Криптография с открытым ключом II

Сергей Николенко

Криптография — CS Club, осень 2009

Outline

Цели теоретической криптографии

- Построить надёжную криптосистему.

Цели теоретической криптографии

- Построить надёжную криптосистему.
  - Невозможно, пока не докажем, что P$\neq$NP.

Цели теоретической криптографии

- Построить надёжную криптосистему.
  - Невозможно, пока не докажем, что P$\neq$NP.
- Построить криптосистему, надёжность которой основана на NP-трудной задаче.

Цели теоретической криптографии

- Построить надёжную криптосистему.
  - Невозможно, пока не докажем, что P$\neq$NP.
- Построить криптосистему, надёжность которой основана на NP-трудной задаче.
  - Не умеем.

Цели теоретической криптографии

- Построить надёжную криптосистему.
  - Невозможно, пока не докажем, что P$\neq$NP.
- Построить криптосистему, надёжность которой основана на NP-трудной задаче.
  - Не умеем.
- Построить криптосистему, надёжность которой основана на сложной, но вряд ли NP-трудной задаче.

Цели теоретической криптографии

- Построить надёжную криптосистему.
  - Невозможно, пока не докажем, что P$\neq$NP.
- Построить криптосистему, надёжность которой основана на NP-трудной задаче.
  - Не умеем.
- Построить криптосистему, надёжность которой основана на сложной, но вряд ли NP-трудной задаче.
  - Это мы уже делали: криптосистема Рабина, RSA.

Цели теоретической криптографии

- Построить надёжную криптосистему.
  - Невозможно, пока не докажем, что P$\neq$NP.
- Построить криптосистему, надёжность которой основана на NP-трудной задаче.
  - Не умеем.
- Построить криптосистему, надёжность которой основана на сложной, но вряд ли NP-трудной задаче.
  - Это мы уже делали: криптосистема Рабина, RSA.
- Построить криптосистему, надёжность которой *кажется* основанной на NP-трудной задаче.

Цели теоретической криптографии

- Построить надёжную криптосистему.
  - Невозможно, пока не докажем, что P$\neq$NP.
- Построить криптосистему, надёжность которой основана на NP-трудной задаче.
  - Не умеем.
- Построить криптосистему, надёжность которой основана на сложной, но вряд ли NP-трудной задаче.
  - Это мы уже делали: криптосистема Рабина, RSA.
- Построить криптосистему, надёжность которой *кажется* основанной на NP-трудной задаче.
  - Этим мы займёмся сейчас.

Цели теоретической криптографии

- Иначе говоря, всё, что мы можем попытаться сделать — это *сделать вид*, что противнику надо решить NP-трудную задачу.
- Сейчас мы рассмотрим одну из таких криптосистем; она основана на *кодах, исправляющих ошибки*.

Суть

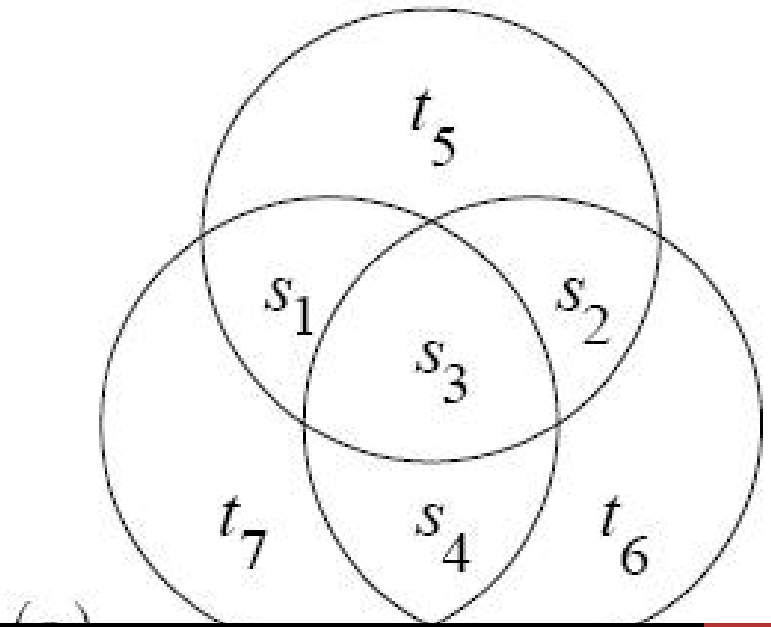
- Что такое коды, исправляющие ошибки (error–correcting codes)?
- Это коды, которые умеют даже по неправильному кодовому слову достаточно часто выдавать правильное сообщение.
- *Задача декодирования*: по сигналу понять, какое кодовое слово передавалось.
- Говорят, что код *исправляет t ошибок*, если он корректно декодирует любой сигнал, искажённый в $\leq t$ битах.

