Лекция 11

Введение в криптографию с открытым ключом

11.1 Постановка задачи

В криптографии существует множество задач, которые имеют широкую область применения. Это и закрытая передача данных по открытым каналам, и алгоритмы электронной подписи, и интерактивные доказательства с нулевым разглашением, и многие другие. Чтобы составить понятие об этой тематике, в этой лекции мы поговорим о так называемых «протоколах шифрования с открытым ключом». Задача состоит в следующем. Имеется читатель A и писатель B (таких писателей может быть много). Проблема заключается в том, чтобы передать данные от B к A в зашифрованном виде, чтобы никто из тех, кто перехватит данные на пути из B к A, не смог расшифровать их. В начале работы протокола у A и B нет никакой общей конфиденциальной информации. Канал между A и B все время открыт, то есть читать из него могут все желающие.

11.2 Криптосистема с открытым ключом[1]

Неформально, идея системы, которая бы позволила нам решить поставленную задачу, состоит в следующем. Первоначально читатель A создает пару ключей. Один ключ (`pub`) публикуется. Он общедоступен и называется открытым (публичным) ключом. Второй ключ (`pri`) читатель оставляет у себя и хранит в секрете. Этот ключ называется закрытым (личным, секретным, приватным). Алгоритм кодирования устроен так, что любой может закодировать сообщение, зная открытый ключ. Задача раскодирования сообщения легко решается, если известен закрытый ключ (`pri`). Если противник не знает этого ключа, то раскодировать такое сообщение ему будет крайне трудно; тем труднее, чем длиннее были выбраны ключи `pri` и `pub`.

Определение 11.1. С формальной точки зрения, криптосистема для шифрования с открытым ключом состоит из трех алгоритмов, каждый их которых может использовать случайные числа и заканчивает свою работу за время, математическое

[1] Public Key Encryption Scheme

ожидание которого полиномиально от длины входа. (Вероятность берется по используемым алгоритмом случайным битам, но не по входам алгоритма.)

Генератор ключей —

$$G : 1^k \mapsto (\mathtt{pub}, \mathtt{pri})$$

генерирует ключи «сложности» k (для простоты можно представлять себе, что k — «длина» ключа).

Шифровальщик —

$$E : (\mathtt{pub}, m) \mapsto \mathtt{code}.$$

Дешифровщик —

$$D : (\mathtt{pri}, \mathtt{code}) \mapsto m.$$

От шифровальщика и дешифровщика, мы, естественно, должны дополнительно потребовать, чтобы сообщения корректно расшифровывались, то есть для любого сообщения m вероятность события $D(E(m, \mathtt{pub}), \mathtt{pri}) \neq m$ не превосходит $1/2$ (естественно, мы можем уменьшить вероятность ошибки по приемлемой величины, повторив пересылку многократно). Здесь вероятность вновь берется по случайным битам, используемым алгоритмами D, E и G (заметим, что `pub` и `pri` представляют собой случайные переменные, порожденные G), но не по входам этих алгоритмов (т.е. не по m и не по k).

Замечание 11.1. Заметим, что мы ничего не сказали о *надежности* криптосистемы. Формальное определение понятия надежности слишком громоздко и мы отложим его до соответствующих спец. курсов. Неформально же можно сказать, что никакой полиномиальный по времени алгоритм не должен уметь со сколько-нибудь существенной вероятностью успеха расшифровывать сообщения, имея в распоряжении лишь `pub` и `code`; здесь вероятность уже берется не только по использованным случайным битам, но и по зашифрованным сообщениям m.

11.3 Пример: RSA

В качестве примера мы опишем самый простой и популярный алгоритм шифрования с открытым ключом, который называется RSA (по первым буквам фамилий его авторов: Rivest, Shamir, Adleman). Эта (наиболее простая) версия алгоритма, конечно, не является надежной, но она служит базой для его многочисленных модификаций, которые на самом деле применяются на практике.

11.3.1 Совсем простая версия

Итак выберем два очень больших (например, состоящих из не более, чем $k/2$ битов, где 1^k — вход генератора G) различных простых числа p, q. Вычислим их произведение $n = pq$. Выберем число e так, чтобы НОД$(e, \phi(n)) = 1$ (вспомним, что функция

Эйлера $\phi(n) = (p-1)(q-1)$ может быть легко вычислена *тем, кто знает* p *и* q). Далее, находим $d \in [1..n-1]$, такое, что $de \equiv 1 \pmod{\phi(n)}$. Оно, очевидно, существует и единственно в силу того, что НОД$(e, \phi(n)) = 1$, и может быть найдено при помощи алгоритма Евклида.

Задача 11.1. А подходящее e как найти?

Итак, мы описали генератор ключей G, который выдает публичный ключ $\mathtt{pub} = (e, n)$ и приватный ключ $\mathtt{pri} = (d, n)$. Разобьем сообщение на числа, не превосходящие n; достаточно определить E и D на таких «коротких» сообщениях. Функция кодирования

$$E((n, e), m) = m^e \bmod n.$$

Для декодирования мы просто возводим код в степень d:

$$D((n, d), c) = c^d \bmod n.$$

При этом по теореме Ферма-Эйлера[2]

$$D((n, d), E((n, e), x)) \equiv x^{ed} \equiv x^{k \cdot \phi(n)+1} \equiv x \pmod{n}.$$

Мы видим, что зная d, мы можем запросто раскодировать сообщение за полиномиальное время; однако, если d неизвестно, непонятно, как это можно было бы сделать.

