструкция хеш-функций Ajtai
 - Конструкция криптосистемы Ajtai-Dwork

О сложности

- Мы всё время говорим о том, что стойкость криптосистем основана на *сложности* решения тех или иных задач.
- Но о какой сложности идёт речь?
- Нам до сих пор требовалась *криптографическая сложность*: нужно было, чтобы враг не умел решать сложную задачу ни на какой доле входов, вообще никак.

О сложности

- А хотелось бы, чтобы требовалась сложность *в худшем случае*.
- Например, чтобы сделать утверждение «если $P \neq NP$, то криптосистема стойкая», надо как-то связать стойкость именно со сложностью в худшем случае.
- Мы сейчас поговорим об идеях того, как связать криптографическую стойкость со сложностью в худшем случае.

История

- Ajtai, 1996: хеш-функции, стойкость которых основана на сложности в худшем случае задачи n^c -приближения поиска кратчайшего вектора в решётке (shortest vector problem, SVP).
- Т.е. если кто-то умеет обращать функцию из этого семейства, он умеет полиномиально аппроксимировать кратчайшие векторы во всех решётках.

История

- Ajtai, Dwork, 1998: *криптосистема*, стойкость которой основана на сложности в худшем случае.
- Задача, правда, менее ясная: *unique-SVP*, т.е. поиск кратчайшего вектора v в решётке размерности n *при условии*, что нет непараллельных ему векторов длины меньше $n^{\delta} \|v\|$.

Идея конструкции Ajtai

- Мы хотим построить семейство хеш-функций. По сути конструкция — вариант subset sum.
- Рассмотрим m векторов $a_1, \dots, a_m \in \mathbb{Z}_q^n$. Определим $f_{a_1, \dots, a_m} : \{0, 1\}^m \rightarrow \{0, 1\}^{n \log q}$ как

$$f_{a_1, \dots, a_m}(b_1, \dots, b_m) = \sum_{i=1}^m b_i a_i \pmod{q}.$$

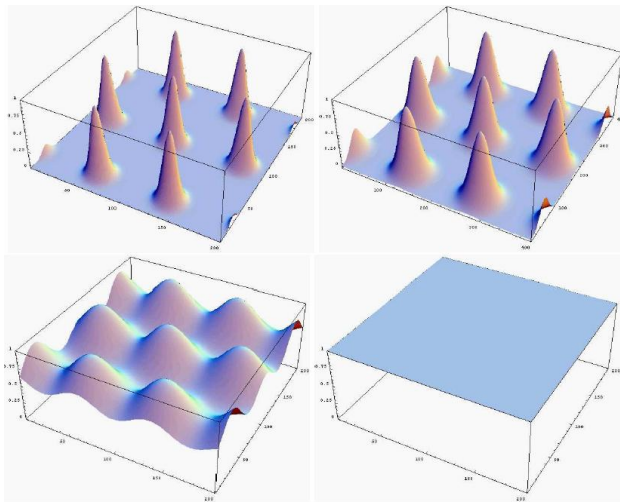
Наше семейство хеш-функций — это набор функций $\mathcal{F} = \{f_{a_1, \dots, a_m} \mid a_1, \dots, a_m \in \mathbb{Z}_q^n\}$.

- Чтобы семейство было стойким, нужно, чтобы для случайной функции $f \in \mathcal{F}$ никакой полиномиальный алгоритм не мог бы со значительной вероятностью найти коллизию, т.е. такие $x \neq y$, что $f(x) = f(y)$.

Идея доказательства

- Мы хотим доказать, что если такой алгоритм существует, то, используя его как оракула, мы сможем найти кратчайший вектор в любой решётке.
- “Такой алгоритм” умеет по случайным $a_1, \dots, a_m \in \mathbb{Z}_q^n$ находить со значительной вероятностью их линейную комбинацию, равную 0, т.е. $b_1, \dots, b_m \in \{0, \pm 1\}$:
$$\sum_{i=1}^m b_i a_i \equiv 0 \pmod{q}.$$
- Идея сведения — рассмотреть решётку как распределение вероятностей и добавлять в неё нормально распределённый шум.
- Если добавить достаточно много шума, решётка станет неотличима от равномерного распределения.

Шум в решётке



Идея доказательства

- Пусть мы нашли такой параметр s , что с ним распределение решётки достаточно близко к равномерному.
- Тогда давайте возьмём несколько векторов по этому нормальному распределению (они будут короткими) и приведём их к параллелепипеду решётки.
- Получатся практически равномерно распределённые точки в параллелепипеде.
- Теперь разобьём решётку на q^n ячеек и поставим в соответствие каждой ячейке элемент \mathbb{Z}_q^n .
- Наши исходные векторы теперь соответствуют числам из \mathbb{Z}_q^n .

Идея доказательства

- Применим наш алгоритм, который умеет находить коллизии, к этим числам. Он найдёт линейную комбинацию, равную 0, т.е. линейную комбинацию наших векторов, очень близкую к вектору, равному нулю «по модулю решётки», т.е. очень близкую к вектору решётки.
- И, так как исходные векторы были короткими, этот вектор решётки тоже будет коротким.
- Остались только сложные статистические соображения.


Идея конструкции Ajtai-Dwork

- Сама конструкция криптосистемы тоже очень несложная.
- Рассмотрим большое число N ; секретный ключ — просто число $h \in [\sqrt{N}, 2\sqrt{N})$.
- Публичный ключ — набор из $m = O(\log N)$ чисел $0 \leq a_1, \dots, a_m \leq N - 1$, которые «достаточно близки» к числам, кратным N/h (N не обязательно делится на h), плюс номер одного из этих чисел a_{i_0} , которое достаточно близко к *нечётному* кратному N/h .

Идея конструкции Ajtai-Dwork

- Код нуля: сумма случайного подмножества $\{a_i\}$.
- Код единицы: сумма случайного подмножества $\{a_i\}$ плюс ещё $\lfloor a_{i_0}/2 \rfloor$.
- Декодирование: поделить число на N/h . Если получилось мало, выдать 0, если много — выдать 1.
- Декодирование корректно: все a_i близки к тому, чтобы делиться на N/h , а $a_{i_0}/2$ не близко.
- Доказательство стойкости, конечно, гораздо сложнее.

Спасибо за внимание!

- Lecture notes и слайды будут появляться на моей homepage:
<http://logic.pdmi.ras.ru/~sergey/>
- Присылайте любые замечания, решения упражнений, новые численные примеры и прочее по адресам:
sergey@logic.pdmi.ras.ru, snikolenko@gmail.com
- Заходите в ЖЖ  [smartnik](#).