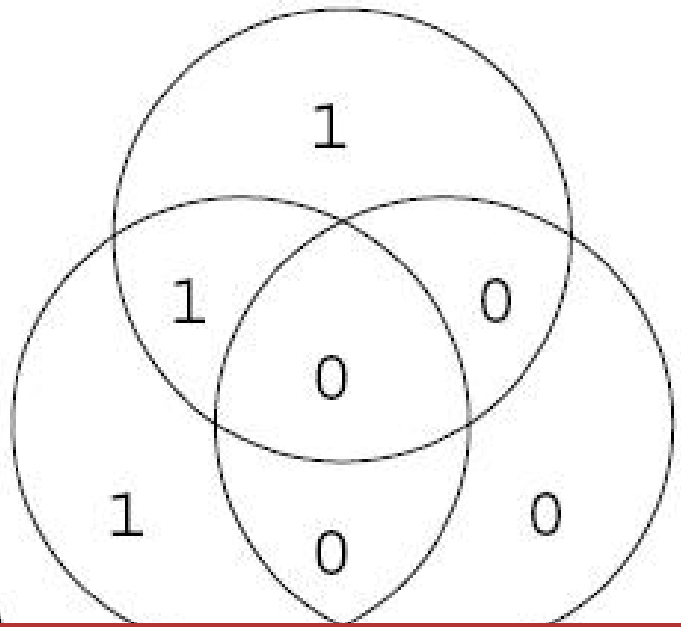
Линейные коды

- Пусть у нас блок размера k переходит в блок размера n при кодировании ($n > k$, разумеется).
- Предположим, что все биты кодового слова являются линейными функциями от битов сообщения (parity checks).
- Такие коды называются *линейными*.
- Эквивалентное определение — код линейный, если сумма кодовых слов является кодовым словом.

Пример

- Широко известен код Хэмминга $(7,4)$ (на 4 бита сообщения 7 битов сигнала).
- Линейные функции — parity от битов сообщения по следующему правилу: они равны сумме попадающих в соответствующий круг битов сообщения.





Пример

- Главное свойство этого кода — то, что кодовые слова отличаются друг от друга как минимум в трёх битах.
- Другой способ достичь того же — повторять каждый бит три раза.
- Повторять гораздо менее эффективно: сообщение увеличивается в 3 раза, а в коде Хэмминга всего в $\frac{7}{4}$.
- Но есть и тонкий положительный эффект; какой?

Линейный код в общем виде

- Кодовое слово получается в виде $c = sG$, где G — матрица, называющаяся *генератором* кода.
- Если сначала сообщение повторяется, то $G = [\, I_k \; P \,]$
- Например, для $(7, 4)$–кода Хэмминга

$$G = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{pmatrix}.$$

Синдромы

- Для декодирования используются так называемые *синдромы*. Синдром — это разница между реальным сигналом и сигналом, вычисленным на основании полученных битов сообщения.
- Если $c = sG$, и $G = [\, I_k \; P \,]$, то синдром $z = Hr$, где $H = [\, -P^\top \; I_{n-k} \,]$. Например, для $(7, 4)$–кода Хэмминга

$$H = \begin{pmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 \end{pmatrix}.$$

Синдромы

- Для всякого валидного кодового слова c $Hc^\top = 0$:

$$Hc^\top = H(sG)^\top = HG^\top s^\top = \begin{pmatrix} -P^\top & I_{n-k} \end{pmatrix} \begin{pmatrix} I_k \\ P \end{pmatrix} s^\top = 0.$$

- Если же полученное слово r неправильное, и в нём есть шум, то результат $Hr^\top$ помогает найти n и декодировать.