11.3.2 Чуть более надежная версия

Сразу видно несколько прорех в приведенной выше «простой» версии RSA.

1. Плохо такой версией шифровать биты: при $m = 0$ и $m = 1$ расшифровать полученный код всегда тривиально (он равен m).

2. Повторяемость: одинаковые сообщения имеют одинаковые коды; чтобы соперник этим не воспользовался, придется каждый раз выбирать новый ключ.

3. Мультипликативность: произведение кодов есть код произведения. Если мы случайно знаем два закодированных сообщения, сможем расшифровать и их произведение (либо же надо каждый раз выбирать новый ключ).

Чуть более надежная версия выглядит так: разобьем сообщение на биты, и для шифрования одного бита b

- выберем случайное число r от 1 до $n/2 - 1$;

[2] На лекции меня «поймали» на применении теоремы Ферма-Эйлера в формулировке

Теорема. *Если разложение* n *на простые множители не содержит квадратов, то*

$$\forall x\ \forall k\ x^{k \cdot \phi(n)+1} \equiv x \pmod{n}.$$

(Заметим, что здесь нет условия на взаимную простоту x и n.) Я не обнаружил ее в книге И.М.Виноградова; вероятно, не появлялась она и в курсе теории чисел. Тем не менее, она верна. Ее доказательство является упражнением по теории чисел (она следует из малой теоремы Ферма для простых n).

- зашифруем $2r + b$.

Иначе говоря,

$$\begin{aligned}\widetilde{E}((n,e),b) &= E((n,e),2r+b),\\ \widetilde{D}((n,d),c) &= D((n,d),c) \bmod 2.\end{aligned}$$

Теперь шансы получить «простой» код мало зависят от самого сообщения — они зависят лишь от попавшихся нам случайных битов.

11.4 Односторонние функции

Увы, достаточные условия существования криптосистем с открытым ключом (именно, существование так называемых функций «с секретом») слишком громоздки и не уместились в этом курсе. (Более того, эти общие условия не влекут надежности конкретных криптосистем: например, RSA.) Однако мы можем сформулировать несколько необходимых условий.

Определение 11.2. *Односторонней функцией* называется функция

$$f : X \to Y,$$

вычислимая за полиномиальное время, обратная к которой не может быть вычислена[3] за полиномиальное время. (Обратная — любая функция

$$g : Y \to X,$$

такая, что $\forall x \in X \ f(g(f(x))) = f(x)$.)

Замечание 11.2. Можно считать, что $X, Y = \{0,1\}^*$, хотя бывают и односторонние функции, определенные не на всех строках.

Вторая компонента (`pub`) генератора ключей, очевидно, должна быть односторонней функцией от используемых генератором случайных битов[4] (в противном случае можно было бы вычислить по `pub` случайные биты, а по ним — `pri`). Это не единственная односторонняя функция, которую можно извлечь из криптосистемы.

Упражнение 11.1. А какие еще?

Определение 11.3. Различают *односторонние в наихудшем случае*[5] и *сильно односторонние*[6] функции.

Для односторонних в наихудшем случае «не может быть вычислена за полиномиальное время» означает «не существует полиномиального по времени алгоритма (не

[3] Более формальные варианты этого определения см. чуть ниже.

[4] Если при одном и том же k генератор время от времени использует разное количество случайных битов, строчку случайных битов можно дополнить «фиктивными» битами так, чтобы длина этой строки была лишь [монотонно строго возрастающей] функцией от k; длину ключа k можно вычислить по количеству этих битов.

[5] worst-case one-way

[6] strongly one-way

использующего случайные числа), который по $f(x)$ вычисляет некоторую обратную функцию $g(f(x))$ для *каждого* x».

Для сильно односторонних «не может» означает «для любого полиномиального по времени алгоритма A (использующего случайные числа), для любого полинома q, для достаточно больших k

$$\mathbf{P}\{x \mid f(A(f(x))) = f(x)\} < \frac{1}{q(|x|)},$$

где вероятность берется как по случайным битам, использованным алгоритмом A, так и по случайным $x \in X$ длины k.»

Замечание 11.3. Несложно показать, что $\mathbf{P} \neq \mathbf{NP}$ тогда и только тогда, когда существуют функции, односторонние в наихудшем случае.

Имеются следующие следствия (первое из них мы также не будем доказывать, так как не давали необходимых определений):

$\exists$ надежное шифрование с открытым ключом $\Rightarrow$
$\exists$ сильно односторонние функции $\Rightarrow$
$\exists$ односторонние в наихудшем случае функции $\Leftrightarrow$
$\mathbf{P} \neq \mathbf{NP}$.

Имеются ли в первых двух случаях обратные следствия, неизвестно.