Защита информации

Павел Юдаев

МГТУ им. Баумана, Кафедра ИУ-9

Москва, 2014

Раздел 10 - Управление ключами

Эквивалентные определения:

Опр.

Односторонняя функция (one way function) - это функция, для кот. $\exists A \in \mathrm{PT}$ вычисляющий ее для любого входного значения. Но $\nexists B \in \mathrm{PPT}$ - алгоритм обращения функции, дающий правильный ответ с не пренебр. малой вер-ю, т.е. $\nexists B \in \mathrm{PPT}: \ P[f(B(f(x))) = f(x)] < \varepsilon(n)$.

Опр.

Функция $f : \{0,1\}^* \to \{0,1\}^*$ является *односторонней функцией*, если она вычисляется за полиномиальное время на детерминированной машине Тьюринга, но не существует полиномиальной вероятностной машины Тьюринга, которая обращает эту функцию с более чем пренебр. малой вероятностью.

Предположительно, это - о.ф. (не доказано, что это - о.ф., но нет и свидетельств против):

- любая стойкая к коллизиям хэш функция
- умножение двух больших простых чисел одинаковой длины и факторизация результата
- возведение в степень и логарифм в конечном поле

Если о.ф. сущ., то $P \neq NP$.

Опр.

Односторонняя функция с секретом - при неизвестном секрете это о.ф. по обоим аргументам, при известном секрете $\exists B \in \mathrm{PT}$ - вычисляет обратную.

Предположительно, это - о.ф. с секретом:

- разложение числа на простые множители. Секрет - один из простых множителей.

Деление занимает полиномиальное время.

Важно! из опр. о.ф. следует, что она должна быть тяжело обращаема в подавл. большинстве случаев, а не в худшем случае. Это отличается от опр. NP-трудности, где сложность берется в худшем случае.

Пример

Пусть сущ. полиномиальный алгоритм, который для не пренебр. малой доли множества значений ф-ции f находит прообраз. Тогда f - не о.ф.
Однако, если удастся выделить подмножество множества определения функции f, для образа которого не известен полиномиальный алгоритм обращения с не пренебр. малой вероятностью угадать прообраз, то на нем f будет (кандидатом на) о.ф.!

Пример

Факторизация $n = pq$ - кандидат на о.ф. Факторизация $n = 2^q$ тривиальна.

(пояснить рисунком)

Раздел 10 - Управление ключами

Срок жизни сессионного ключа ограничен.
Цель: по одному ключу создать много ключей.

SK - первичный ключ (secret key) - случайно, равномерно.
F - ПСФ, CTX - контекст для процесса. Key derivation function:

$$KDF(SK, CTX, L) = F(SK, (CTX||0)), ..., F(SK, (CTX||L-1))$$

Пользователь $\Rightarrow$ не равномерное распр-е SK. Возможен подбор производного ключа - перебор по словарю из наиболее часто исп-ся SK (dictionary attack).

Добавим случайности.

Опр.

Соль (salt) - случайные (равномерно распред.) данные, которые подаются на вход односторонней функции вместе с секретным ключом для получения производных ключей или хэш-значения от секретного ключа.

HKDF (hash based KDF) - ф-ция создания производных ключей, основанная на хэш функции (RFC-5869).
H - стойкая к коллизиям хэш функция.

salt - значение р.р. с.в., фикс. публичная величина.
$k = H(salt, SK)$ - р.р. ключ (extract) и
$KDF(k, CTX, L) = F(k, (CTX||0)), ..., F(k, (CTX||L-1))$
(expand)

Современные ф-и: PBKFD2, bcrypt.

Раздел 10 - Управление ключами

Способы:
1) Храним пары логин-пароль открытым текстом - имеет доступ админ! Их возможно украсть.

2) Зашифруем. Ключ шифра на той же машине, и админ все равно знает все пароли. Не годится.

1-е, простое решение: хранить пары $(login, H(pwd))$, при вводе пароля сравнивать $H(input)$ с соотв. записью. Админ не знает наши пароли.

Атака перебором по словарю. Злоум-к:

- если он выкрадет файл, быстро узнает пароли по заранее вычисленным для всего словаря $H(entry)$
- если не сможет выкрасть файл, будет перебирать пароли из словаря, система будет вычислять $H(input)$, с высокой вер-ю найдется совпадение. (Наиболее частая атака.)

2-е решение: используем медленную хэш функцию. Храним $(login, H^{(c)}(pwd))$. Значение c - публичная величина.

Уязвимо к атаке перебором по словарю. Злоум-к:

- если выкрадет файл, быстро узнает пароли по заранее вычисленным для всего словаря $H^{(c)}(entry)$
- если не сможет выкрасть файл, будет перебирать пароли из словаря, система будет вычислять $H^{(c)}(input)$, с высокой вер-ю найдется совпадение. Атака займет больше времени, т.к. хэш функция выч. медленно (до 0.1 сек. польз-ль не заметит). Атаку ускоряет использование радужных таблиц (rainbow tables): управляемый перебор для данной х/ф $H^{(c)}$.

3-е, лучшее решение: используем медленную хэш функцию и соль. Храним $(login, salt, H^{(c)}(salt, pwd))$. Вычисляем $H^{(c)}(salt, input)$.

Для каждого польз-ля (лучше) своя соль.

- если зл-к не сможет выкрасть файл, будет перебирать пароли из словаря И для каждого значения пароля - все возможные варианты соли. Это очень долго и защищает от использования атаки с радужными таблицами.
- если злоум-к выкрадет файл, чтобы узнать пароли, необходимо заново выч. для всего словаря значения $H^{(c)}(salt, entry)$.
- если у польз-й разные соли, то разные х/ф для пользователей, поэтому атака на каждый пароль по отдельности! А не на все сразу.

Литература к лекции

нет