Постановка задачи декодирования

- Получаемый вектор r — это сумма кодового слова и шума:

$$r = sG + n^\top.$$

- Задача декодирования синдрома — это задача поиска такого вектора шума n, что

$$Hn = z.$$

- Если код исправляет t ошибок, то среди слов, отличающихся от s в $\leq t$ местах, есть только одно решение этого уравнения.
- Проще говоря, для кода, исправляющего t ошибок, в шаре радиуса t вокруг каждого кодового слова других кодовых слов не встречается.

Сложность задачи декодирования

- Для кода Хэмминга решить задачу декодирования несложно.
- Однако это не всегда так.
- Для абстрактного линейного кода нужно решать задачу MLD (maximum likelihood decoding): по данному вектору с ошибкой найти ближайшее кодовое слово из кода.

Декодирование и сечения

- Любопытный пример кода: рассмотрим граф $G = (V, E)$ и код $C_G \subseteq \{0,1\}^{|E|}$, где каждое кодовое слово — это сечение графа.
- Почему это линейный код?

Декодирование и сечения

- Любопытный пример кода: рассмотрим граф $G = (V, E)$ и код $C_G \subseteq \{0,1\}^{|E|}$, где каждое кодовое слово — это сечение графа.
- Почему это линейный код?
- Почему декодировать его будет NP-трудно?

Декодирование и сечения

- Любопытный пример кода: рассмотрим граф $G = (V, E)$ и код $C_G \subseteq \{0,1\}^{|E|}$, где каждое кодовое слово — это сечение графа.
- Почему это линейный код?
- Мы доказали, что декодировать произвольный линейный код — NP-трудная задача. Её-то мы и будем маскировать.

Goppa codes

- Валерий Денисович Гоппа — первым (1981) осознал связь между алгебраической геометрией и теорией кодирования.
- Вообще говоря, коды Гоппы — это линейные коды, порождённые несингулярными проективными кривыми над конечными полями.
- Но нам нужен только конкретный частный случай, который можно описать без глубокой теории.

Двоичные коды Гоппы

- Зафиксируем число m (обычно 10, 11 или 12), число $n \leq 2^m$ (часто берут $n = 2^m$) и $t \in [2, \frac{2^m-1}{m}]$.
- Выберем последовательность $a_1, \ldots, a_n \in \mathbb{F}_{2^m}$ (если $n = 2^m$, то это просто все элементы в лексикографическом порядке).
- Определим многочлен

$$h = \prod_{i=1}^{n} (x - a_i) \in \mathbb{F}_{2^m}[x].$$

- Зафиксируем неприводимый многочлен g степени t.

Двоичные коды Гоппы

- Тогда множество кодовых слов — это:

$$\Gamma = \Gamma(a_1 \ldots, a_n, g) = \left\{ c \in \mathbb{F}_2^n : \sum_{i=1}^{n} c_i \frac{h}{x - a_i} \equiv 0 \pmod{g} \right\}.$$

- Иначе говоря, множество Γ — это ядро отображения–«синдрома» $\mathbb{F}_2^n \to \mathbb{F}_{2^m}^t$, которое действует как

$$c = (c_0, \ldots, c_{n-1}) \mapsto$$

$$\mapsto b_0 + b_1 x + \ldots + b_{t-1} x^{t-1} \equiv \sum_{i=1}^{n} c_i \frac{h}{x - a_i} \pmod{g}.$$

- Значит, размерность кода — по меньшей мере $n - mt$, и мы получили $(n, \geq n - mt)$-код.

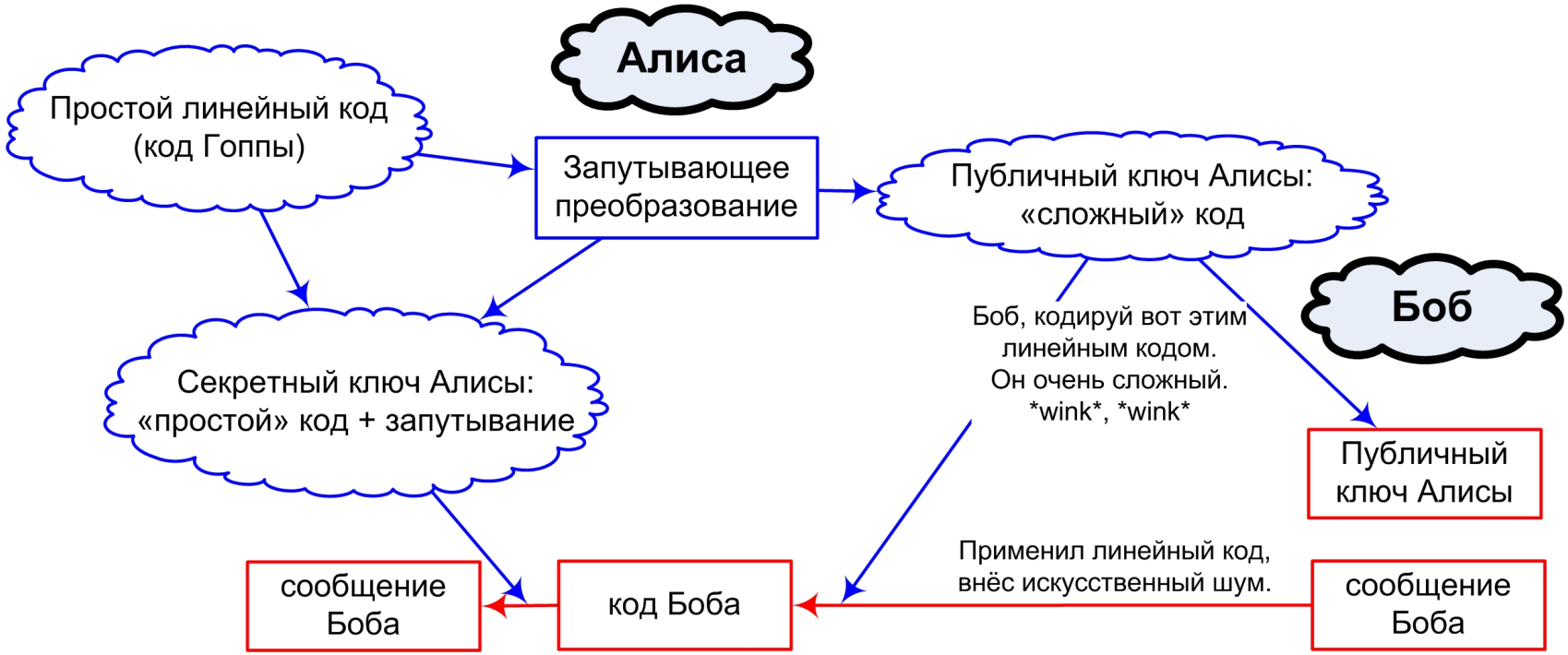
Двоичные коды Гоппы

- Многочлены $\frac{h}{x-a_1} \pmod g, \ldots, \frac{h}{x-a_n} \pmod g$ — это, если их рассмотреть как векторы битов, строки матрицы синдромов.
- В общем, коды Гоппы, обладают хорошими свойствами как коды, и для них есть полиномиальные алгоритмы кодирования и декодирования.

Упражнение. Докажите, что код Гоппы исправляет t ошибок.

Криптосистема МакЭлиса: обзор

- А теперь собственно криптосистема МакЭлиса (McEliece cryptosystem).



Криптосистема МакЭлиса: ключи

- Генерация ключа: фиксируем k, n, t. Каждый участник делает следующее.
  1. Выбрать матрицу кода G размера $k \times n$ для (n, k)-линейного кода, исправляющего t ошибок, для которого известен эффективный алгоритм декодирования (например, код Гоппы).
  2. Выбрать случайную невырожденную матрицу S размера $k \times k$.
  3. Выбрать случайную матрицу перестановки P размера $n \times n$.
  4. Выдать как публичный ключ t и $\widehat{G} = SGP$; секретный ключ — (S, G, P).

Криптосистема МакЭлиса: [де]кодирование

- Алгоритм кодирования (даны t, $\hat{G}$ и сообщение m).
  1. Представить сообщение как строку длины k.
  2. Выбрать случайный вектор шума z длины n с $\leq t$ единичками.
  3. Закодировать $c = m\hat{G} + z$.
- Алгоритм декодирования (даны c и ключ (S, G, P)).
  1. Вычислить $\hat{c} = cP^{-1}$.
  2. Декодировать то, что получилось, алгоритмом декодирования кода; получится $\hat{m}$.
  3. Вычислить $m = \hat{m}S^{-1}$.

Упражнение. Доказать, что алгоритм декодирования корректен.

Криптосистема МакЭлиса: о стойкости

- По сути: мы хотим попросить врага решить NP-трудную задачу.
- Но не можем.
- Поэтому мы берём простой частный случай NP-трудной задачи, а потом «запутываем» его так, чтобы врагу было не догадаться, какой это частный случай.
- Для многих кодов криптосистему МакЭлиса взломали; для кодов Гоппы пока нет.

Криптосистема МакЭлиса: о стойкости

- Атака: найти матрицу G' кода Гоппы, соответствующую данной $\hat{G}$. Этого делать пока никто не умеет.
- Другая атака:
  - выбрать k столбцов из $\hat{G}$, ограничить на них матрицу и векторы; будет $c_k = m_k \hat{G}_k + z_k$.
  - предположить, что $z_k = 0$;
  - решить напрямую систему $c_k = m_k \hat{G}_k$, найти m_k.
- Вероятность того, что $z_k = 0$, очень мала.
- Только в 2008 году смогли успешно атаковать криптосистему МакЭлиса для $n = 1024$, $k = 524$, $t = 50$.

Subset Sum

- И снова будем маскировать NP-полную задачу, на этот раз задачу о рюкзаке, точнее, subset sum.
- Можно ли из заданного набора чисел $\{b_1, \ldots, b_n\}$ по данному числу s выбрать такое подмножество $\{b_{i_1}, \ldots, b_{i_l}\}$, что $\sum_{j=1}^{l} b_{i_j} = s$? И что это будет за подмножество?
- Это — трудная (NP-полная) задача, её в общем виде решить трудно.

Subset Sum

- Есть приближённые алгоритмы, но точного нету. Тривиальный требует время $O(n2^n)$ (перебрать все подмножества n чисел).
- Менее тривиальный работает за время и память $O(n2^{n/2})$ [Horowitz, Sahni, 1974].
  1. Разбить n элементов на две половины по $\frac{n}{2}$.
  2. В каждой половине посчитать все возможные суммы; получатся два вектора длины $2^{n/2}$ каждый.
  3. Отсортировать векторы.
  4. Теперь идти по первому вектору сверху вниз, по второму снизу вверх: если сумма больше s, то смещаемся вниз по первому вектору, если меньше, то поднимаемся вверх по второму вектору. Так мы не пропустим пару, сумма которой равна s, если такая есть.
- Более эффективных точных алгоритмов не известно.

Subset Sum

- Давайте рассмотрим частный случай. Последовательность чисел $(b_1, \ldots, b_n)$ называется *супервозрастающей* (superincreasing), если для любого $2 \leq i \leq n$

$$b_i > \sum_{j=1}^{i-1} b_j.$$

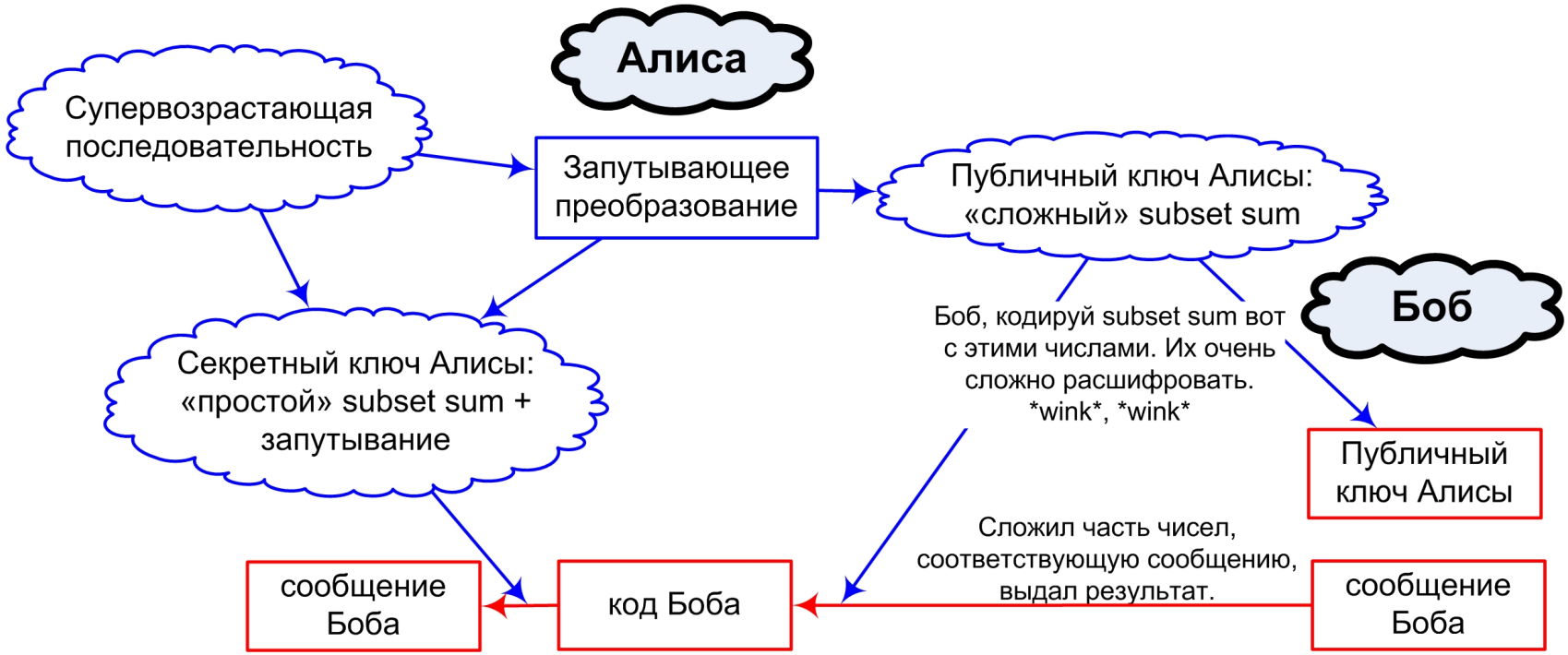
- Можете ли вы предложить более эффективный алгоритм решения subset sum, если известно, что $(b_1, \ldots, b_n)$ образуют супервозрастающую последовательность?

Subset Sum

- Да, алгоритм нехитрый: взять самое большое число, которое умещается в s, вычесть его, повторить.
- Итак, у нас есть задача, решить которую в общем виде очень трудно, а в частном случае — легко.
- Значит, для криптографии мы можем попробовать лёгкий частный случай зашифровать как сложную задачу.

Криптосистема Меркле-Хеллмана: обзор

- Криптосистема Меркле-Хеллмана (Merkle-Hellman cryptosystem).



Криптосистема Меркле-Хеллмана: ключи

- Алгоритм генерации ключей.
  1. Зафиксировать n. Выбрать супервозрастающую последовательность $(b_1, \ldots, b_n)$ и модуль M, такой, что $M > b_1 + \ldots + b_n$.
  2. Выбрать случайное число $W < M$, взаимно простое с M.
  3. Выбрать случайную перестановку $\pi \in S_{\{1,2,\ldots,n\}}$.
  4. Вычислить $a_i = Wb_{\pi(i)} \pmod{M}$, $i = 1, \ldots, n$.
  5. Теперь публичный ключ — $(a_1, \ldots, a_n)$, секретный — $(\pi, M, W, b_1, \ldots, b_n)$.

Криптосистема Меркле-Хеллмана: [де]кодирование

- Алгоритм кодирования (вход: сообщение m, ключ $(a_1, \ldots, a_n)$).
  1. Представить m как битовую строку $m_1 m_2 \ldots m_n$.
  2. Вычислить шифр $c = m_1 a_1 + m_2 a_2 + \ldots + m_n a_n$.
- Алгоритм декодирования (вход: шифр c, ключ $(\pi, M, W, b_1, \ldots, b_n)$).
  1. Вычислить $d = W^{-1} c \pmod{M}$.
  2. Найти такие $r_1, \ldots, r_n$, что $d = b_1 r_1 + \ldots + b_n r_n$.
  3. Теперь биты сообщения — это $m_i = r_{\pi(i)}$, $i = 1, 2, \ldots, n$.

Криптосистема Меркле-Хеллмана

- Иначе говоря, мы маскируем простой частный случай сложной задачи как сложную задачу, и врагу должно быть сложно распознать, что эта задача простая.
- Но, может быть, враг на самом деле может как-то распознать, что это простая задача, и решить её?
- Для криптосистемы Меркле-Хеллмана — да, может. Многие криптосистемы, основанные на задаче о рюкзаке, ломаются при помощи сведения к задачам о решётках.
- Сейчас мы их и рассмотрим.

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**.